



南方科技大学

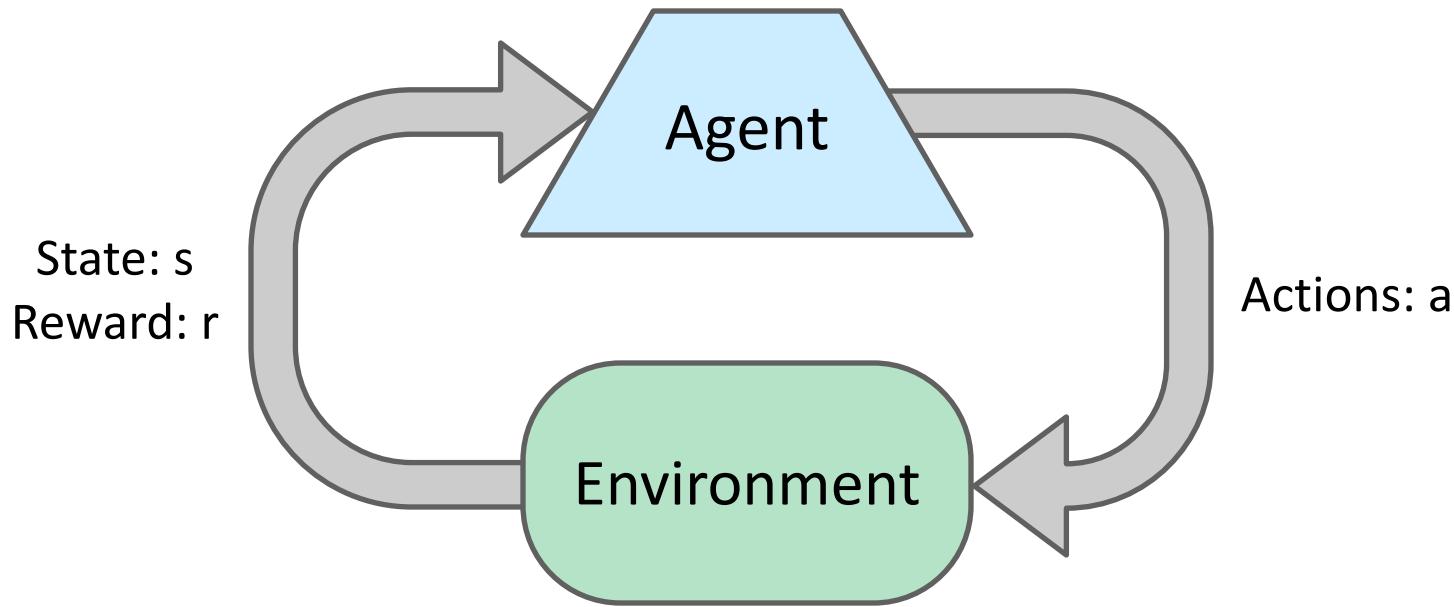
# MAT8034: Machine Learning

## Markov Decision Processes

Fang Kong

<https://fangkongx.github.io/Teaching/MAT8034/Spring2025/index.html>

# Reinforcement Learning



- Basic idea:
  - Receive feedback in the form of rewards
  - Transfer to the next state after taking an action
  - Must (learn to) act so as to **maximize expected rewards**
  - All learning is based on observed samples of outcomes!

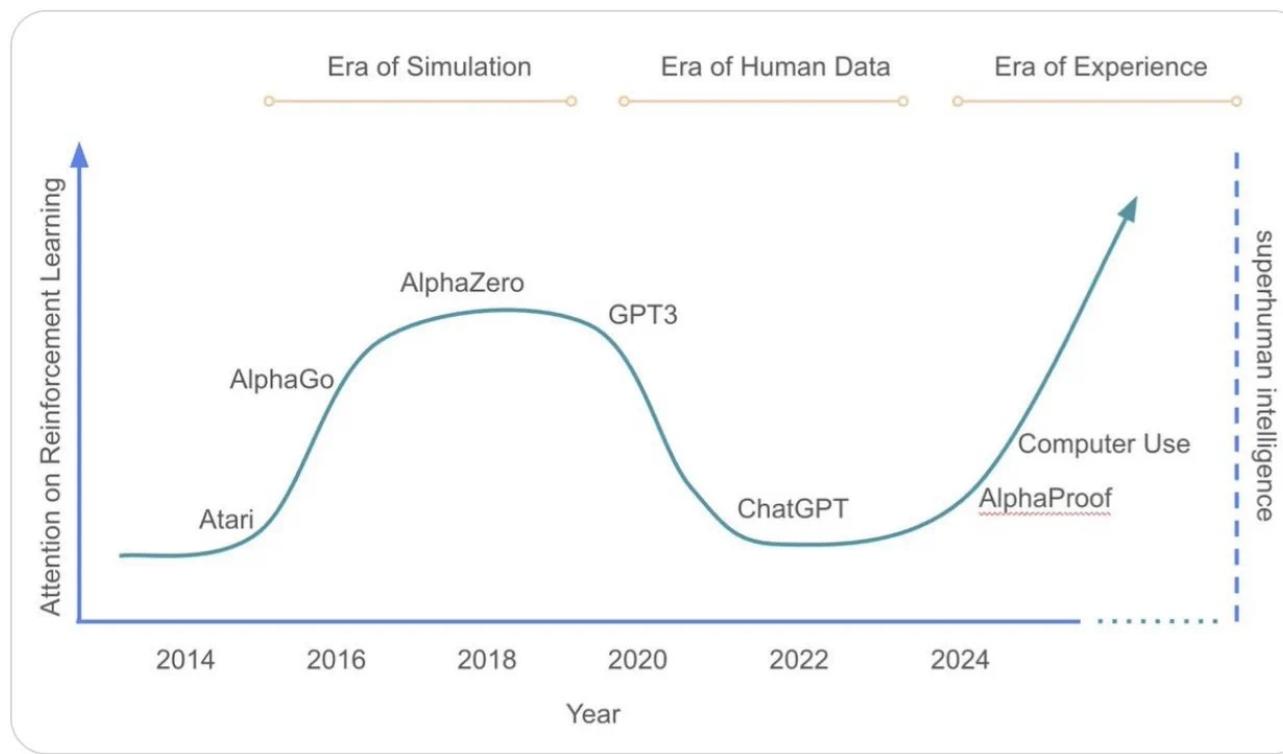


Maziyar PANAHY ✅

@MaziyarPanahi



"Welcome to the Era of Experience!" 🤖



# Example: Breakout (DeepMind)

---



# Example: AlphaGo (2016)

---

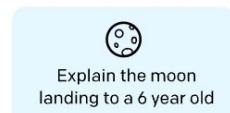


# Reinforcement Learning in ChatGPT

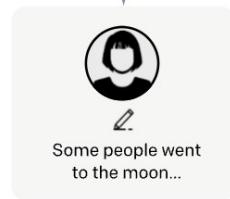
Step 1

**Collect demonstration data, and train a supervised policy.**

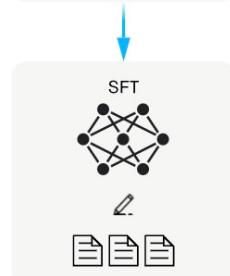
A prompt is sampled from our prompt dataset.



A labeler demonstrates the desired output behavior.



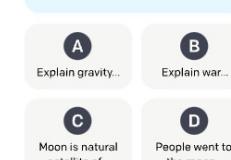
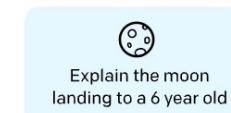
This data is used to fine-tune GPT-3 with supervised learning.



Step 2

**Collect comparison data, and train a reward model.**

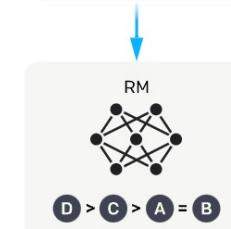
A prompt and several model outputs are sampled.



A labeler ranks the outputs from best to worst.



This data is used to train our reward model.



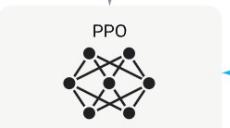
Step 3

**Optimize a policy against the reward model using reinforcement learning.**

A new prompt is sampled from the dataset.



The policy generates an output.



The reward model calculates a reward for the output.



The reward is used to update the policy using PPO.

$r_k$

# Reasoning Capability in LLMs

## DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning

DeepSeek-AI, Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z.F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, Aixin Liu, Bing Xue, Bingxuan Wang, Bochao Wu, Bei Feng, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, Damai Dai, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, Fulí Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Han Bao, Hanwei Xu, Haocheng Wang, Honghui Ding, Huajian Xin, Huazuo Gao, Hui Qu, Hui Li, Jianzhong Guo, Jia Shi Li, Jiawei Wang, Jingchang Chen, Jingyang Yuan, Junjie Qiu, Junlong Li, J.L. Cai, Jiaqi Ni, Jian Liang, Jin Chen, Kai Dong, Kai Hu, Kaige Gao, Kang Guan, Kexin Huang, Kuai Yu, Lean Wang, Lecong Zhang, Liang Zhao, Litong Wang, Liyue Zhang, Lei Xu, Leyi Xia, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Meng Li, Miaojun Wang, Mingming Li, Ning Tian, Panpan Huang, Peng Zhang, Qiancheng Wang, Qinyu Chen, Qiushi Du, Ruiqi Ge, Ruisong Zhang, Ruijie Pan, Runji Wang, R.J. Chen, R.L. Jin, Ruyi Chen, Shanghao Lu, Shangyan Zhou, Shanhua Chen, Shengfeng Ye, Shiyu Wang, Shuiping Yu, Shunfeng Zhou, Shuting Pan, S.S. Li et al. (100 additional authors not shown)

We introduce our first-generation reasoning models, DeepSeek-R1-Zero and DeepSeek-R1. DeepSeek-R1-Zero, a model trained via large-scale reinforcement learning (RL) without supervised fine-tuning (SFT) as a preliminary step, demonstrates remarkable reasoning capabilities. Through RL, DeepSeek-R1-Zero naturally emerges with numerous powerful and intriguing reasoning behaviors. However, it encounters challenges such as poor readability, and language mixing. To address these issues and further enhance reasoning performance, we introduce DeepSeek-R1, which incorporates multi-stage training and cold-start data before RL. DeepSeek-R1 achieves performance comparable to OpenAI-o1-1217 on reasoning tasks. To support the research community, we open-source DeepSeek-R1-Zero, DeepSeek-R1, and six dense models (1.5B, 7B, 8B, 14B, 32B, 70B) distilled from DeepSeek-R1 based on Qwen and Llama.

Subjects: Computation and Language (cs.CL); Artificial Intelligence (cs.AI); Machine Learning (cs.LG)

Cite as: arXiv:2501.12948 [cs.CL]

(or arXiv:2501.12948v1 [cs.CL] for this version)

<https://doi.org/10.48550/arXiv.2501.12948> ⓘ

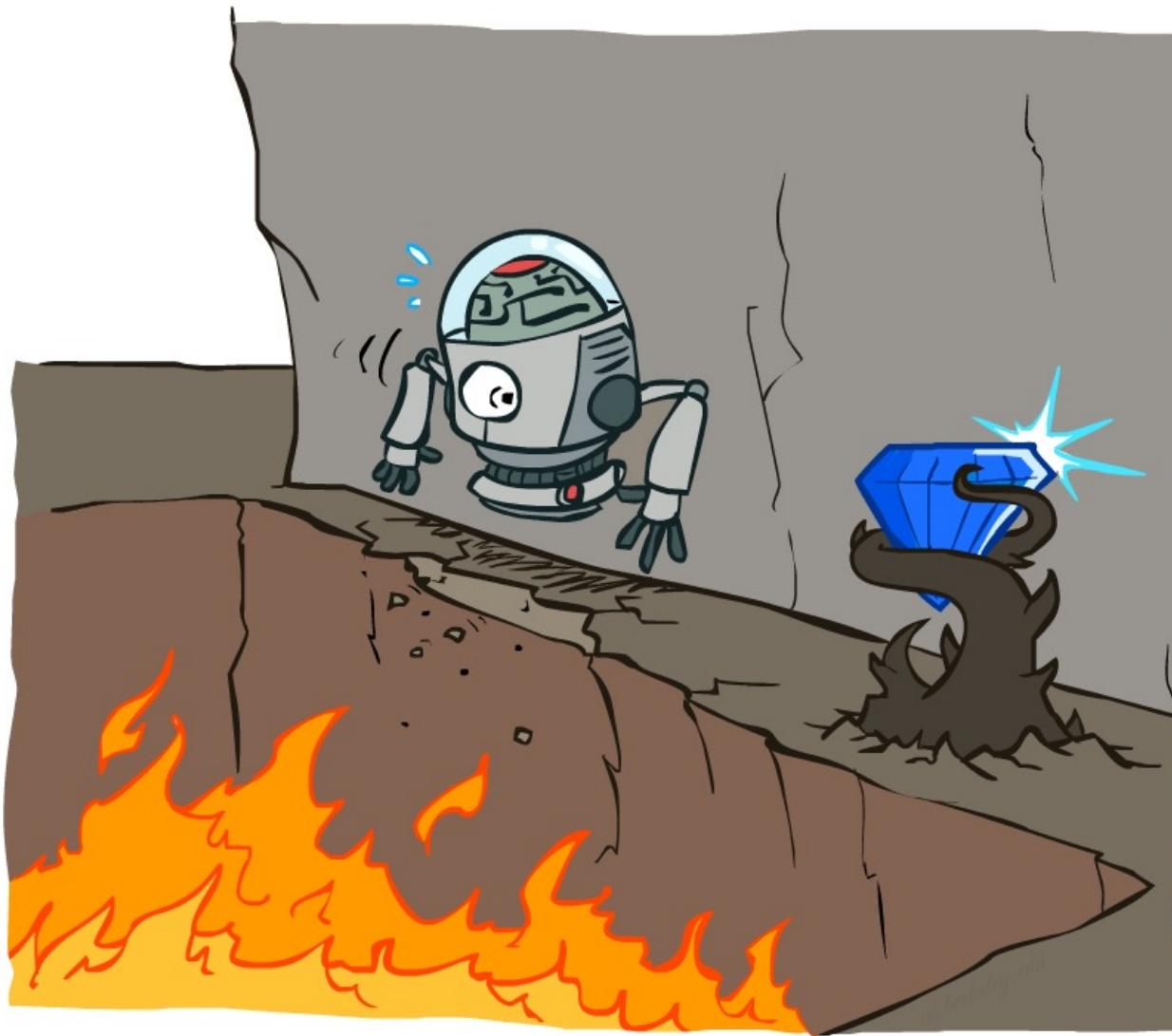
September 12, 2024 Release

## Learning to reason with LLMs

We are introducing OpenAI o1, a new large language model trained with reinforcement learning to perform complex reasoning. o1 thinks before it answers—it can produce a long internal chain of thought before responding to the user.

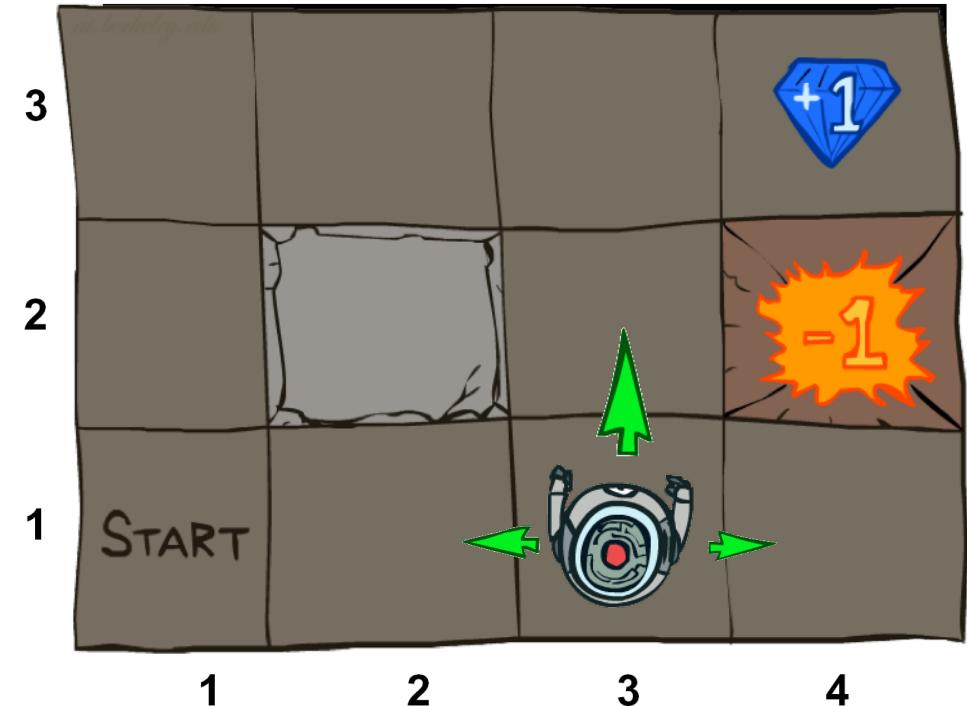
# MDPs: Actions + Search + Probabilities + Time

---



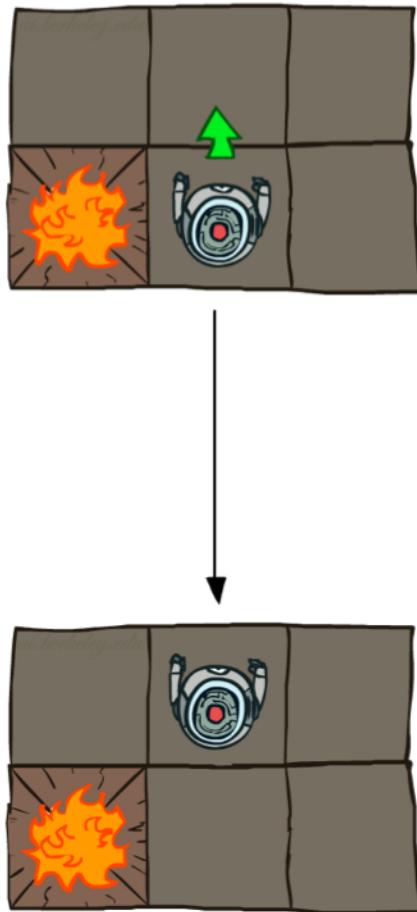
# Example: Grid World

- 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

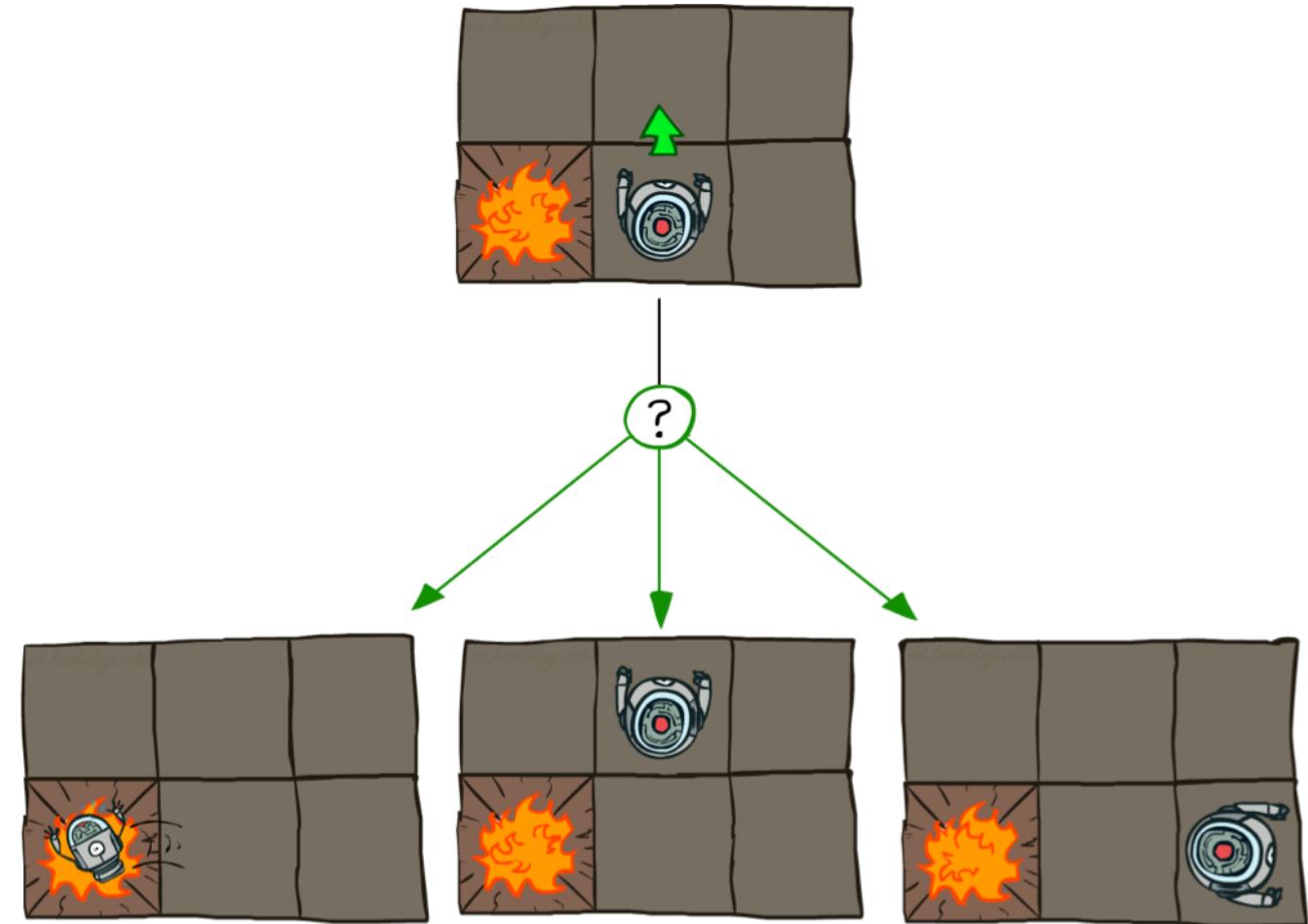


# Grid World Actions

Deterministic Grid World

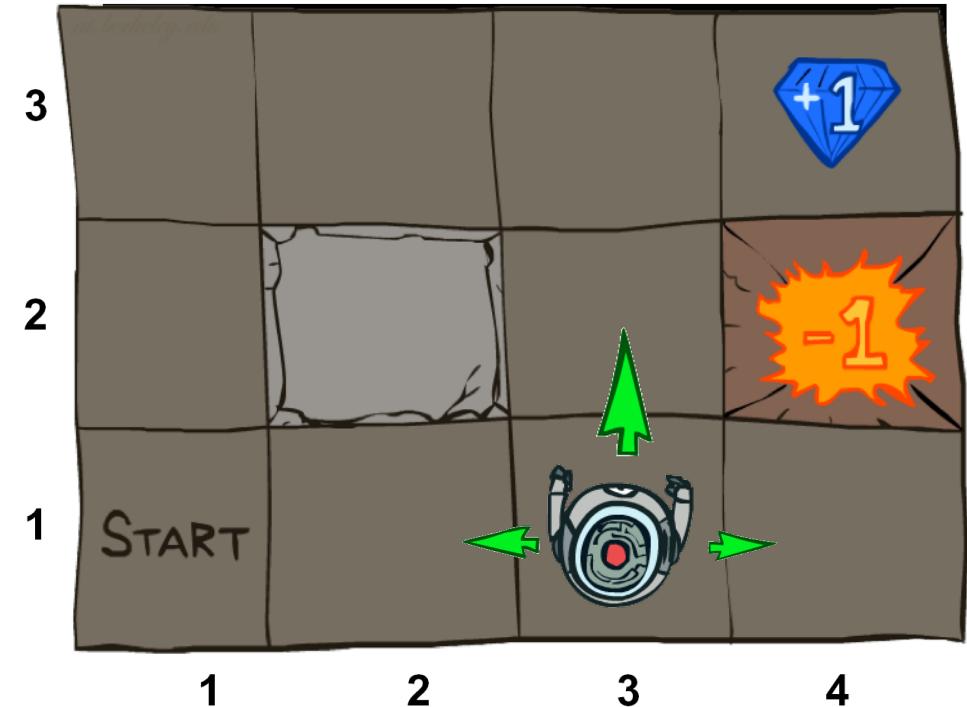


Stochastic Grid World



# 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



# 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

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

=

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

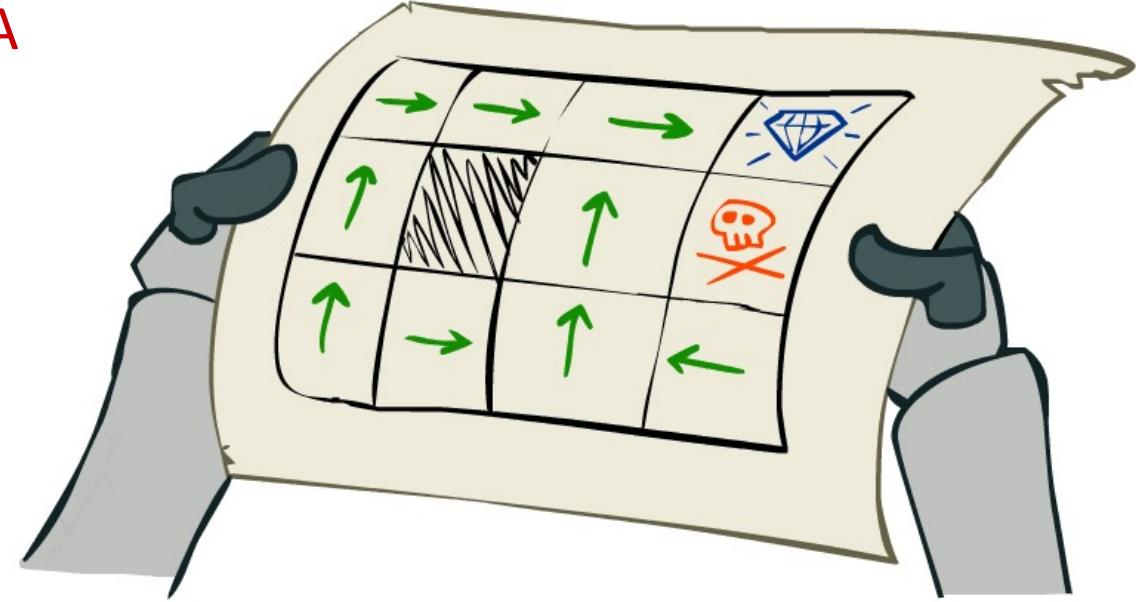


Andrey Markov  
(1856-1922)

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

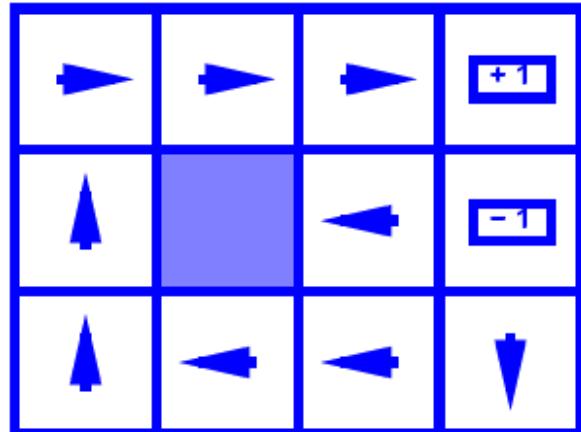
# Policies

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

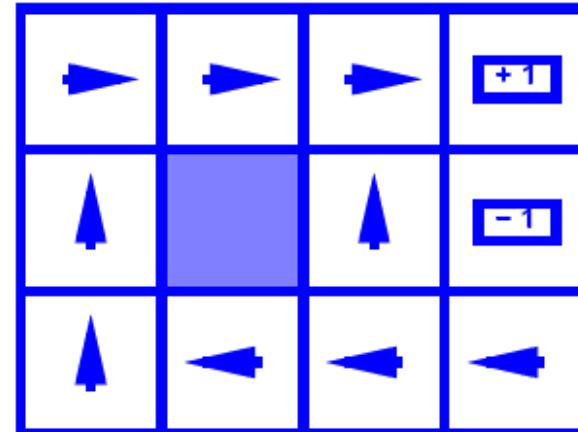


Optimal policy when  $R(s, a, s') = -0.03$   
for all non-terminals  $s$

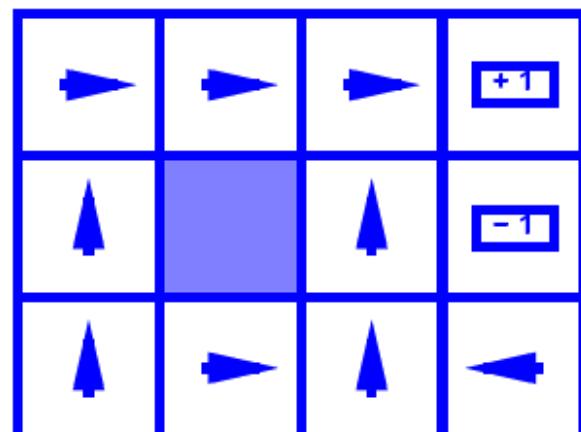
# Optimal Policies



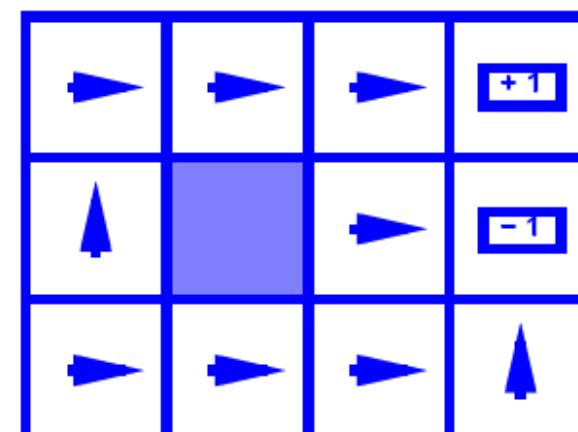
$$R(s) = -0.01$$



$$R(s) = -0.03$$



$$R(s) = -0.4$$



$$R(s) = -2.0$$

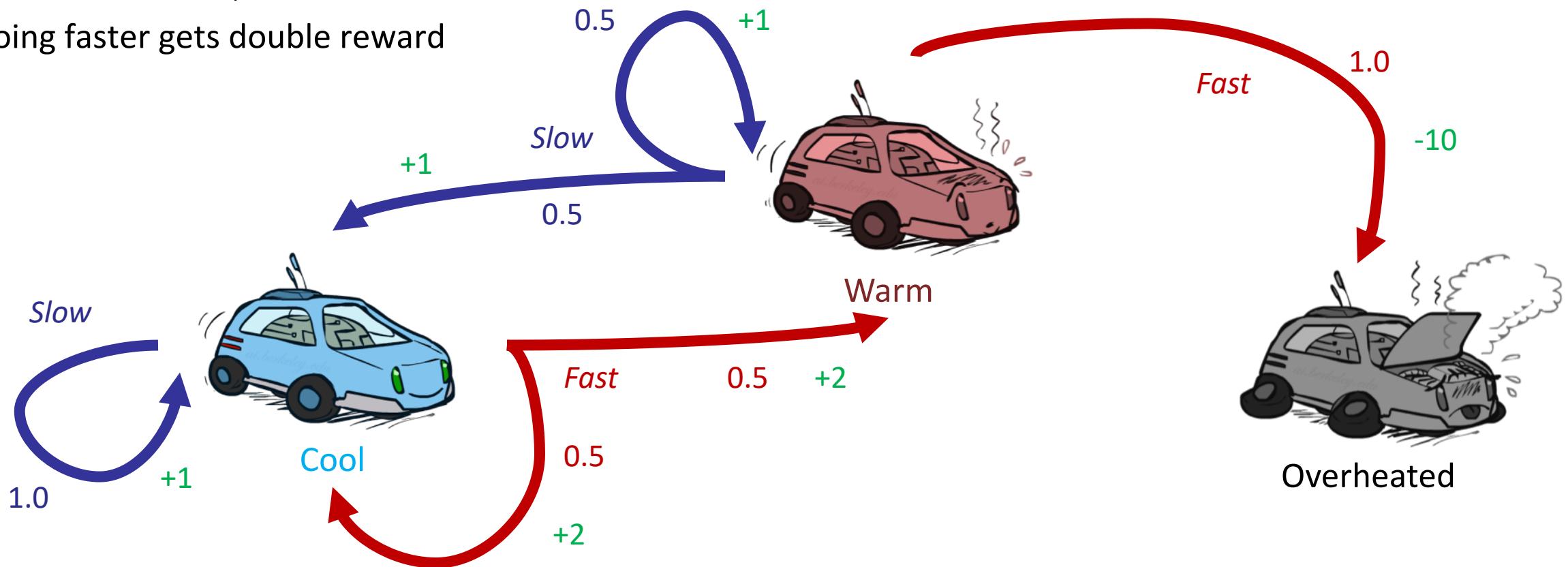
# Example: Racing

---



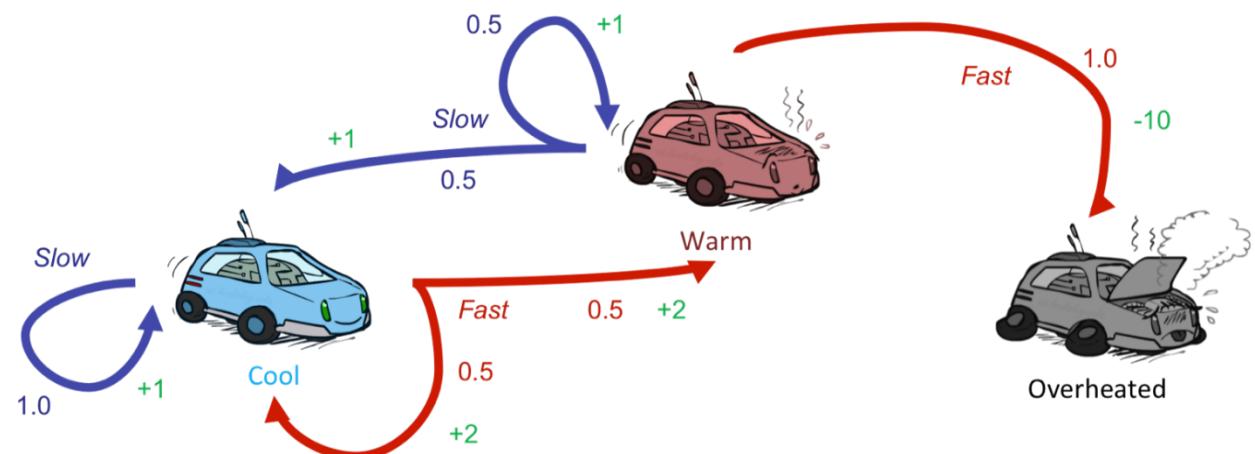
# Example: Racing

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

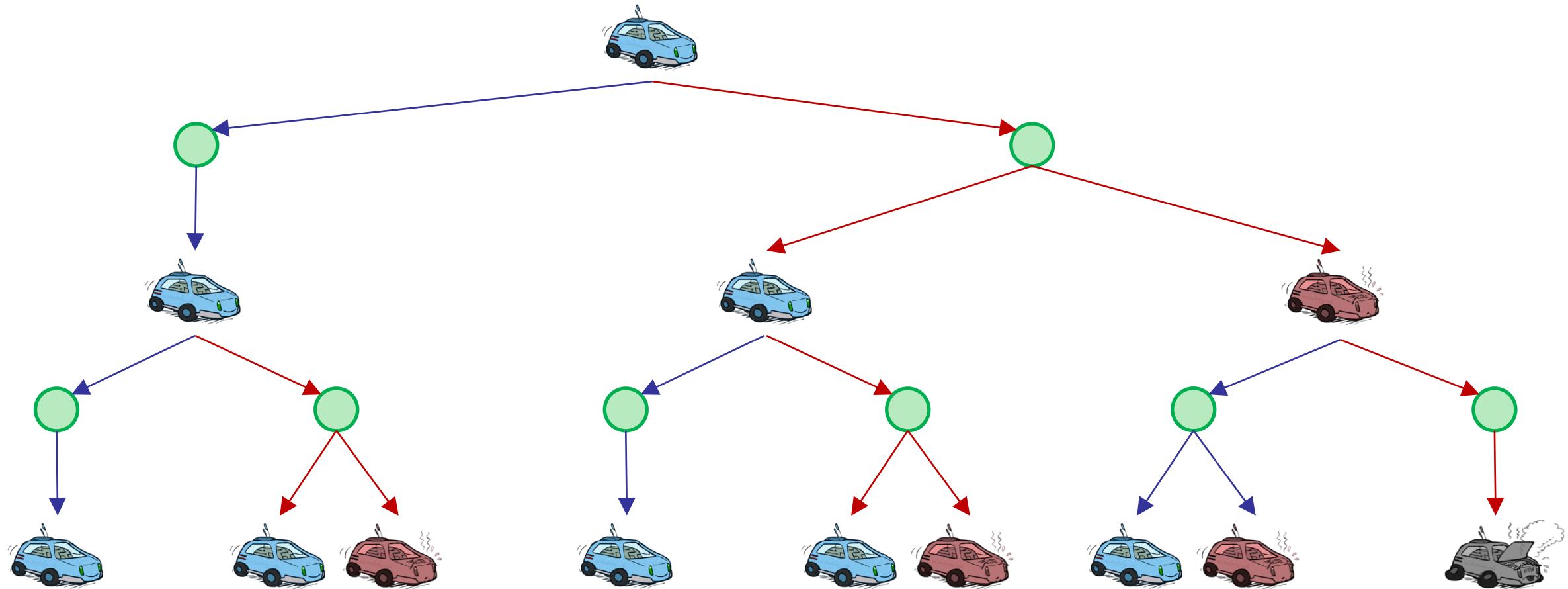


# Example: Racing

$s$	$a$	$s'$	$T(s,a,s')$	$R(s,a,s')$
	Slow		1.0	+1
	Fast		0.5	+2
	Fast		0.5	+2
	Slow		0.5	+1
	Slow		0.5	+1
	Fast		1.0	-10
	(end)		1.0	0

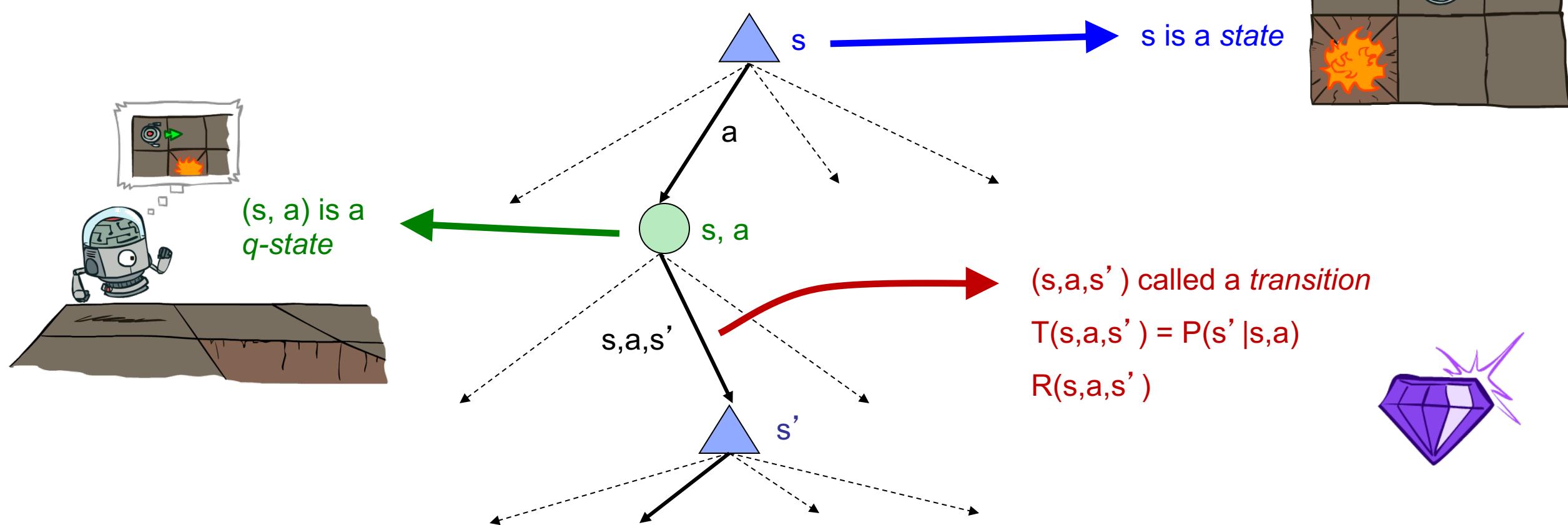


# Racing Search Tree



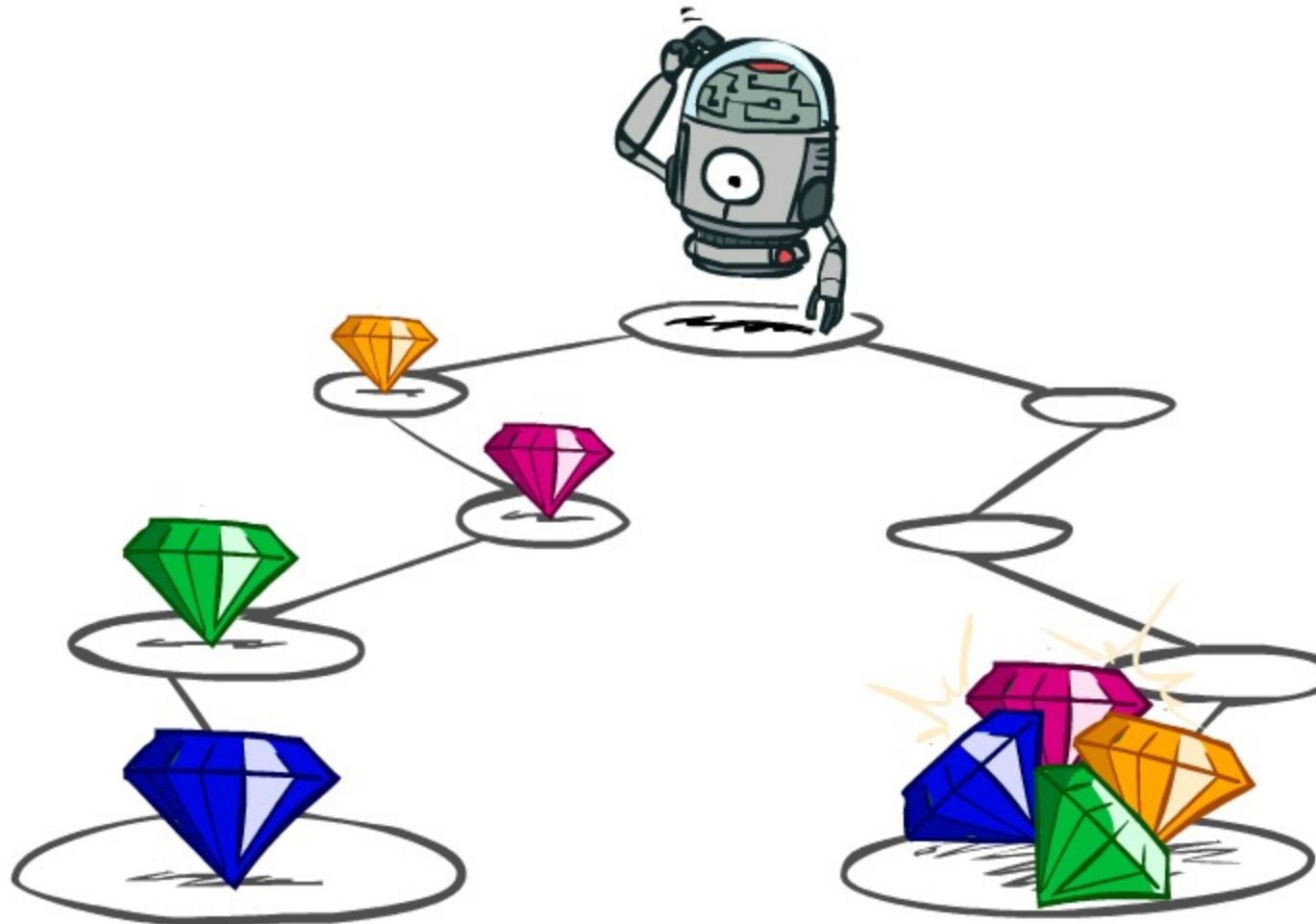
# MDP Search Trees

- Each MDP state projects an expectimax-like search tree



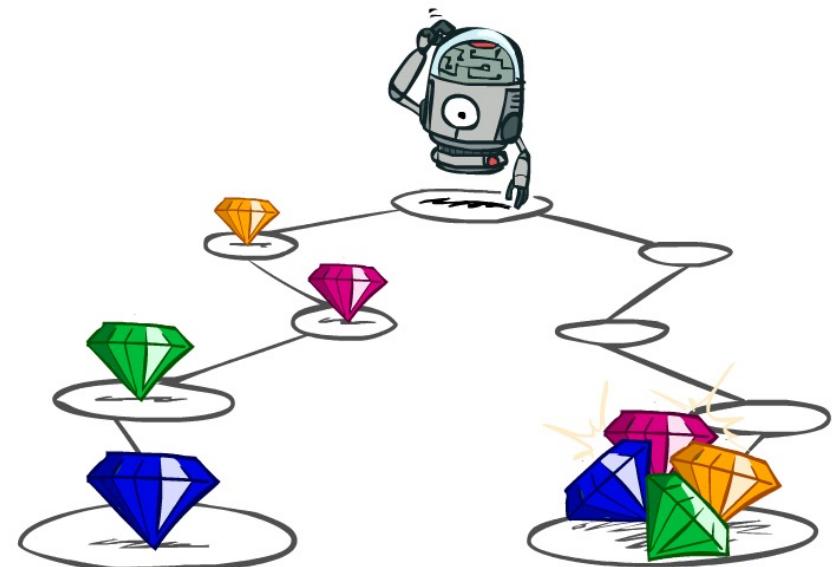
# Utilities of Sequences

---



# 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

# Visualizing Discounting

- How to discount?

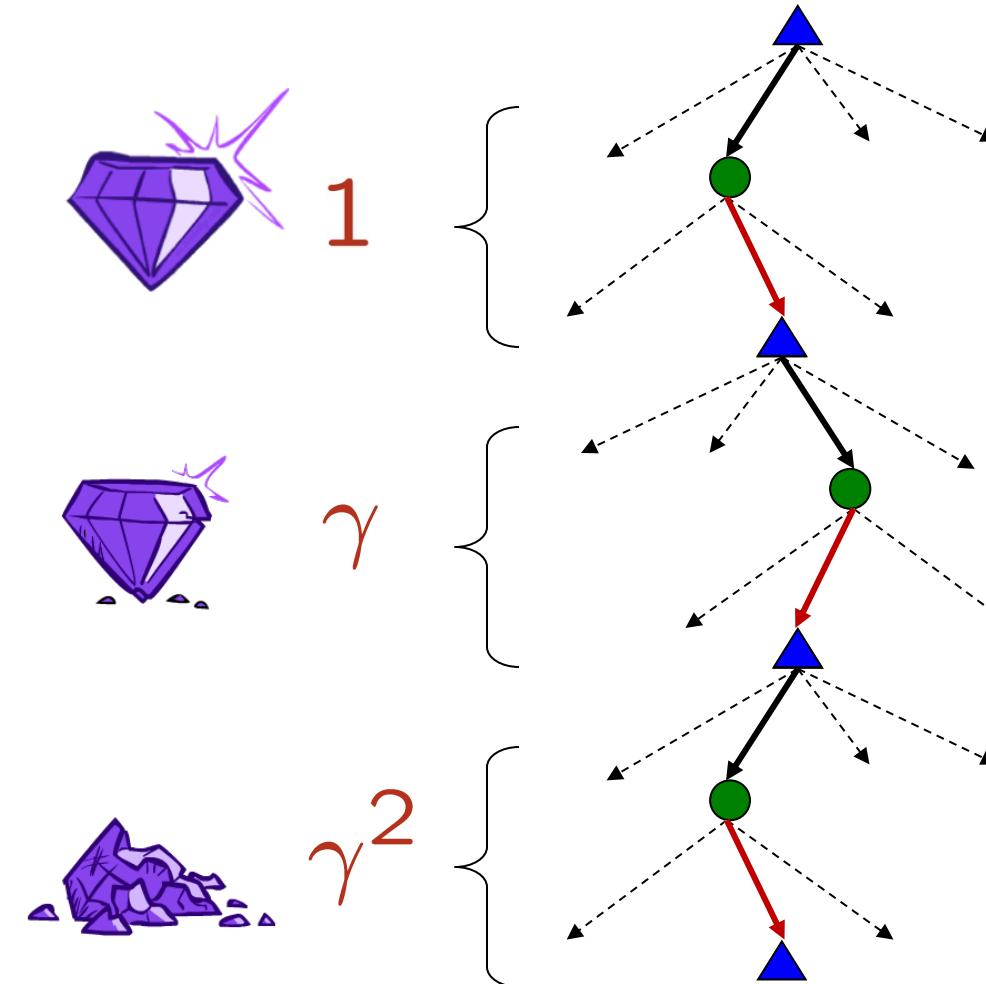
- Each time we descend a level, we multiply in the discount once

- Why discount?

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

- Example: discount of 0.5

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



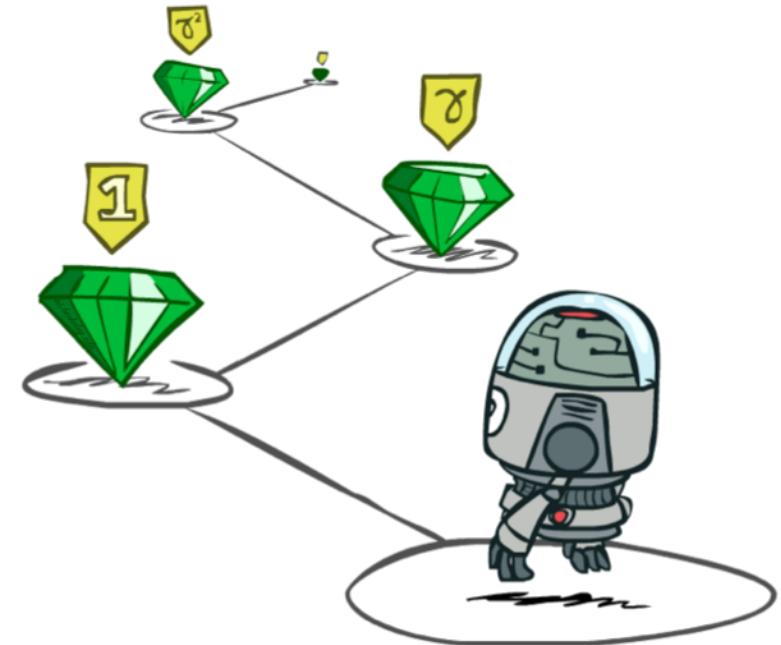
# Stationary Preferences\*

- Theorem: if we assume stationary preferences:

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

$\Updownarrow$

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



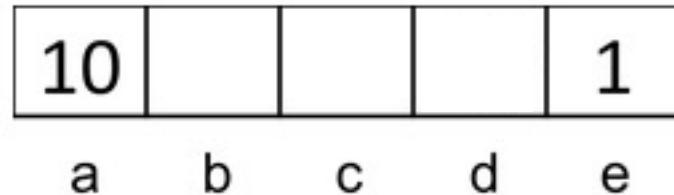
- Then: there are only two ways to define utilities

- Additive utility:  $U([r_0, r_1, r_2, \dots]) = r_0 + r_1 + r_2 + \dots$

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

# Quiz: Discounting

- Given:



- Actions:

- East
- West
- Exit (only available in exit states a, e)

- Transitions: deterministic

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

10				1
----	--	--	--	---

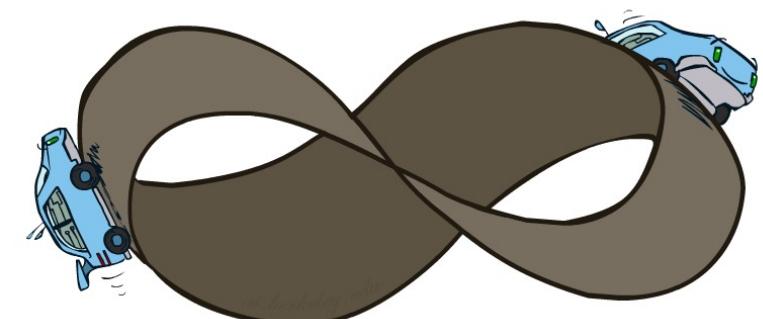
- Quiz 2: For  $\gamma = 0.1$ , what is the optimal policy?

10				1
----	--	--	--	---

- Quiz 3: For which  $\gamma$  are West and East equally good when in state d?

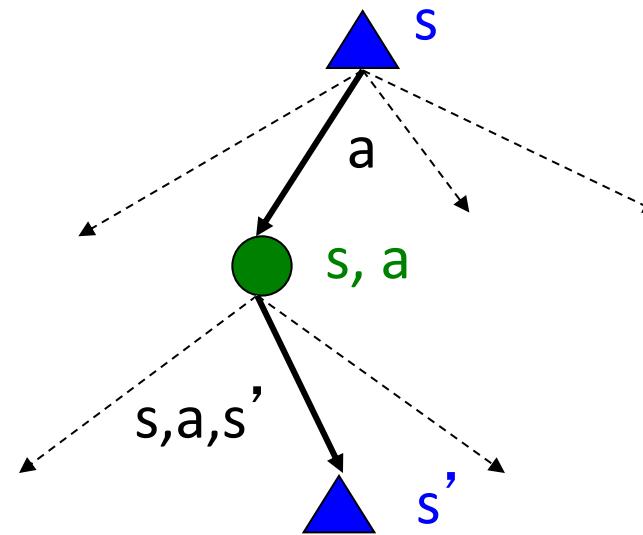
# Infinite Utilities?!

- 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)
  - Discounting: use  $0 < \gamma < 1$ 
$$U([r_0, \dots, r_\infty]) = \sum_{t=0}^{\infty} \gamma^t r_t \leq R_{\max}/(1 - \gamma)$$
    - Smaller  $\gamma$  means smaller “horizon” – shorter term focus
  - Absorbing state: guarantee that for every policy, a terminal state will eventually be reached (like “overheated” for racing)



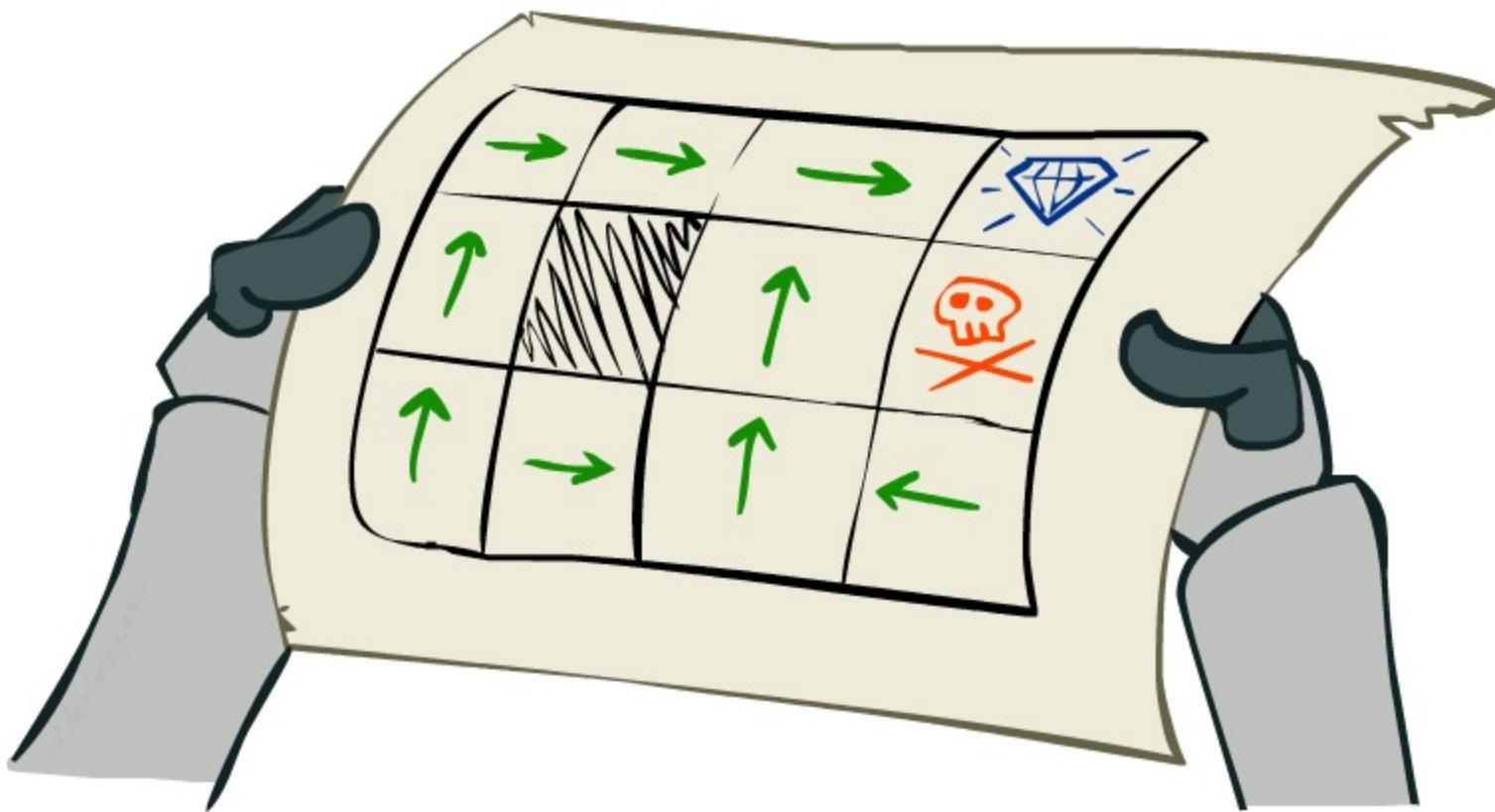
# Recap: Defining MDPs

- 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



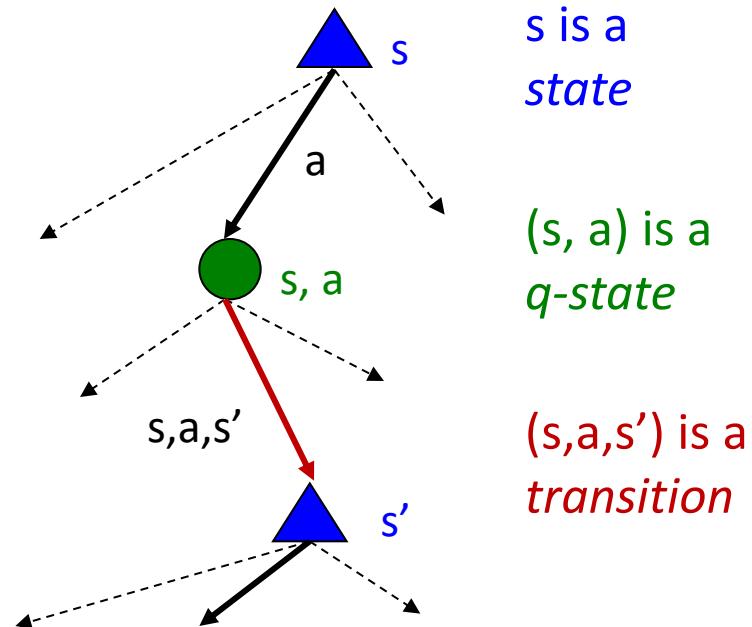
# Solving MDPs

---



# Optimal Quantities

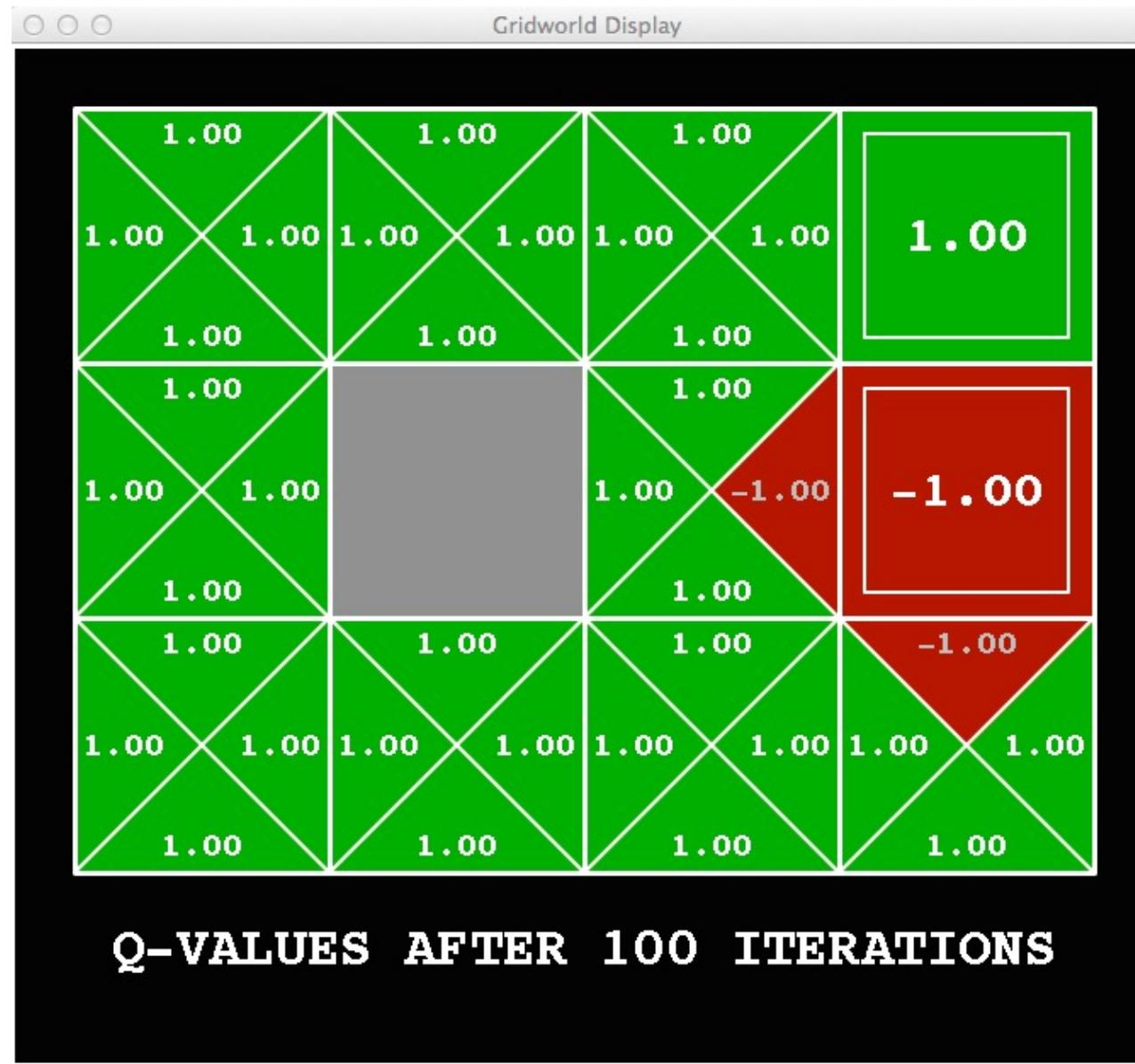
- The value (utility) of a state  $s$ :  
 $V^*(s)$  = expected utility starting in  $s$  and acting optimally
- The value (utility) of a q-state  $(s,a)$ :  
 $Q^*(s,a)$  = expected utility starting out having taken action  $a$  from state  $s$  and (thereafter) acting optimally
- The optimal policy:  
 $\pi^*(s)$  = optimal action from state  $s$



# Snapshot of Demo – Gridworld V Values



# Snapshot of Demo – Gridworld Q Values



# Snapshot of Demo – Gridworld V Values



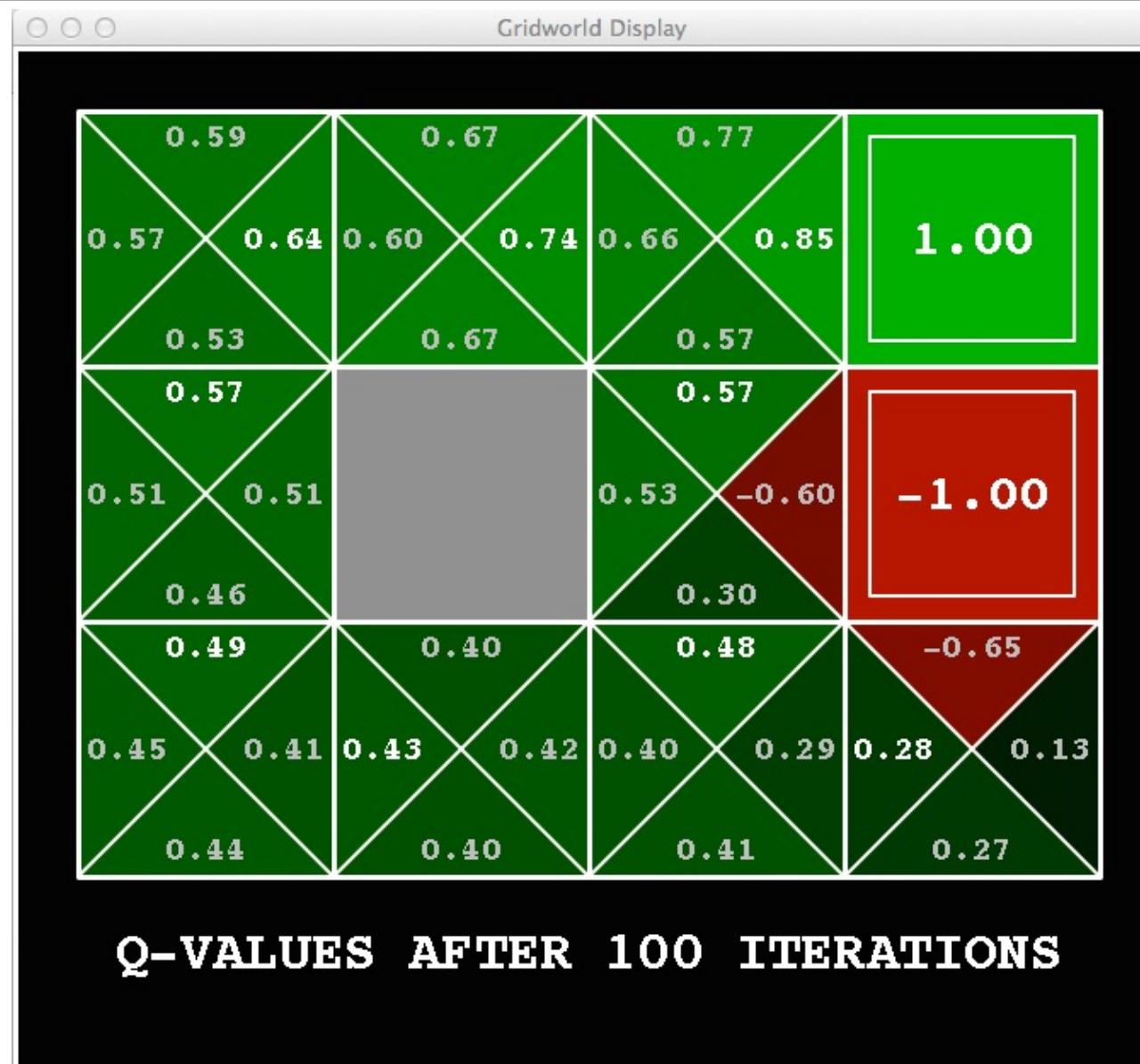
# Snapshot of Demo – Gridworld Q Values



# Snapshot of Demo – Gridworld V Values



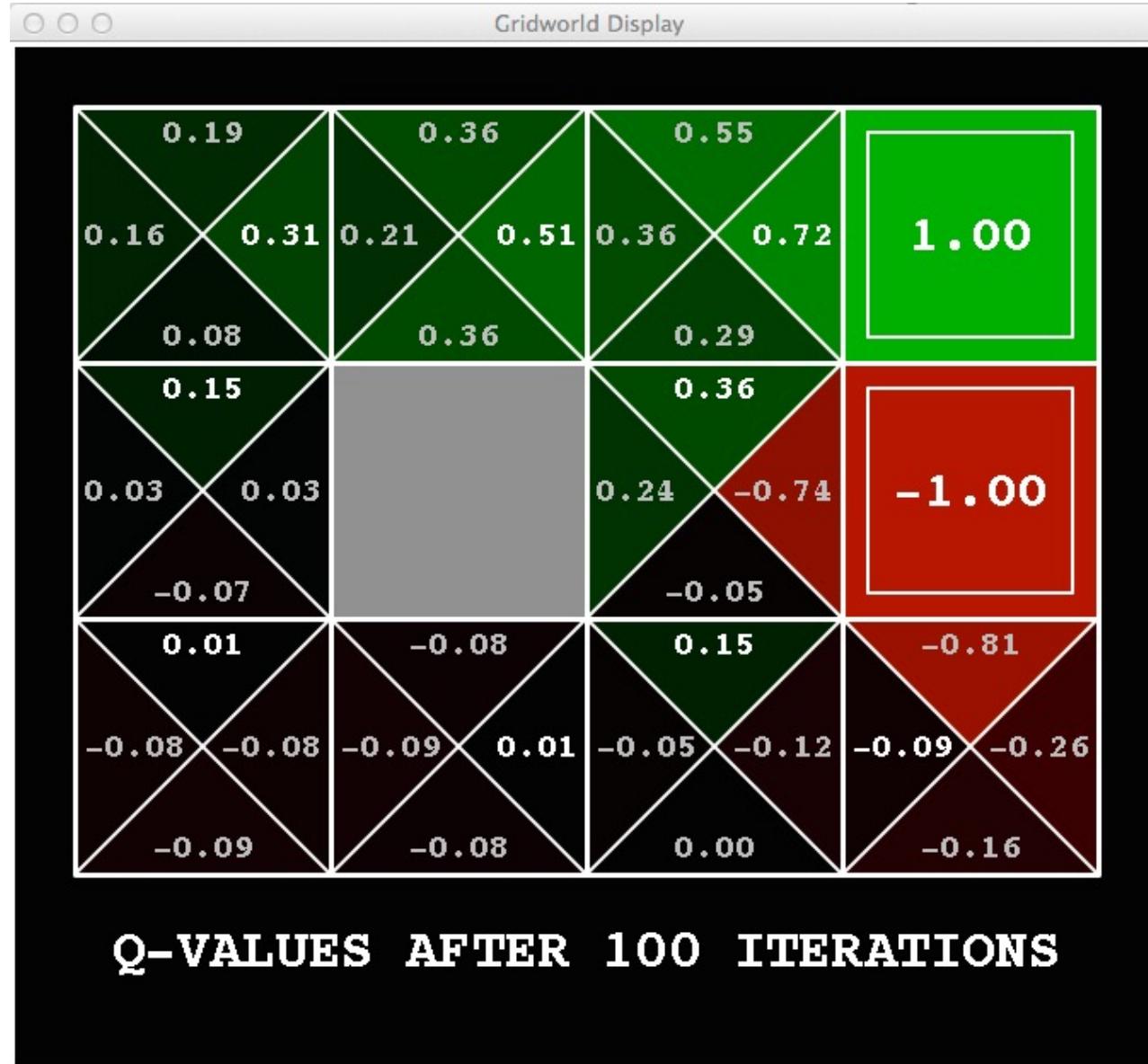
# Snapshot of Demo – Gridworld Q Values



# Snapshot of Demo – Gridworld V Values



# Snapshot of Demo – Gridworld Q Values



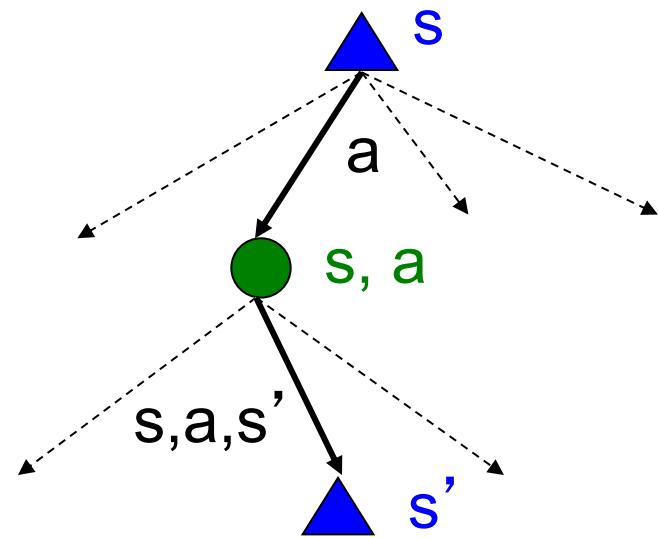
# Values of States

- 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!
- Recursive definition of value:

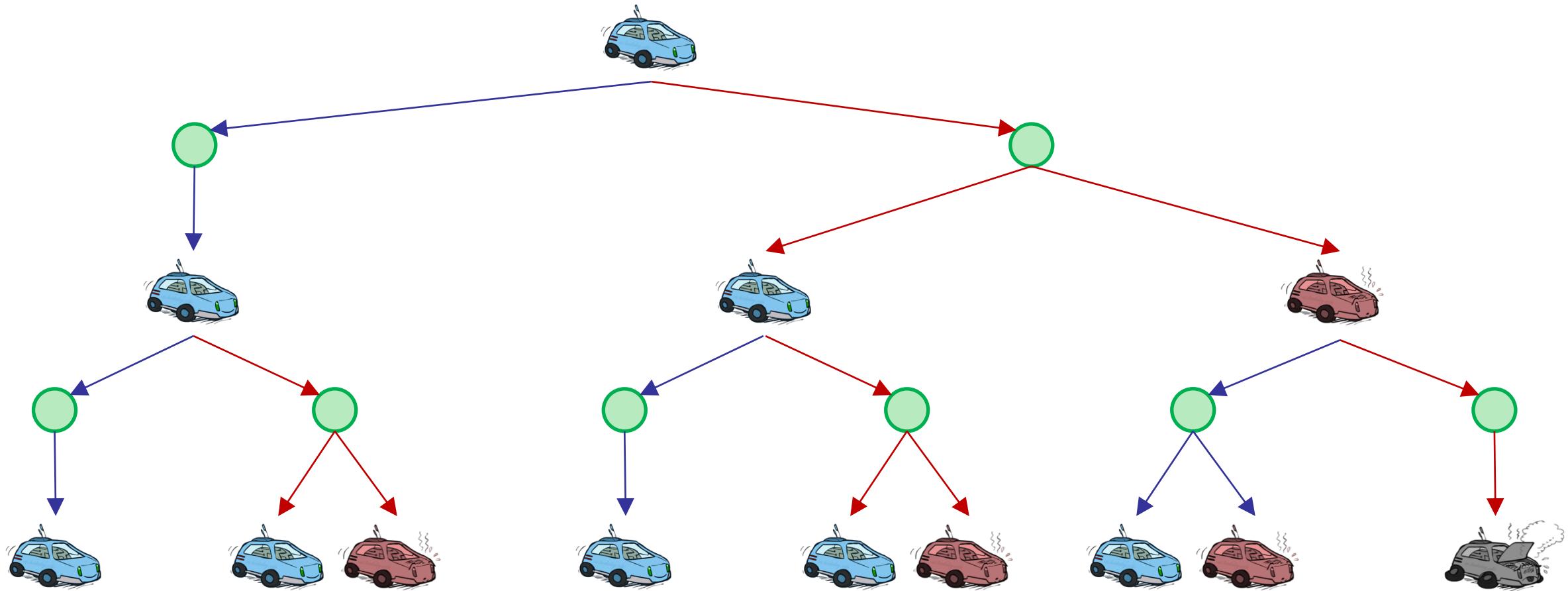
$$V^*(s) = \max_a Q^*(s, a)$$

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

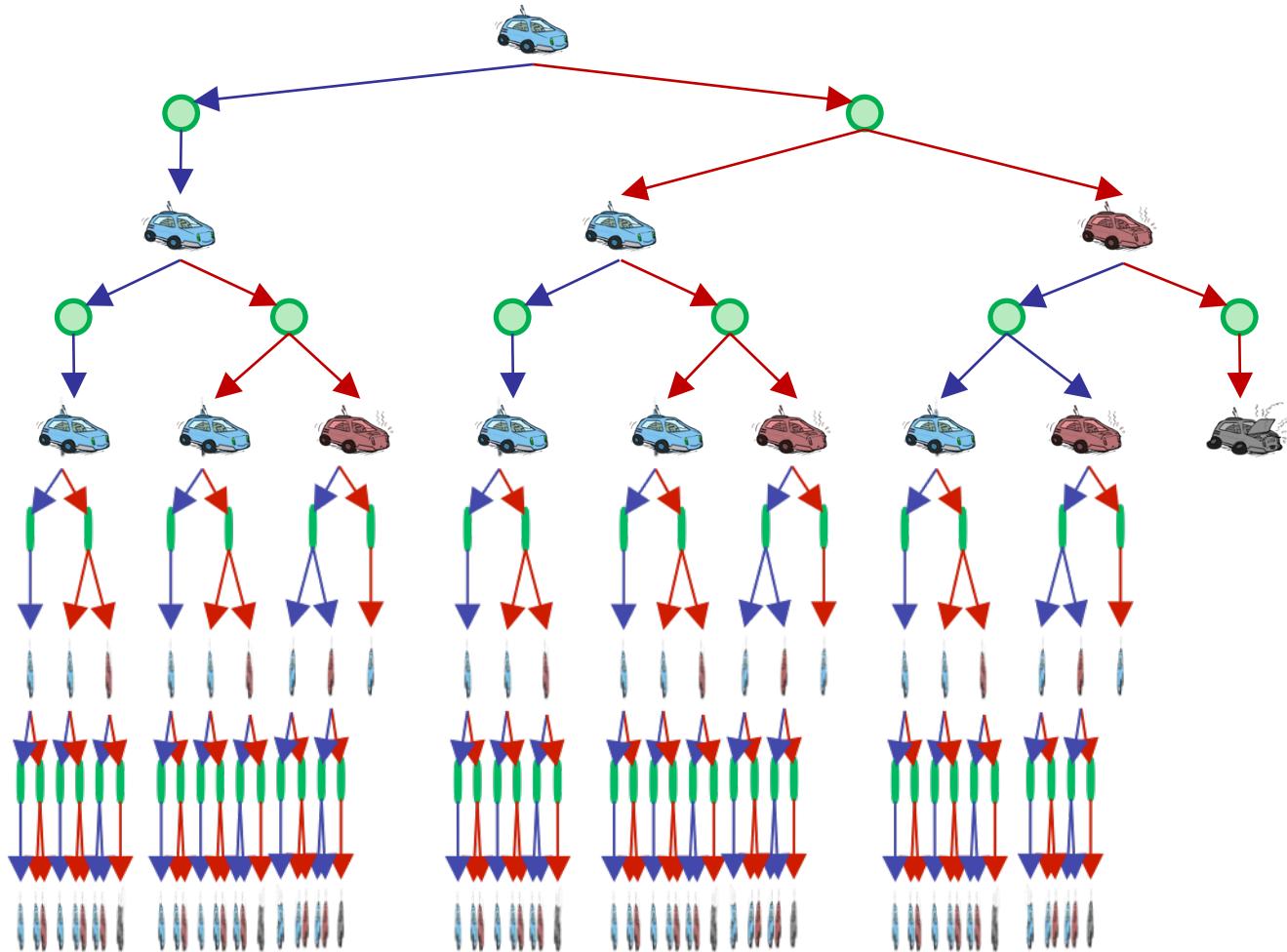
$$V^*(s) = \max_a \sum_{s'} T(s, a, s') [R(s, a, s') + \gamma V^*(s')]$$



# Racing Search Tree

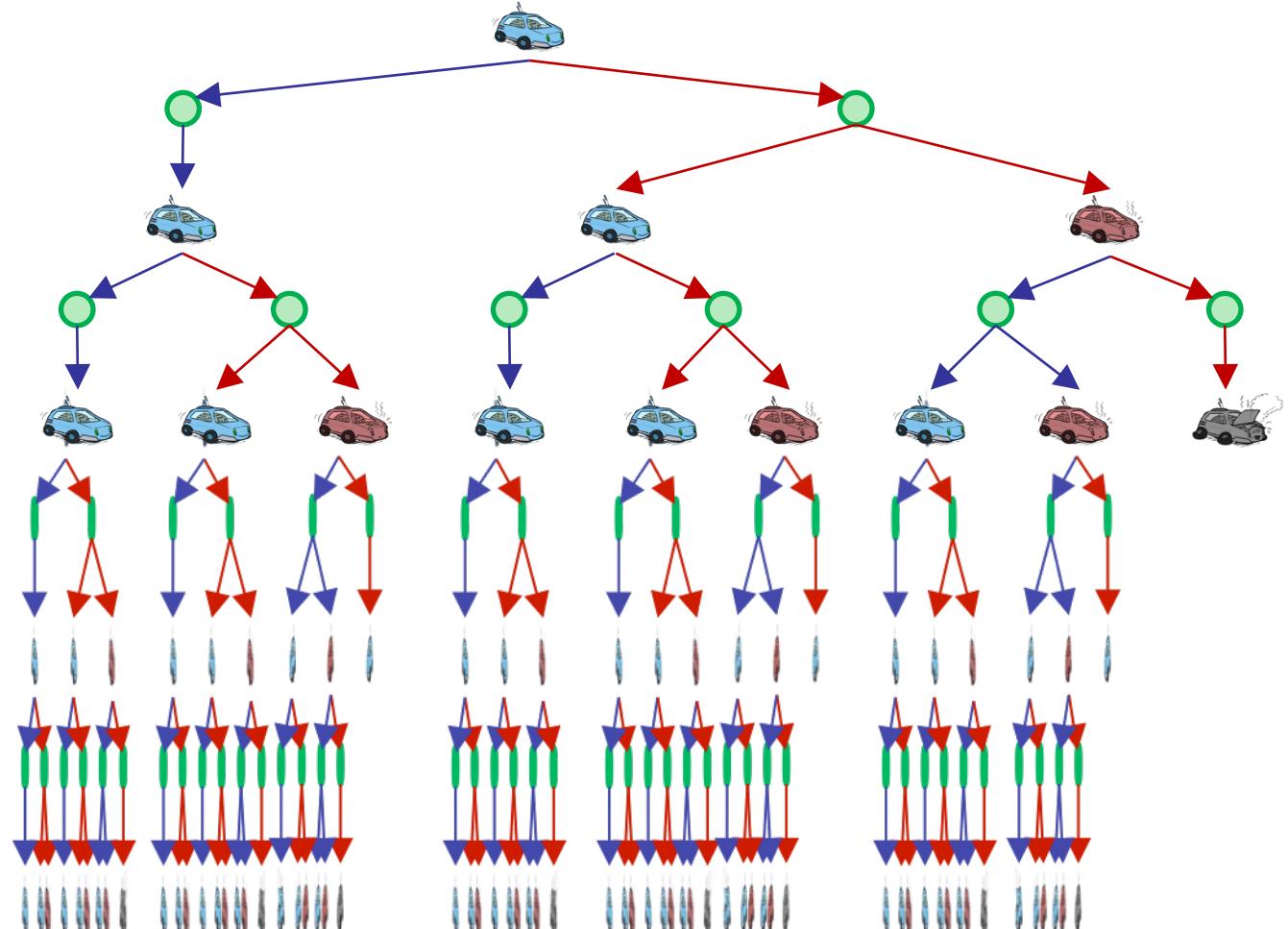


# Racing Search Tree

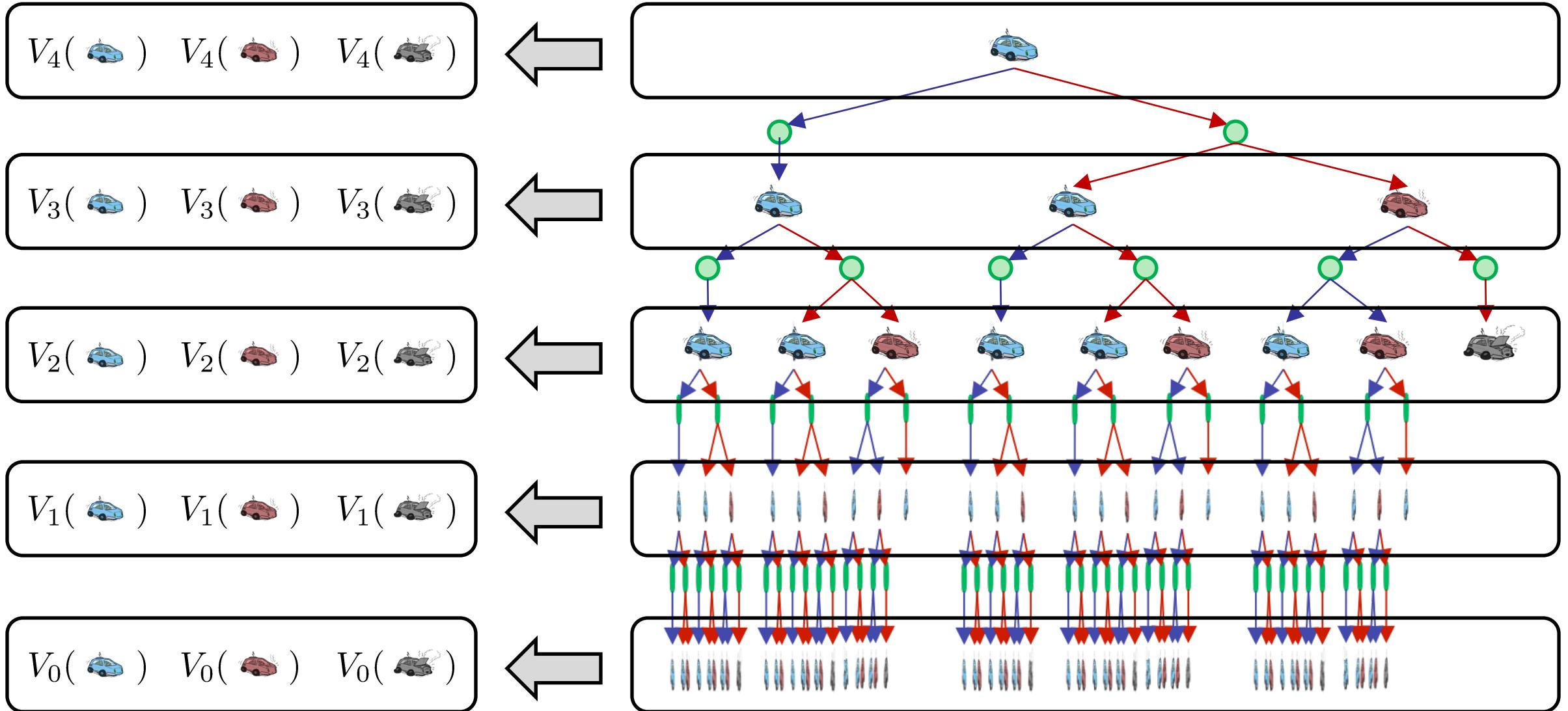


# Racing Search Tree

- Problem: States are repeated
  - Idea: Only compute needed quantities once
- Problem: Tree goes on forever
  - Idea: Do a depth-limited computation, but with increasing depths until change is small
  - Note: deep parts of the tree eventually don't matter if  $\gamma < 1$

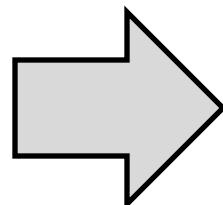
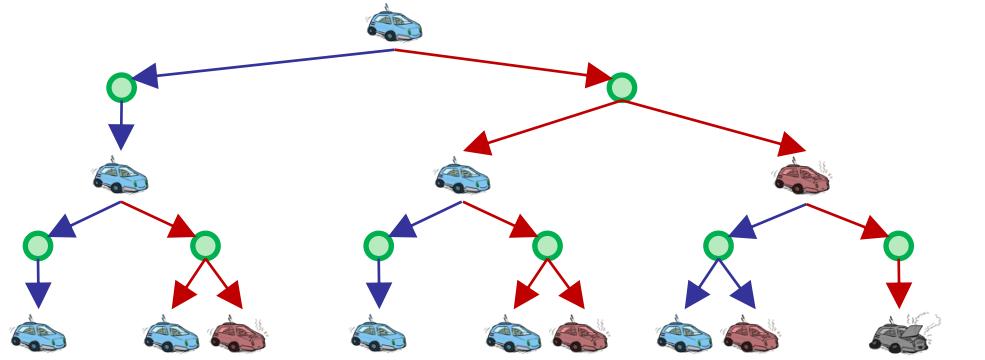
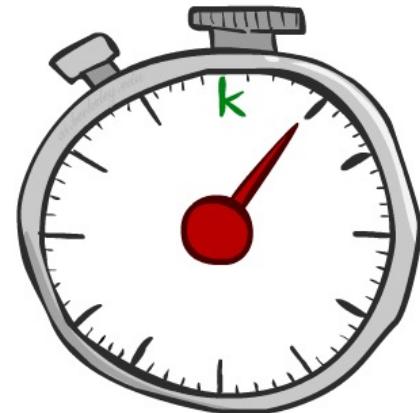


# Computing Time-Limited Values



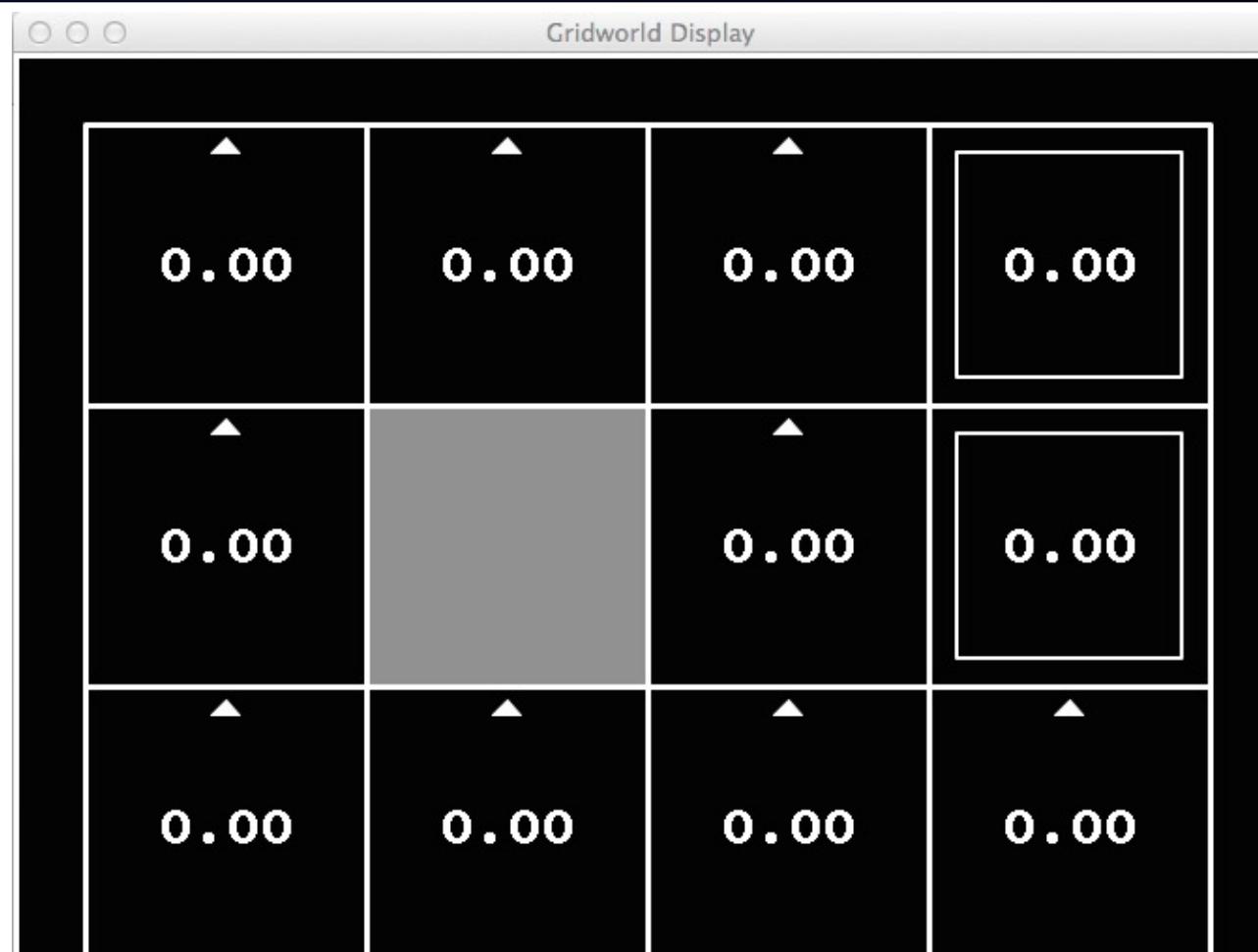
# Time-Limited Values

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



A large, solid blue triangle centered on the page. It has a thin black outline and is oriented vertically, pointing upwards. The interior of the triangle is filled with a medium shade of blue.

**k=0**



**VALUES AFTER 0 ITERATIONS**

Noise = 0.2  
Discount = 0.9  
Living reward = 0

$k=1$



$k=2$



**k=3**



# k=4



Noise = 0.2  
Discount = 0.9  
Living reward = 0

# k=5



# k=6



k=7



**k=8**



k=9



VALUES AFTER 9 ITERATIONS

Noise = 0.2  
Discount = 0.9  
Living reward = 0

# k=10



Noise = 0.2

Discount = 0.9

Living reward = 0

**k=11**



**k=12**



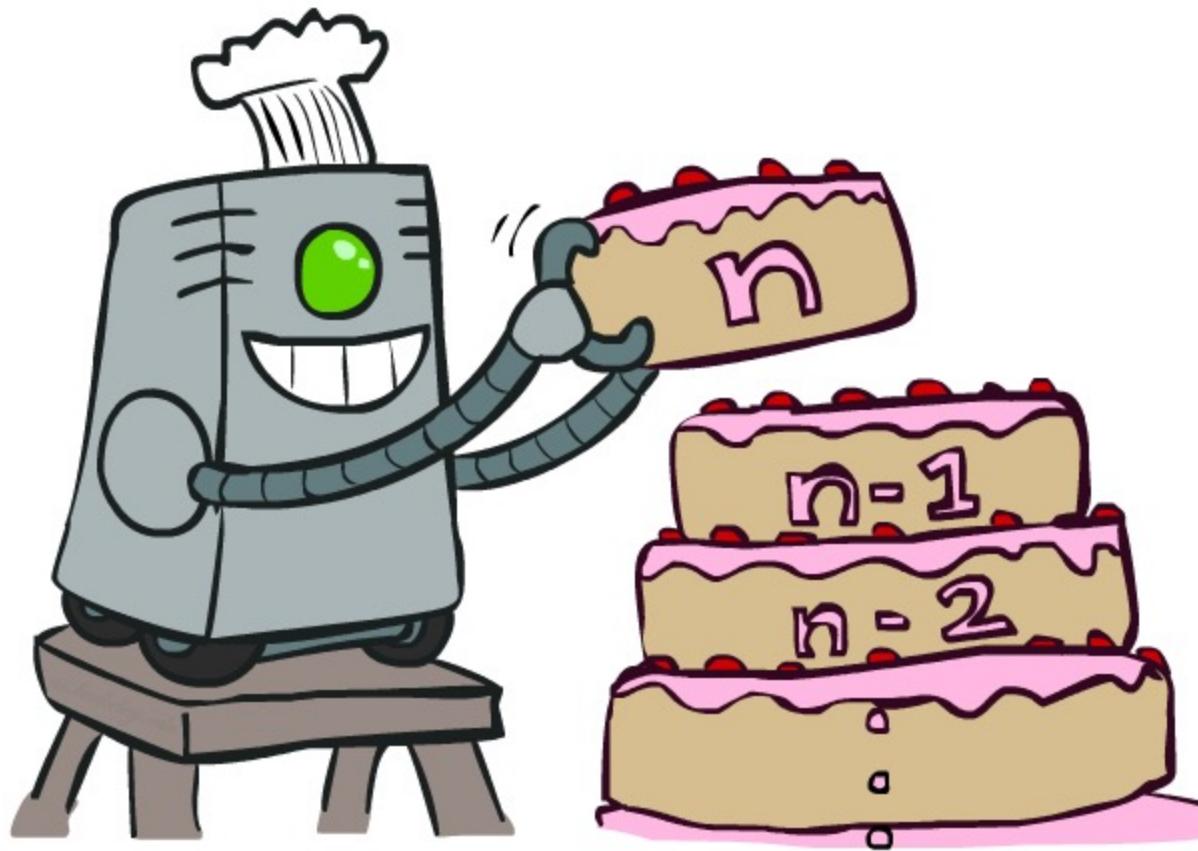
# k=100



Noise = 0.2  
Discount = 0.9  
Living reward = 0

# Value Iteration

---

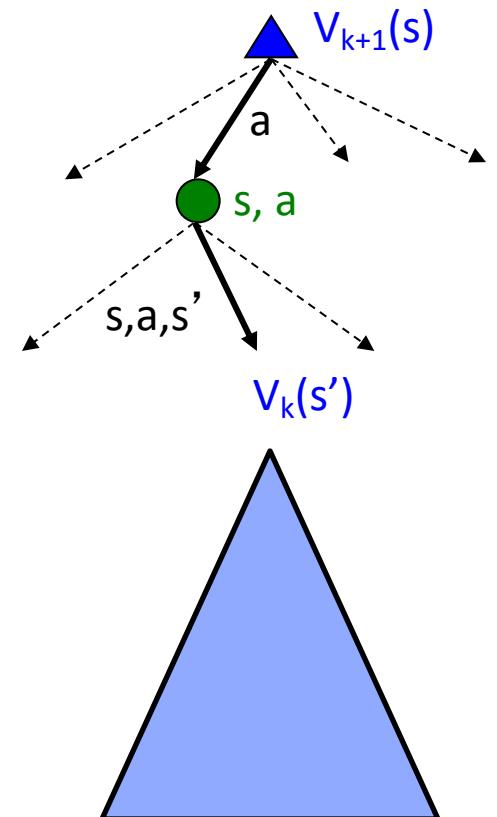


# Value Iteration

- 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:

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

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



# Example: Value Iteration

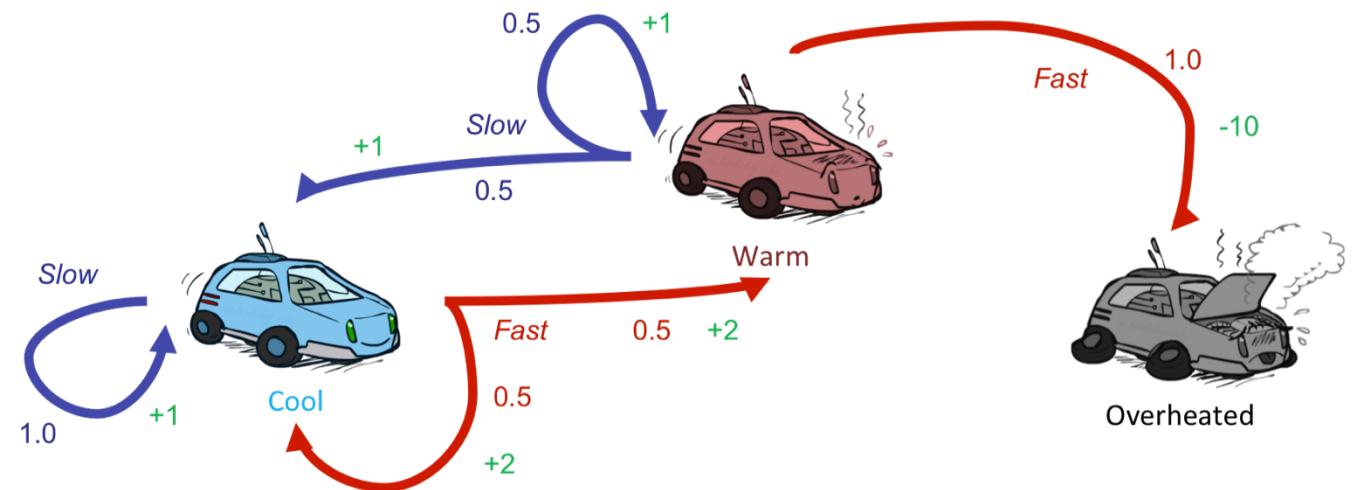
s	a	s'	T(s,a,s')	R(s,a,s')
	Slow		1.0	+1
	Fast		0.5	+2
	Fast		0.5	+2
	Slow		0.5	+1
	Slow		0.5	+1
	Fast		1.0	-10
	(end)		1.0	0

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

Assume no discount!

# Example: Value Iteration

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

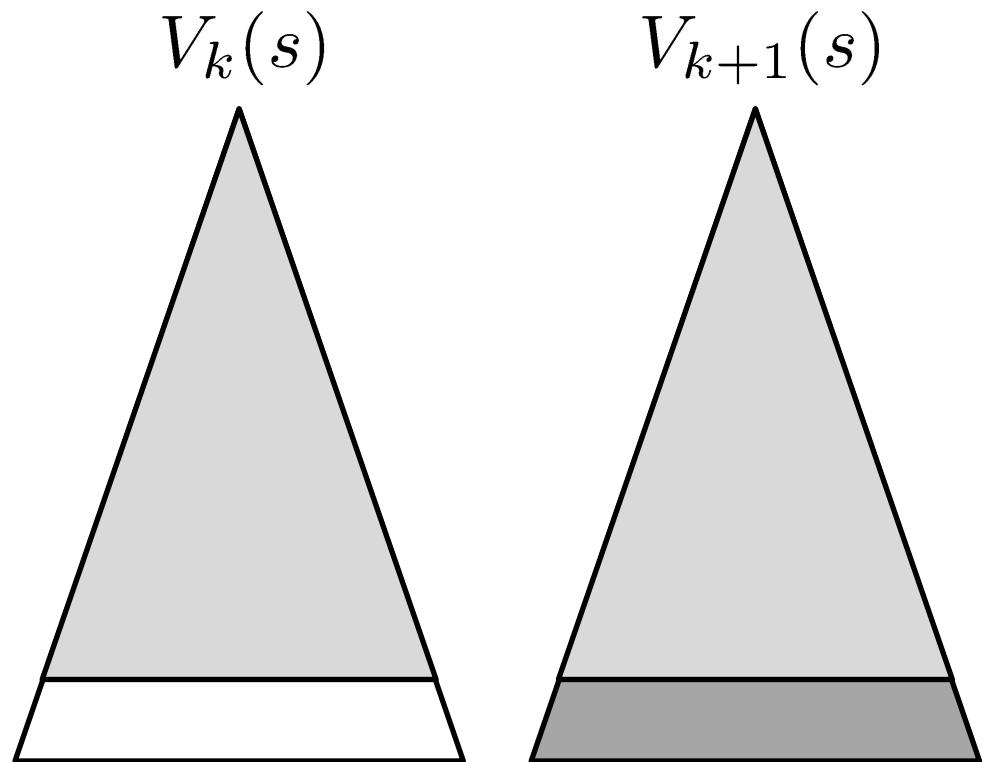


Assume no discount!

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

# 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  $R_{\text{MAX}}$
  - It is at worst  $R_{\text{MIN}}$
  - But everything is discounted by  $\gamma^k$  that far out
  - So  $V_k$  and  $V_{k+1}$  are at most  $\gamma^k \max|R|$  different
  - So as  $k$  increases, the values converge



# Next Time: Policy-Based Methods

---