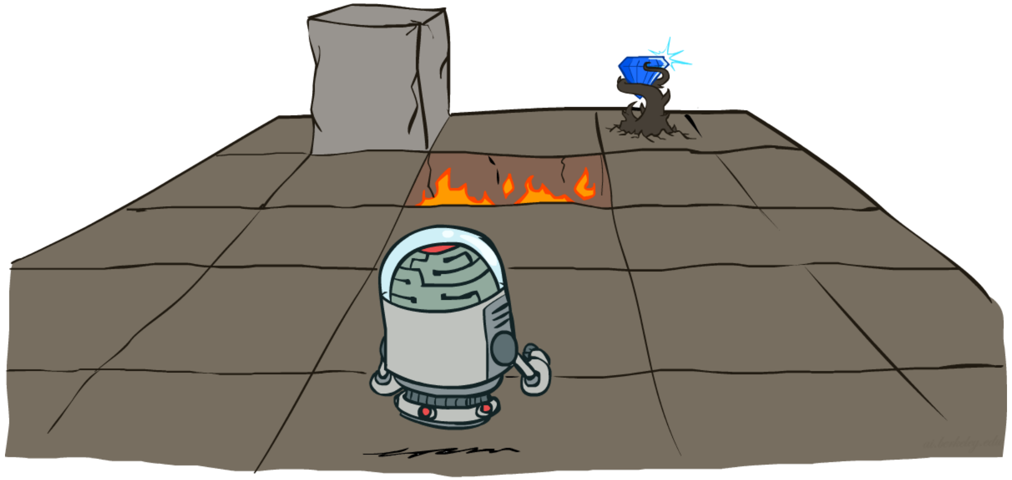
Artificial Intelligence - INFOF311

Markov decision processes, part II

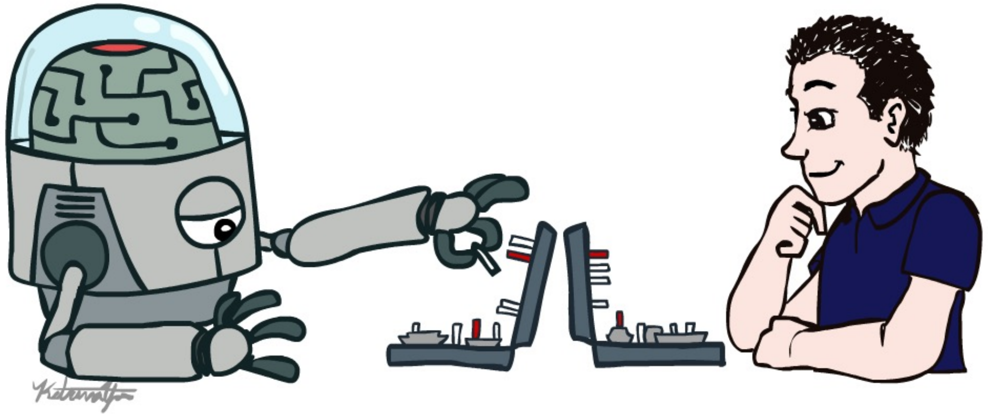


Instructor : Tom Lenaerts

Acknowledgement

We thank Stuart Russell for his generosity in allowing us to use the slide set of the UC Berkeley Course CS188, Introduction to Artificial Intelligence. These slides were created by Dan Klein, Pieter Abbeel and Anca Dragan for CS188 Intro to AI at UC Berkeley. All CS188 materials are available at http://ai.berkeley.edu.

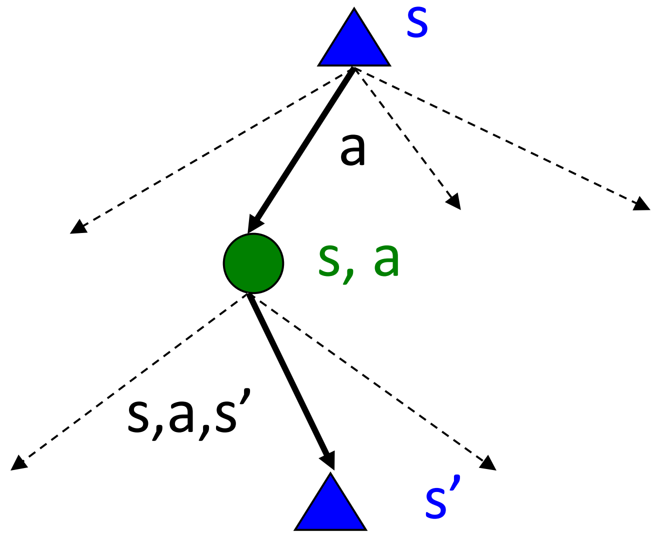
Center for
Human-Compatible
Artificial
Intelligence



The slides for INFOF311 are slightly modified versions of the slides of the spring and summer CS188 sessions in 2021 and 2022

Recap: Defining MDPs

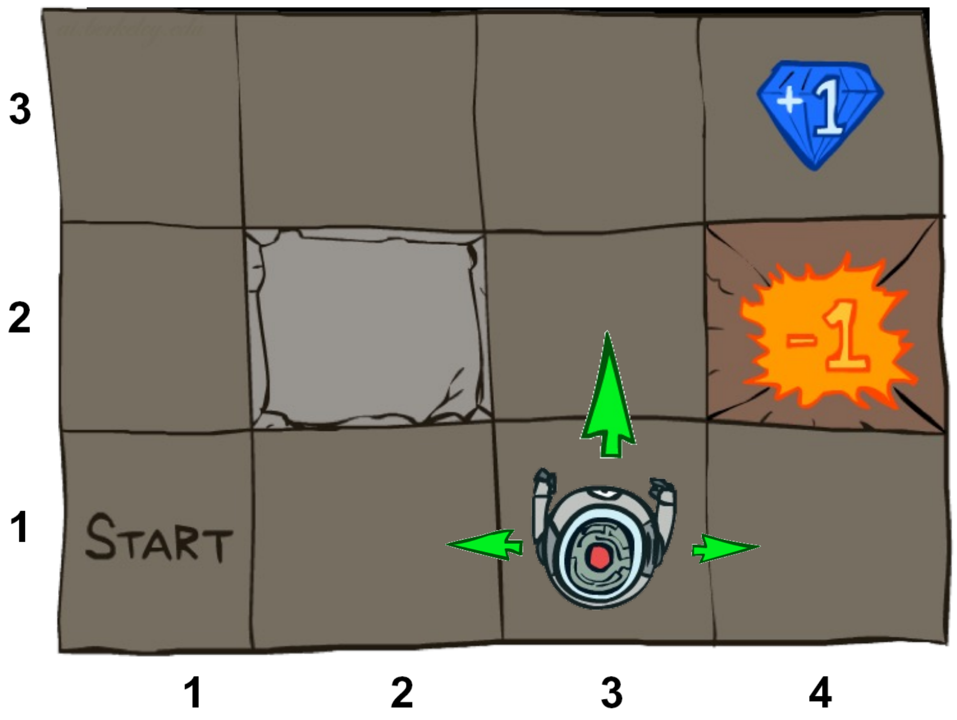
- Markov decision processes:
  - Set of states S
  - Start state s_0
  - Set of actions A
  - Transitions P(s'|s,a) (or T(s,a,s'))
  - Rewards R(s,a,s') (and discount γ)

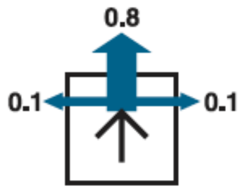


- MDP quantities so far:
  - Policy = Choice of action for each state
  - Utility = sum of (discounted) rewards
  - Values = expected future utility for each state (max node)
  - Q-values = expected future utility from a q-state (chance node)

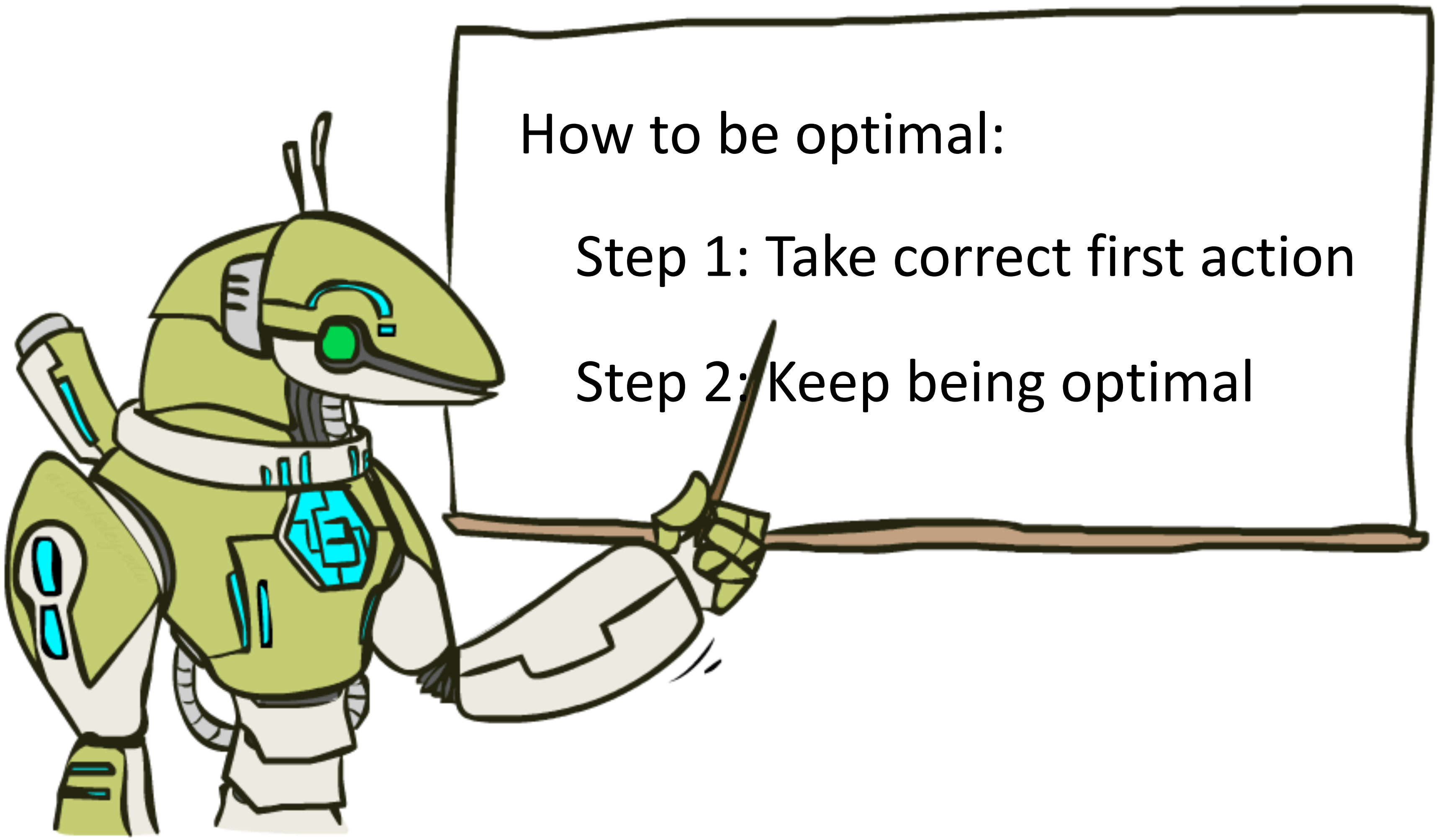
Example: Grid World

- 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 r each step (can be negative)
  - Big rewards come at the end (good or bad)
- Goal: maximize sum of rewards





The Bellman Equations



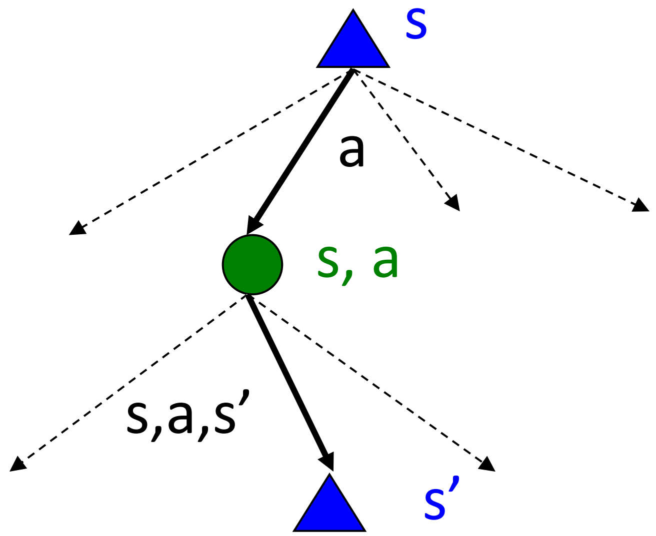
The Bellman Equations

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

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

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

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

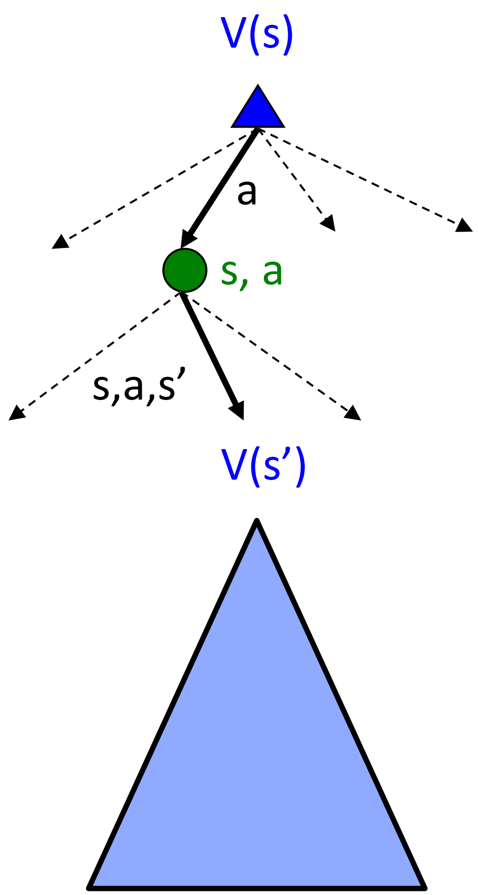


- These are the Bellman equations, and they characterize optimal values in a way we'll use over and over

Value Iteration

- Bellman equations characterize the optimal values:

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



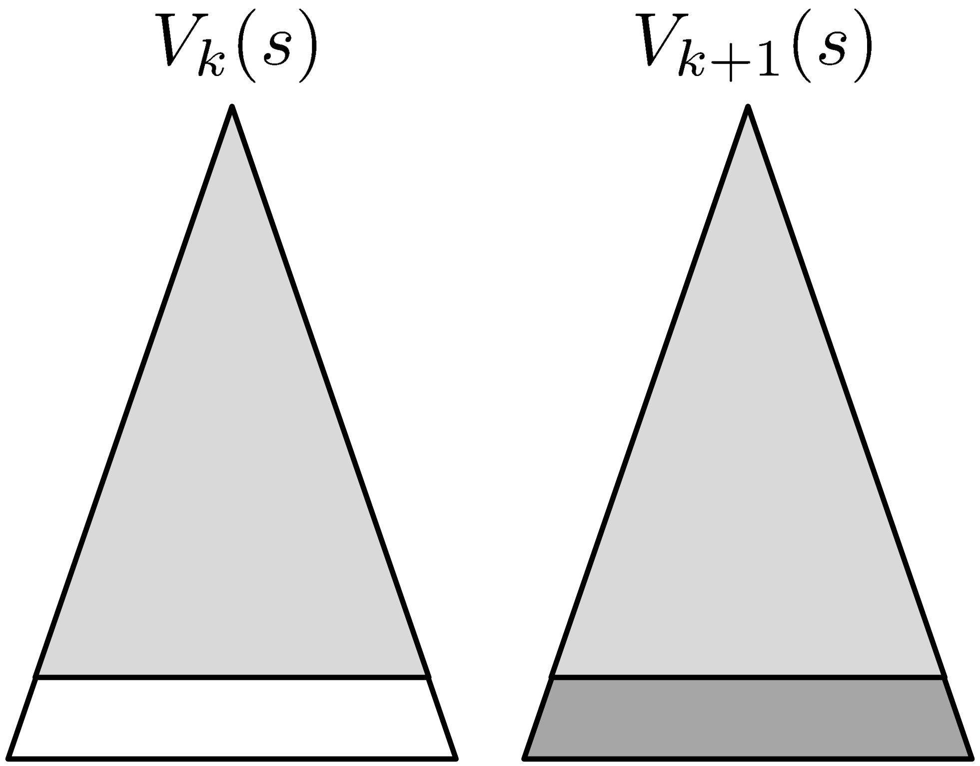
- Value iteration computes them:

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

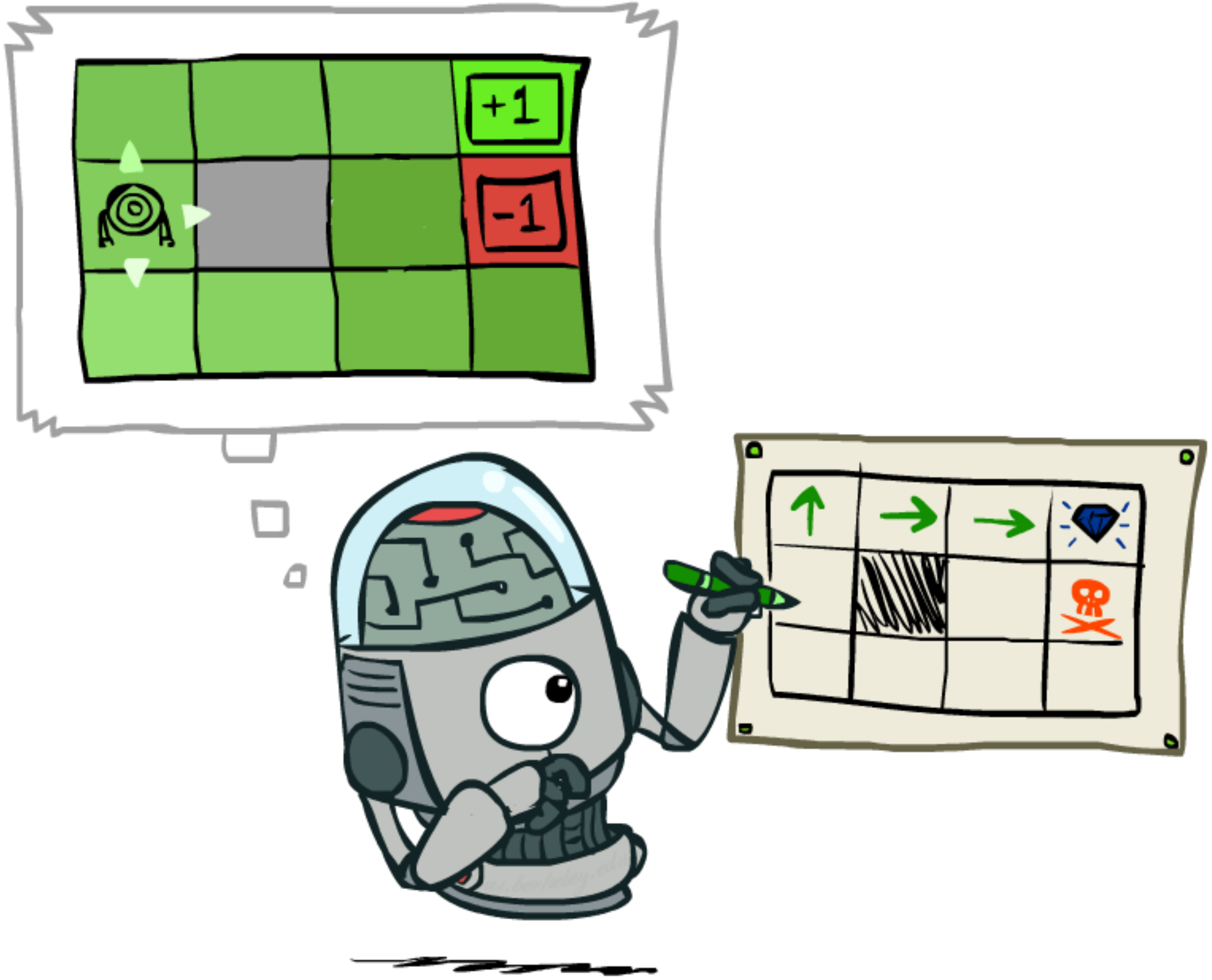
- Value iteration is just a fixed point solution method
  - … though the V_k vectors are also interpretable as time-limited values

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 $\gamma^k \max|R|$ different
  - So as k increases, the values converge



Policy Extraction

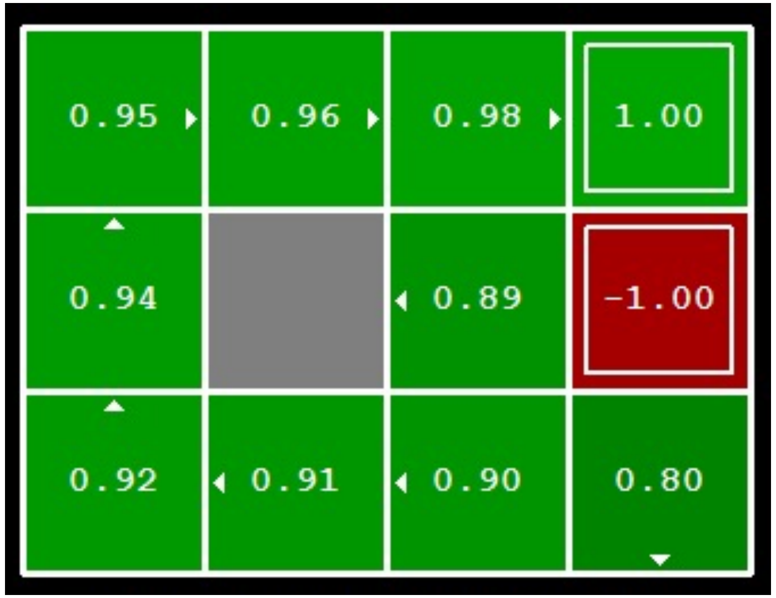


Computing Actions from Values

- 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)

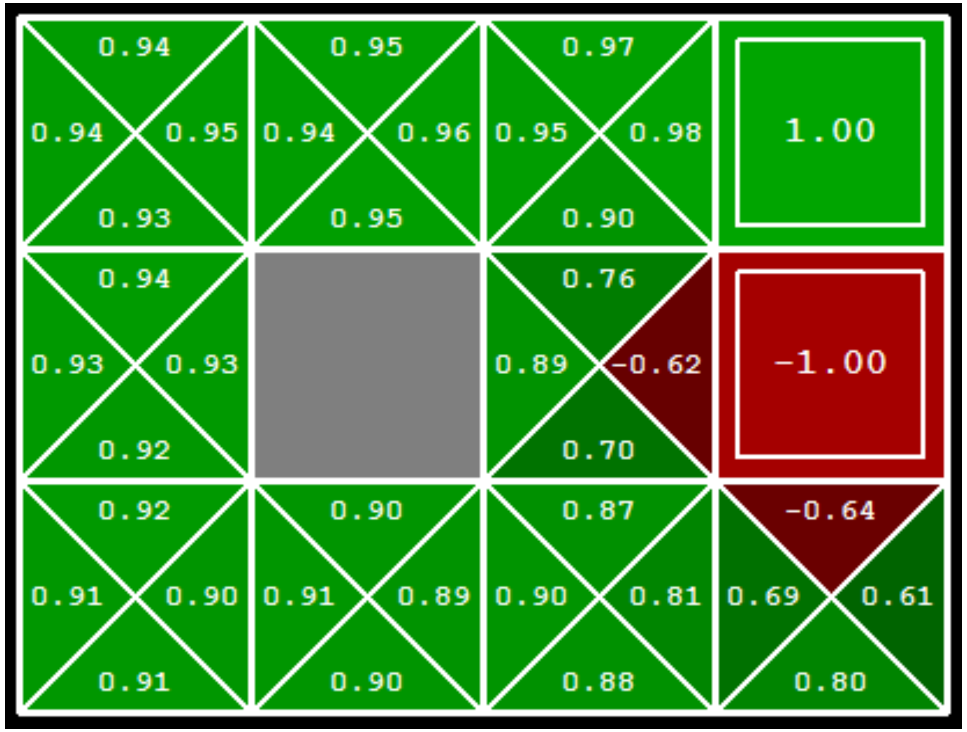


$$\pi^*(s) = \arg\max_a \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V^*(s')]$$

- 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:

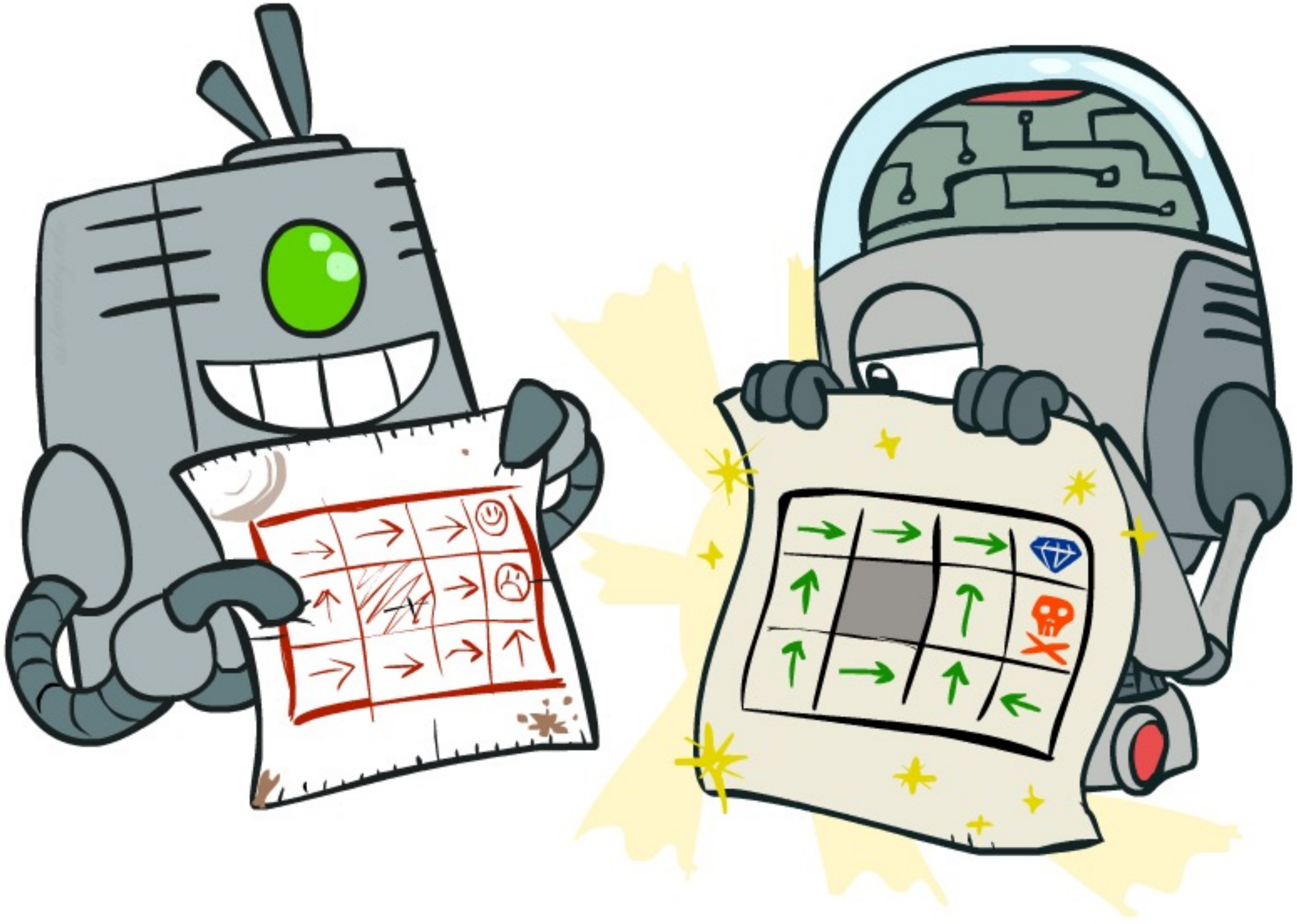


- How should we act?
  - Completely trivial to decide!

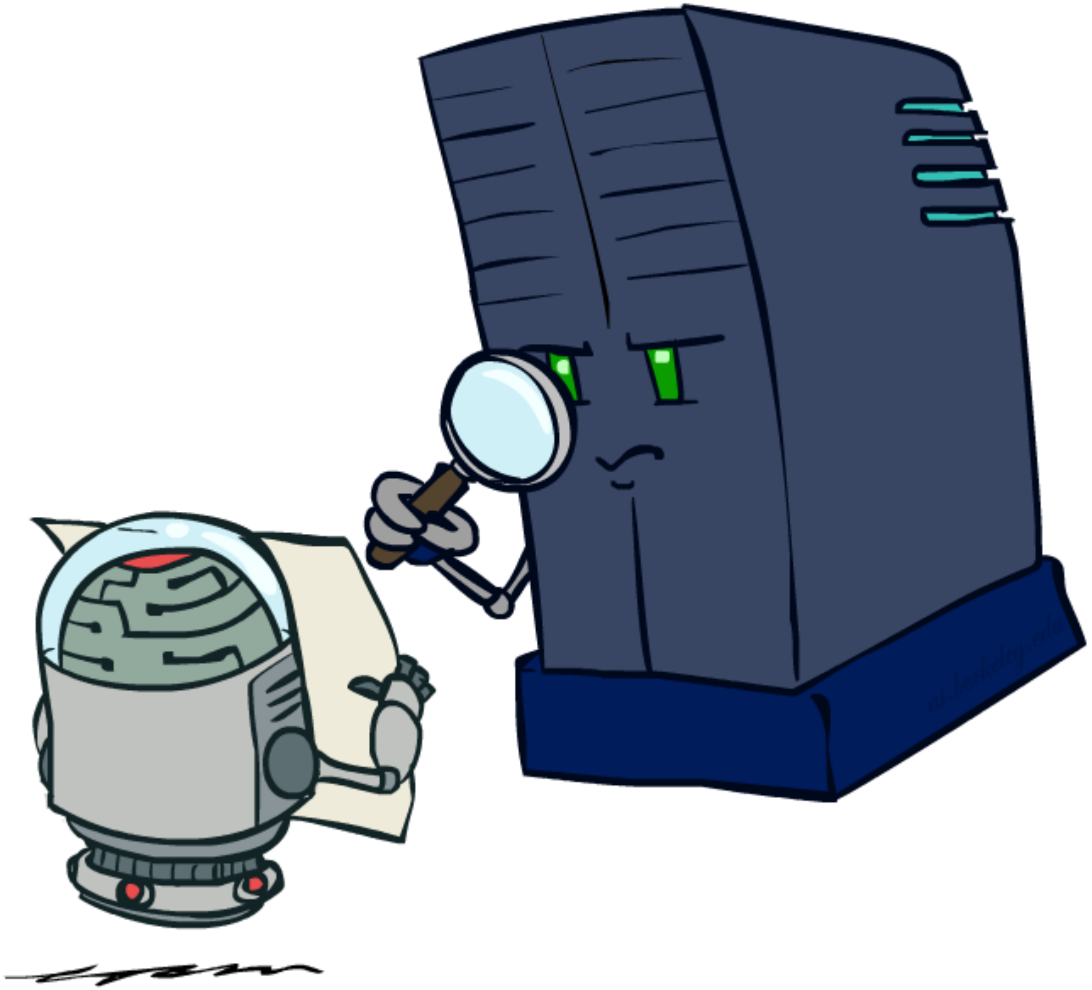
$$\pi^*(s) = \arg\max_a Q^*(s,a)$$

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

Policy Methods

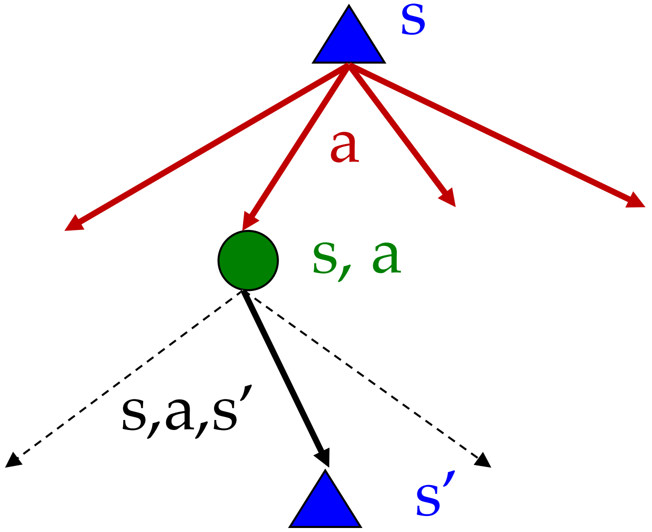


Policy Evaluation

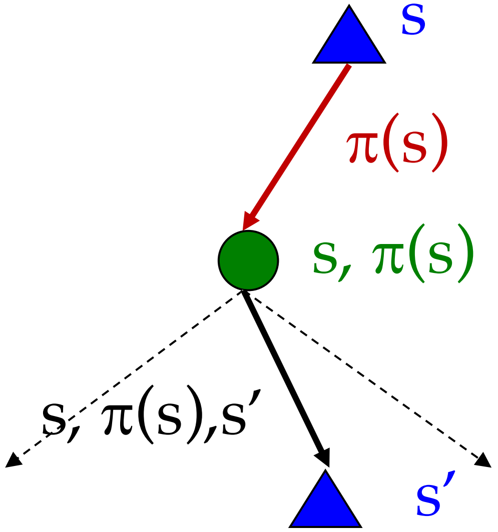


Fixed Policies

Do the optimal action



Do what π says to do



- Expectimax trees max over all actions to compute the optimal values
- If we fixed some policy $\pi(s)$, 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

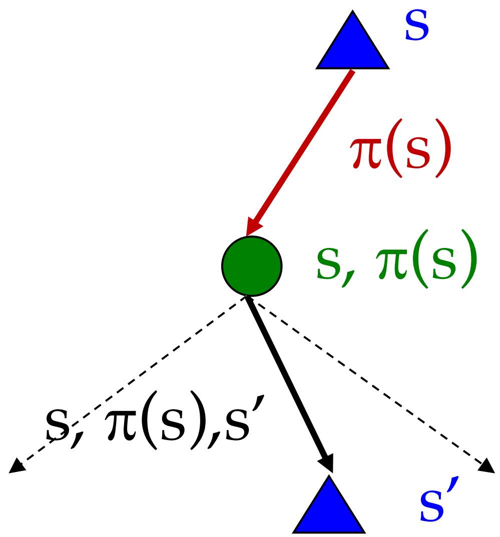
- Another basic operation: compute the utility of a state s under a fixed (generally non-optimal) policy

- 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

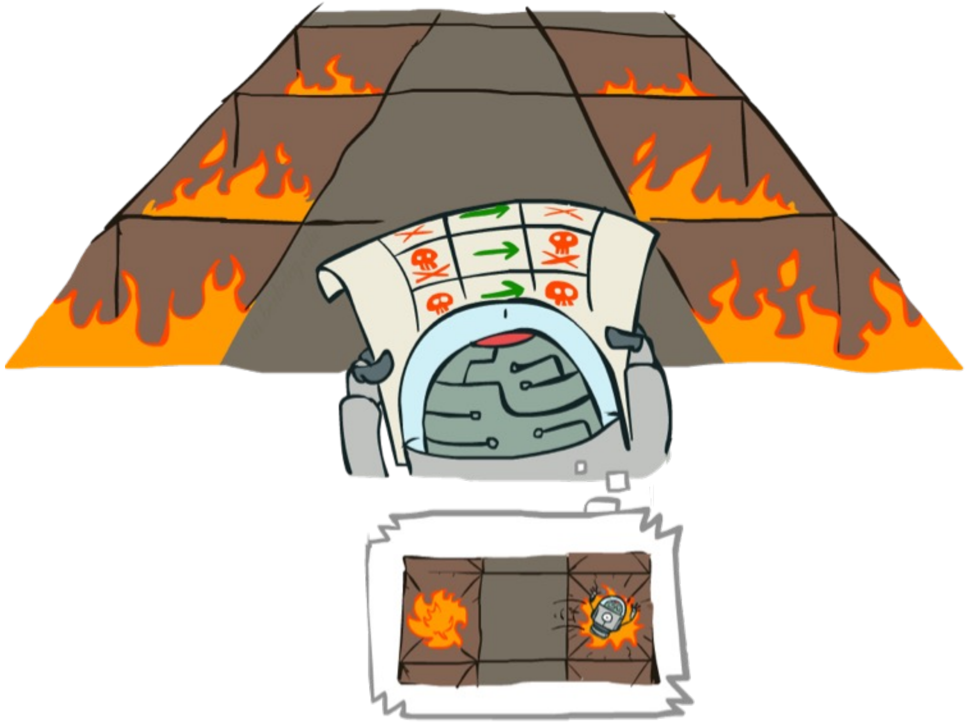
- Recursive relation (one-step look-ahead / Bellman equation):

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

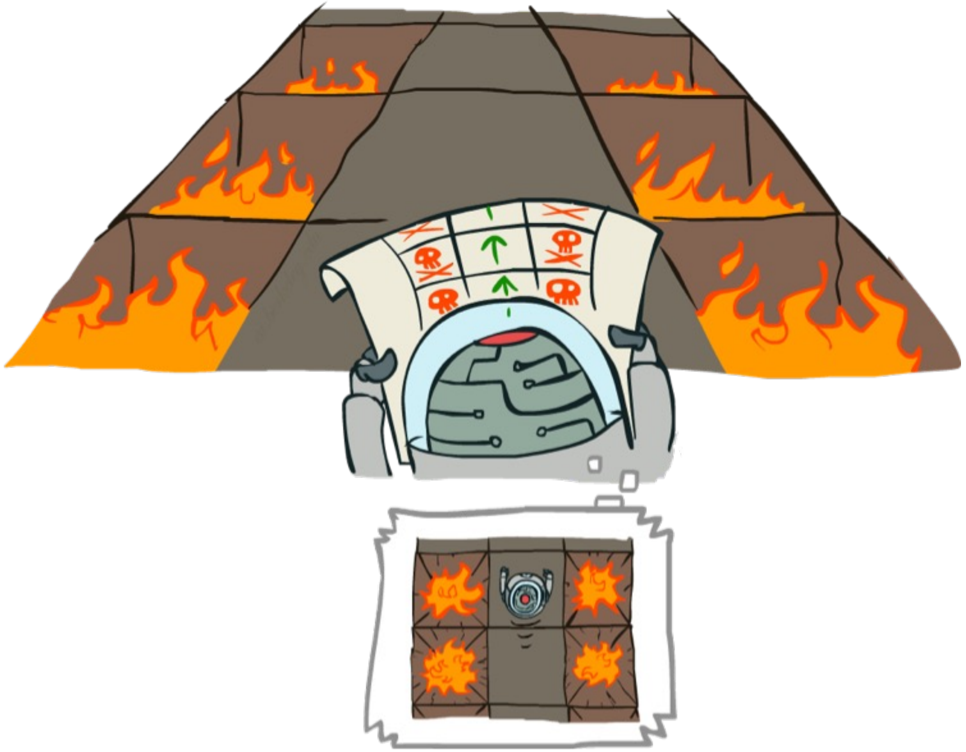


Example: Policy Evaluation

Always Go Right



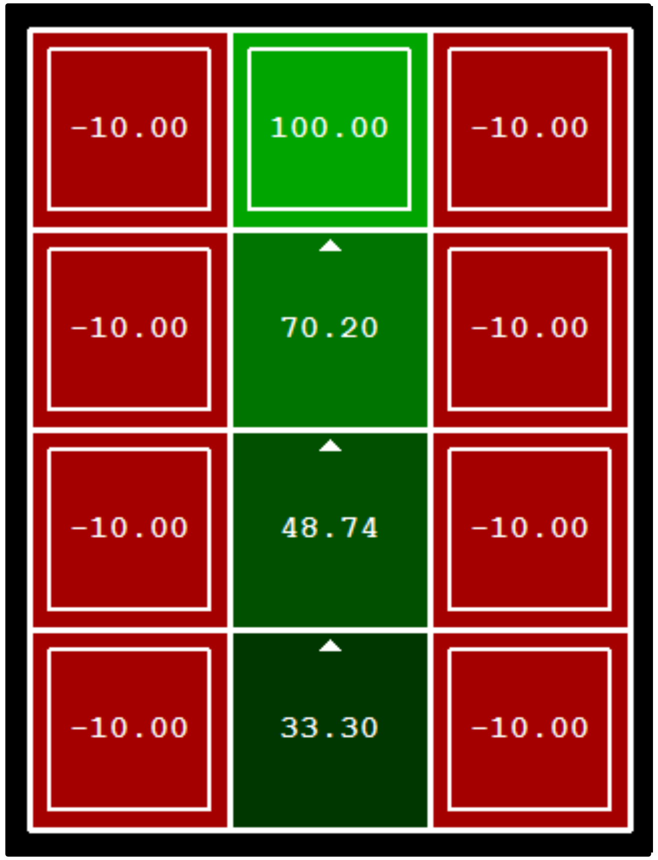
Always Go Forward



Example: Policy Evaluation

Always Go Right

Always Go Forward



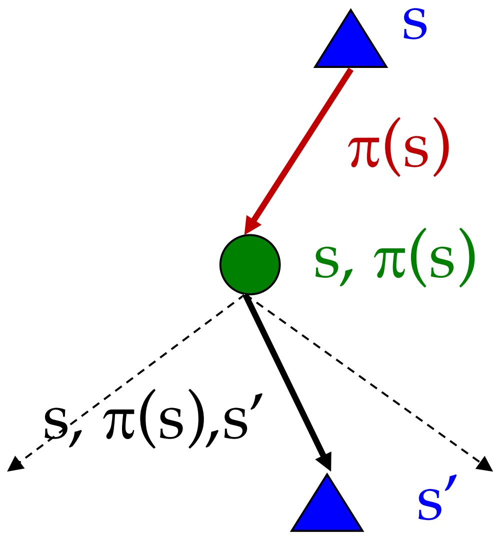
Policy Evaluation

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

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

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

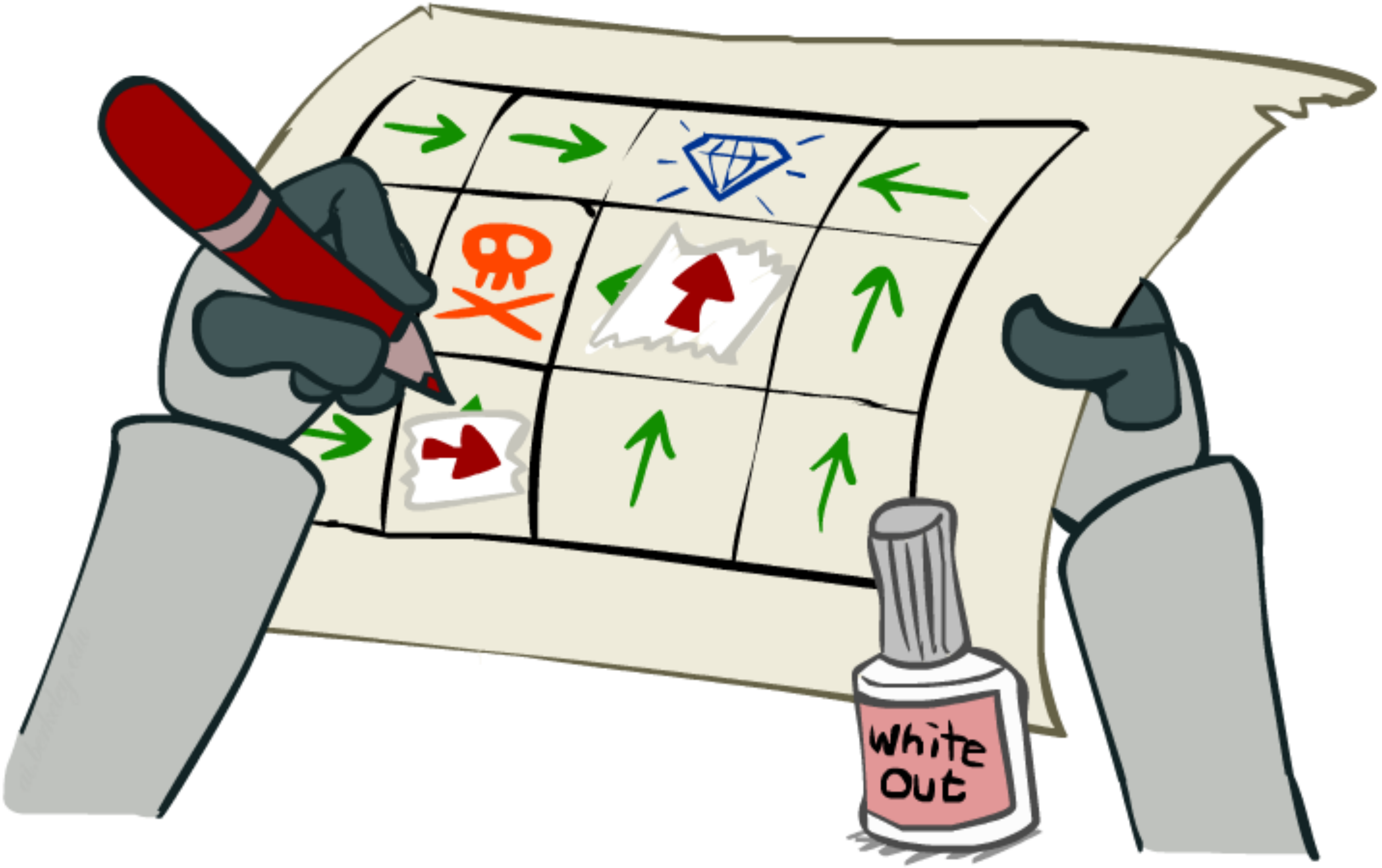
$$V_{k+1}^{\pi}(s) \leftarrow \sum_{s'} T(s, \pi(s), s')[R(s, \pi(s), s') + \gamma V_k^{\pi}(s')]$$



- 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 Iteration



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:

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

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

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

Example: Policy Iteration

π_0 | S | F | $\emptyset$

Policy Evaluation:

$$V^{\pi_0}(B) = 1 + 0.9 \cdot V^{\pi_0}(B) \quad \rightarrow V^{\pi_0}(B) = 10$$

$$V^{\pi_0}(W) = -10 + 0.9 \cdot V^{\pi_0}(O) \quad \rightarrow V^{\pi_0}(W) = -10$$

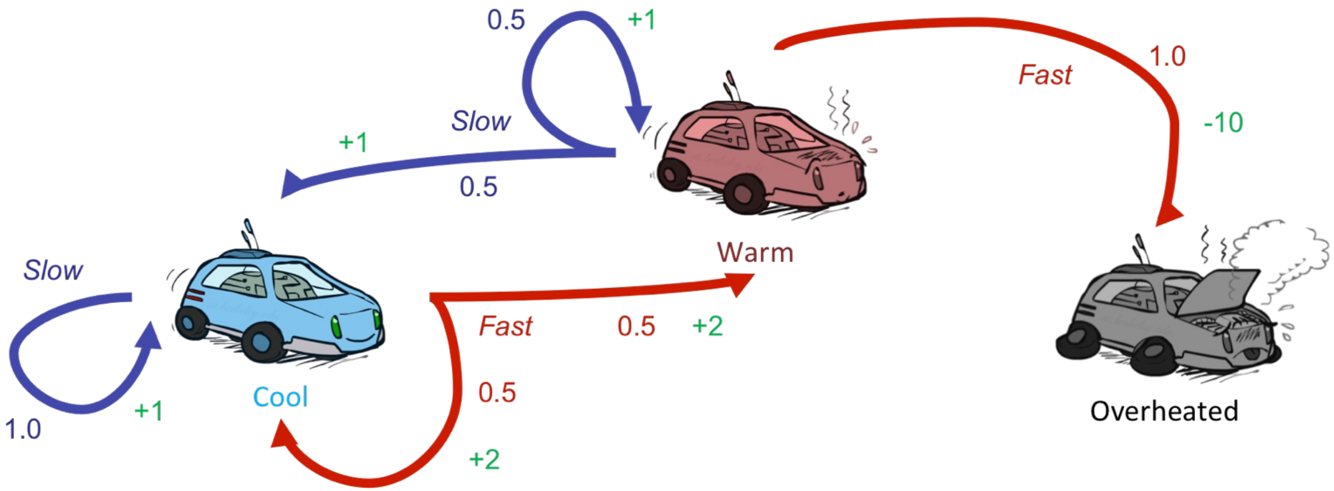
$$V^{\pi_0}(O) = 0$$

Policy Improvement:

π_1

$$S\colon\ 1 + 0.9 \cdot 10 = 10$$

$$F\colon\ 0.5(2 + 0.9 \cdot 10) + 0.5(2 + 0.9 \cdot -10) = 0$$



Assume discount = 0.9

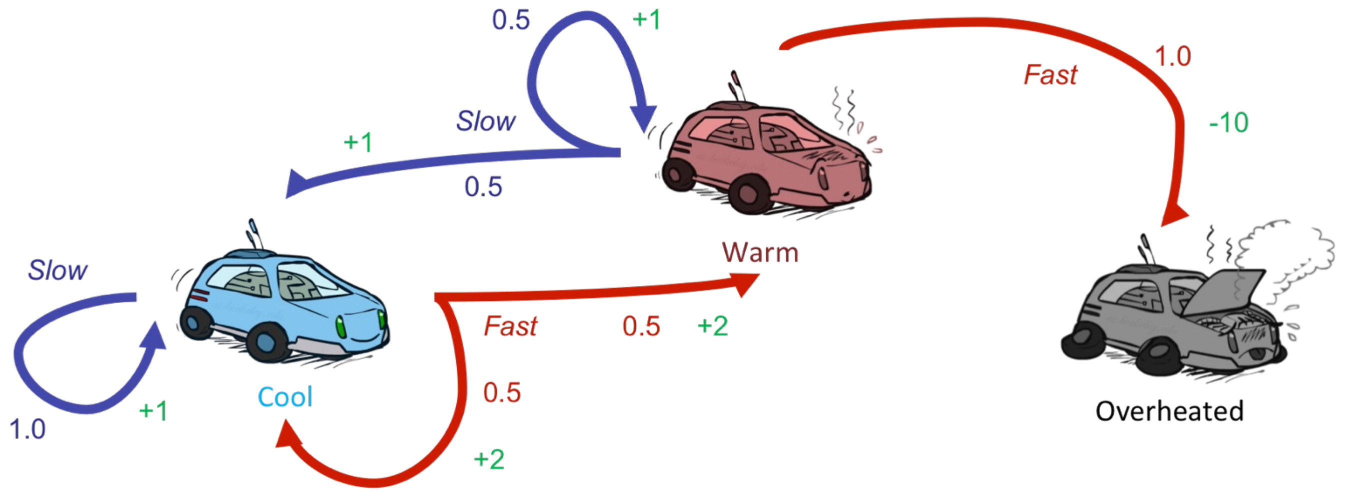
$$V^{\pi}(s) = \sum_{s'} T(s, \pi(s), s')[R(s, \pi(s), s') + \gamma V^{\pi}(s')]$$

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

Example: Policy Iteration

π_0	S	F	$\emptyset$



Policy Evaluation:

$V^{\pi_0}(B) = 1 + 0.9 \cdot V^{\pi_0}(B) \quad \rightarrow V^{\pi_0}(B) = 10$

$V^{\pi_0}(W) = -10 + 0.9 \cdot V^{\pi_0}(O) \quad \rightarrow V^{\pi_0}(W) = -10$

$V^{\pi_0}(O) = 0$

Assume discount = 0.9

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

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

Policy Improvement:

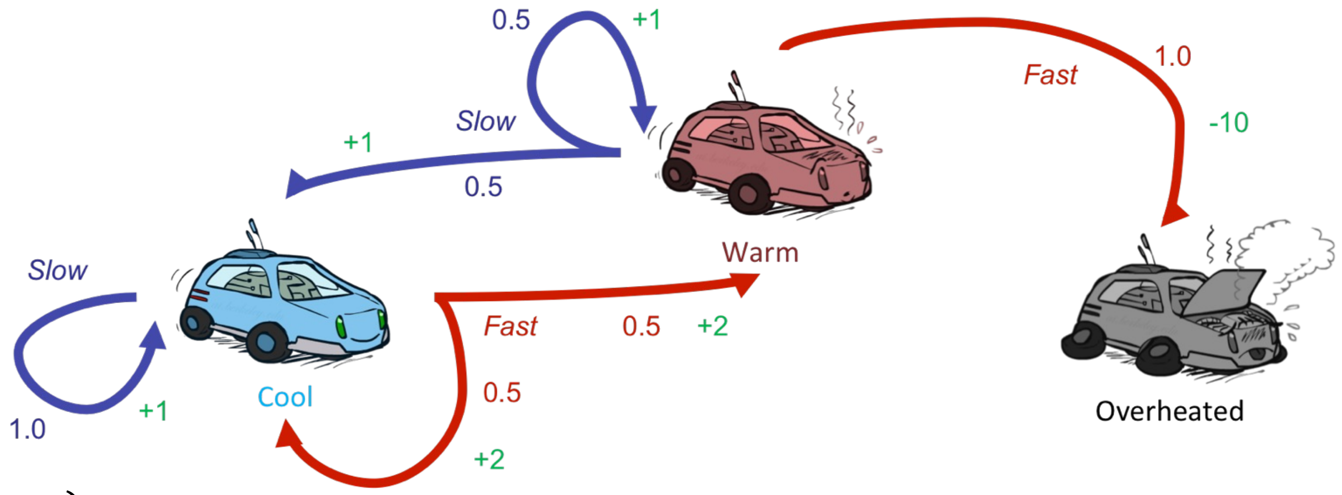
π_1	S	S	$\emptyset$

S: $0.5(1 + 0.9 \cdot 10) + 0.5(1 + 0.9 \cdot -10) = 0$

F: $-10 + 0.9 \cdot 0 = -10$

Example: Policy Iteration

π_1	S	S	$\emptyset$



Policy Evaluation:

$V^{\pi_0}(B) = 1 + 0.9 \cdot V^{\pi_0}(B) \quad \rightarrow V^{\pi_0}(B) = 10$

$V^{\pi_0}(W) = 0.5\big(1 + 0.9 \cdot V^{\pi_0}(B)\big) + 0.5\big(1 + 0.9 \cdot V^{\pi_0}(W)\big) \rightarrow V^{\pi_0}(W) = 10$

$V^{\pi_0}(O) = 0$

Assume discount = 0.9

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

Policy Improvement:

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

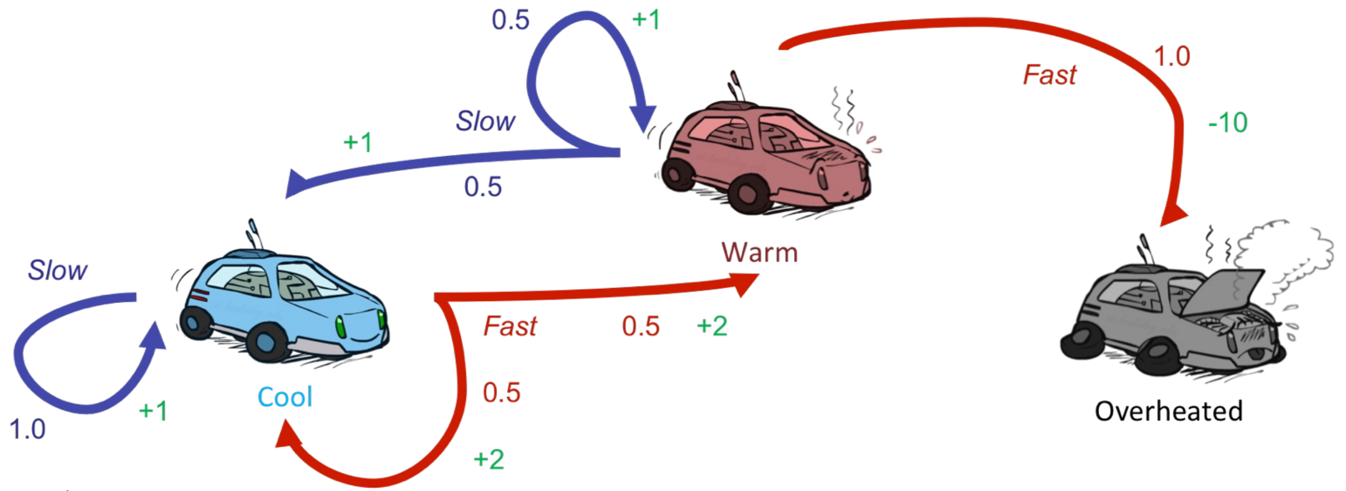
π_2

S: $1 + 0.9 \cdot 10 = 10$

F: $0.5(2 + 0.9 \cdot 10) + 0.5(2 + 0.9 \cdot 10) = 11$

Example: Policy Iteration

π_1	S	S	$\emptyset$



Policy Evaluation:

$$V^{\pi_0}(B) = 1 + 0.9 \cdot V^{\pi_0}(B) \quad \rightarrow V^{\pi_0}(B) = 10$$

$$V^{\pi_0}(W) = 0.5\big(1 + 0.9 \cdot V^{\pi_0}(B)\big) + 0.5\big(1 + 0.9 \cdot V^{\pi_0}(W)\big) \rightarrow V^{\pi_0}(W) = 10$$

$$V^{\pi_0}(O) = 0$$

Assume discount = 0.9

Policy Improvement:

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

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

π_2	F	S	$\emptyset$

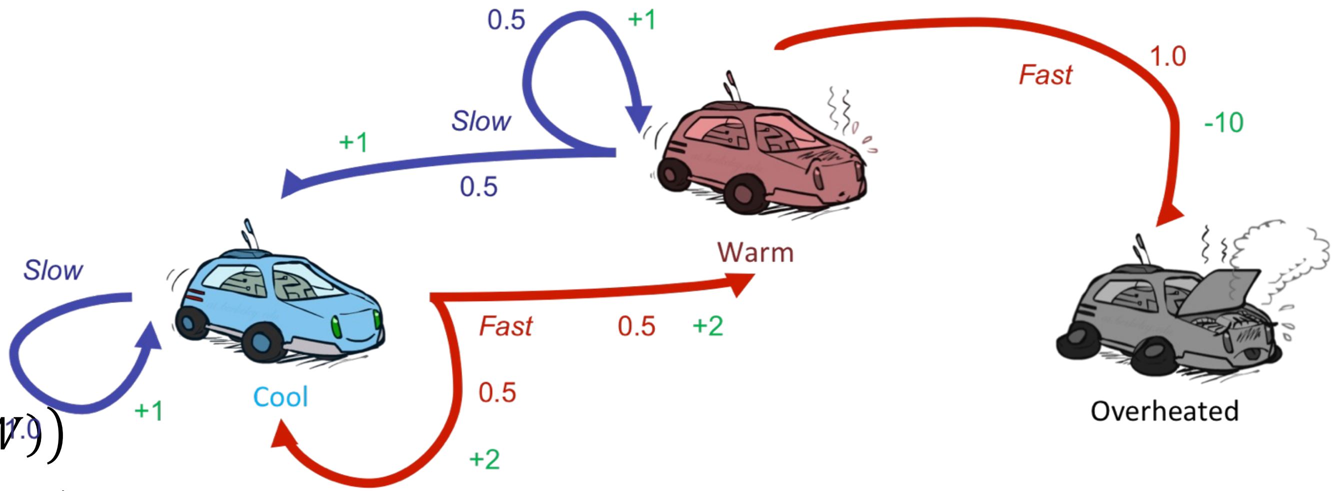
S: $0.5(1 + 0.9 \cdot 10) + 0.5(1 + 0.9 \cdot 10) = 10$

F: $-10 + 0.9 \cdot 0 = -10$

Example: Policy Iteration

π_2

F	S	Ø



Policy Evaluation:

$\rightarrow V^{\pi_0}(B) = 15.5$

$$V^{\pi_0}(B) = 0.5\big(2 + 0.9 \cdot V^{\pi_0}(B)\big) + 0.5\big(2 + 0.9 \cdot V^{\pi_0}(W)\big)$$

$$V^{\pi_0}(W) = 0.5\big(1 + 0.9 \cdot V^{\pi_0}(B)\big) + 0.5\big(1 + 0.9 \cdot V^{\pi_0}(W)\big) \rightarrow V^{\pi_0}(W) = 14.5$$

$$V^{\pi_0}(O) = 0$$

Assume discount = 0.9

Policy Improvement:

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

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

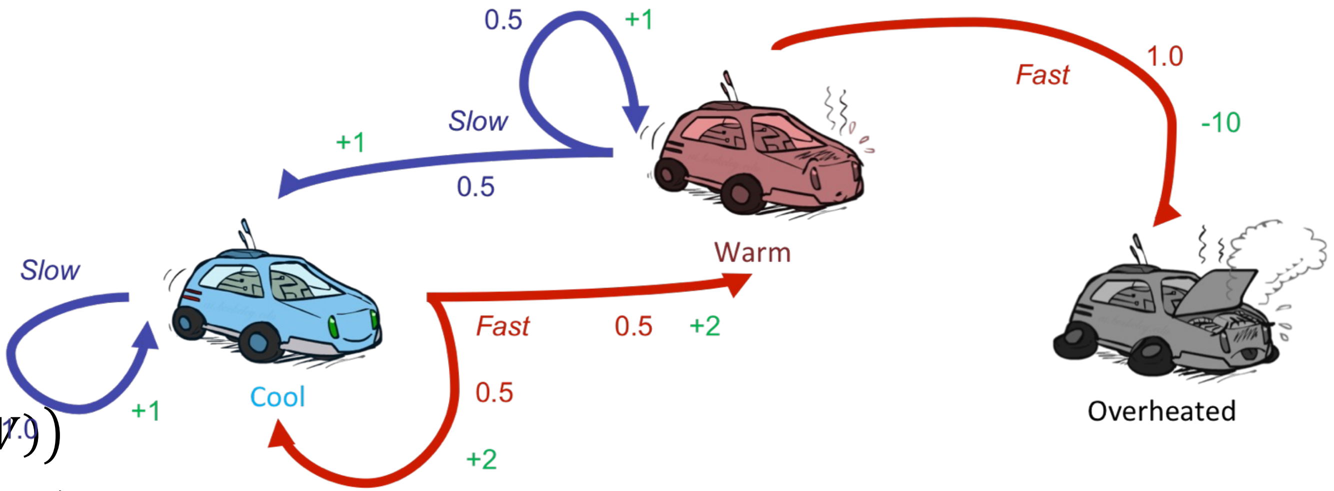
π_3

$S\colon\ 1 + 0.9 \cdot 15.5 = 14.95$

$F\colon\ 0.5(2 + 0.9 \cdot 15.5) + 0.5(2 + 0.9 \cdot 14.5) = 15.5$

Example: Policy Iteration

π_2	F	S	$\emptyset$



Policy Evaluation:

$$V^{\pi_0}(B) = 0.5\big(2 + 0.9 \cdot V^{\pi_0}(B)\big) + 0.5\big(2 + 0.9 \cdot V^{\pi_0}(W)\big) \rightarrow V^{\pi_0}\ (B) = 15.5$$

$$V^{\pi_0}(W) = 0.5\big(1 + 0.9 \cdot V^{\pi_0}(B)\big) + 0.5\big(1 + 0.9 \cdot V^{\pi_0}(W)\big) \rightarrow V^{\pi_0}\ (W) = 14.5$$

$$V^{\pi_0}(O) = 0$$

Assume discount = 0.9

Policy Improvement:

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

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

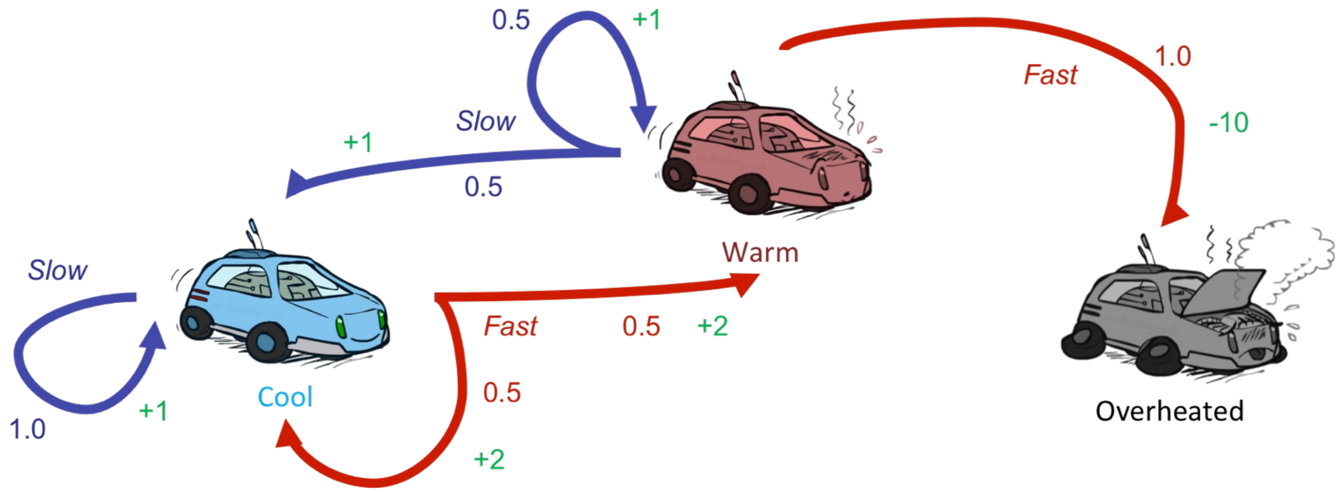
π_3	F	S	$\emptyset$

S: $0.5(1 + 0.9 \cdot 15.5) + 0.5(1 + 0.9 \cdot 14.5) = 14.5$

F: $-10 + 0.9 \cdot 0 = -10$

Example: Value Iteration

V_0	0	0	0
V_1	2	1	0
V_2	3.35	2.35	0
V_3	4.565	3.565	0
V_4	5.6585	4.6585	0



Assume discount = 0.9

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

…

V_{91}	15.499	14.499	0

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
  - Runtime per iteration: $O(|S|^2|A|)$
- 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)
    - Runtime per value iteration update: $O(|S|^2)$ → total runtime to get fixed policy values: $O(|S|^3)$
  - After policy is evaluated, a new policy is chosen (slow like a value iteration pass → $O(|S|^2|A|)$)
  - The new policy will be better (or we're done)
  - Runtime per iteration: $O(|S|^3) + O(|S|^2|A|)$ → slower but can take much fewer iterations
- Both are dynamic programs for solving MDPs

Convergence*

- Proof Sketch

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

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

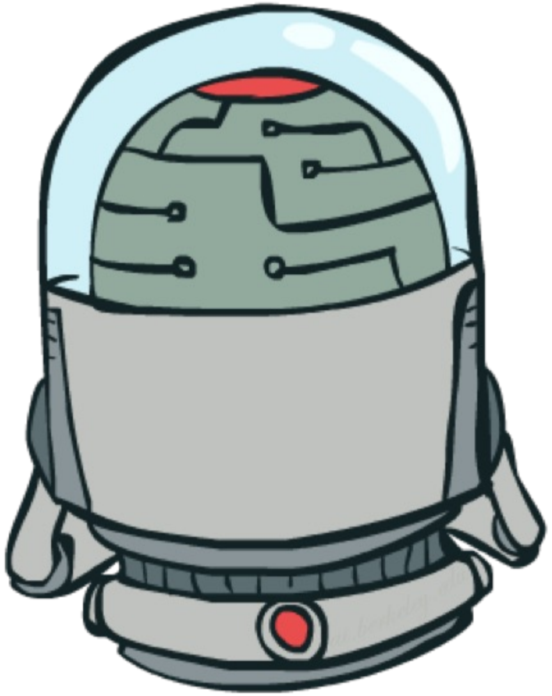
  - Monotonic improvement: $\forall s\ V^{\pi_{i+1}}(s) \geq V^{\pi_i}(s)$
  - Termination: π_i is optimal if $\forall s\ \pi_i(s) = \pi_{i+1}(s)$
    - $\pi_{i+1}(s)$ chooses the best action to take under $V^{\pi_i}(s)$
    - If $\forall s\ \pi_i(s) = \pi_{i+1}(s)$, then $\pi_i(s)$ was already the best action for all states
  - Guaranteed termination: only finite number of policies

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)

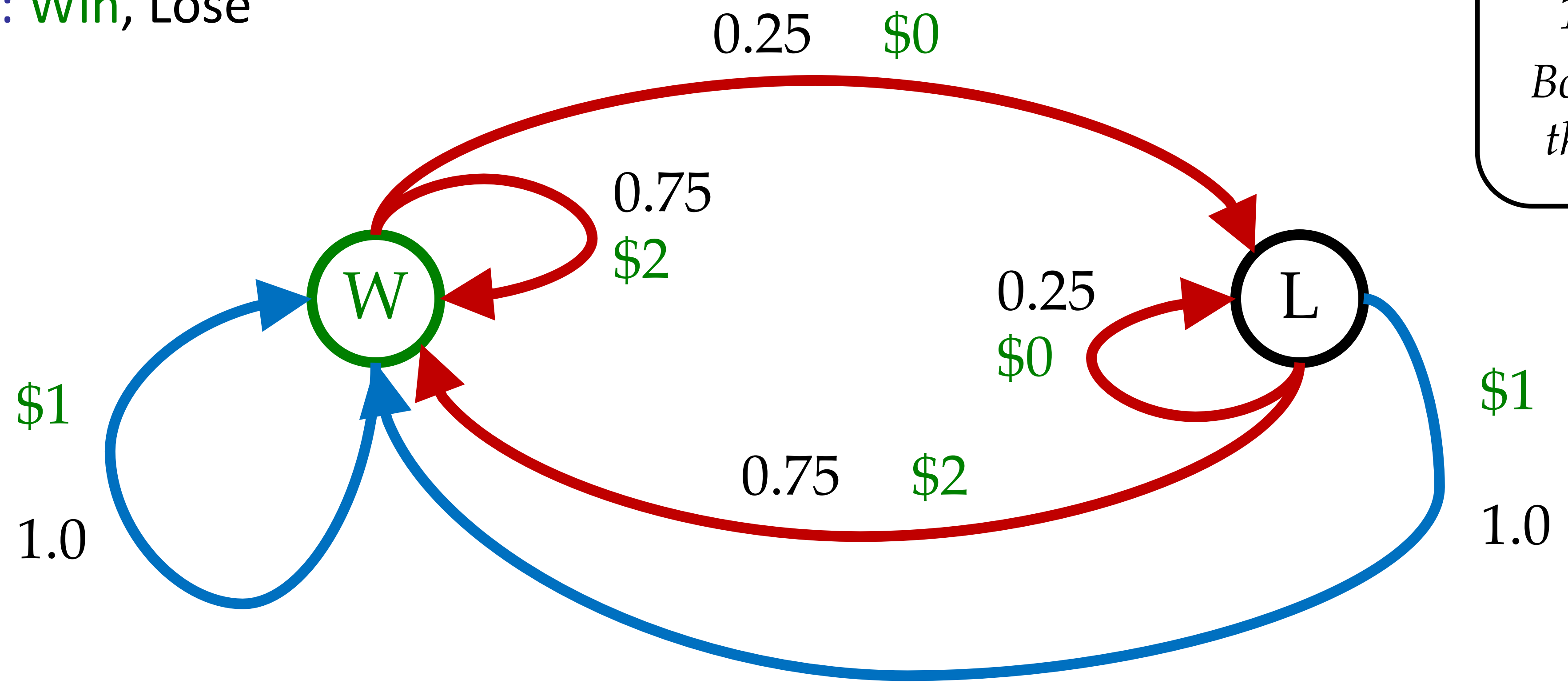
- These all look the same!
  - They basically are – they are all variations of Bellman updates
  - They all use one-step lookahead expectimax fragments
  - They differ only in whether we plug in a fixed policy or max over actions

Double Bandits



Double-Bandit MDP

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



No discount

10 time steps

Both states have the same value

Offline Planning

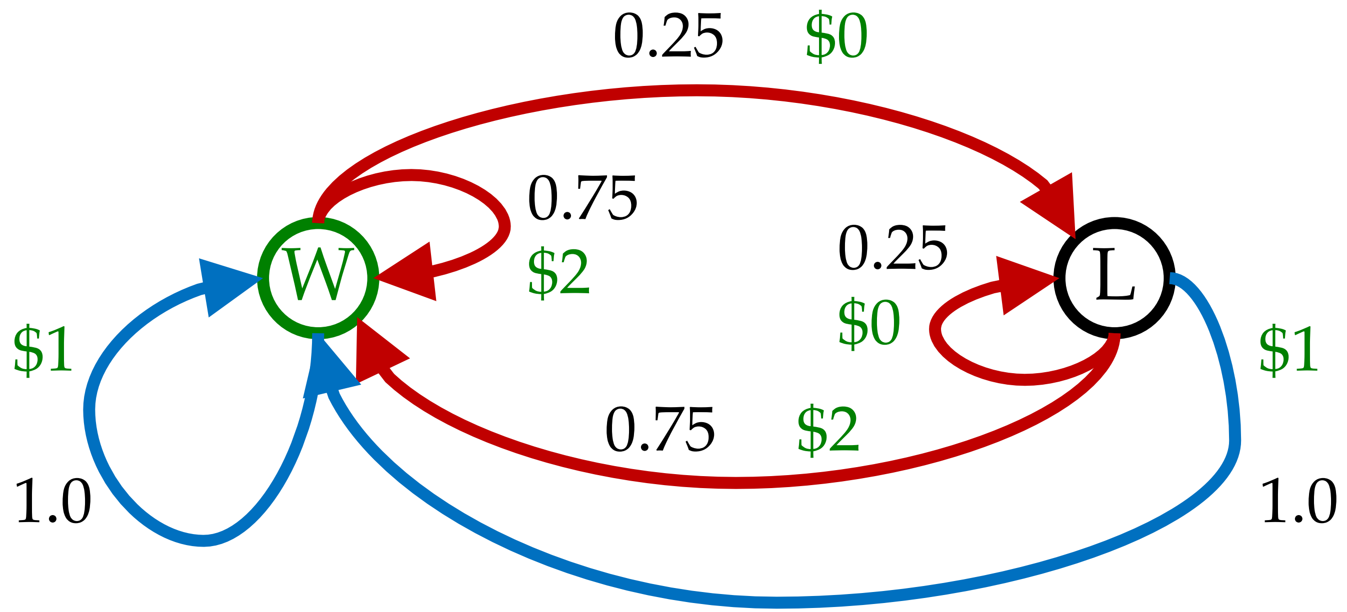
- Solving MDPs is offline planning
  - You determine all quantities through computation
  - You need to know the details of the MDP
  - You do not actually play the game!

No discount

10 time steps

Both states have the same value

	Value
Play Red	15
Play Blue	10



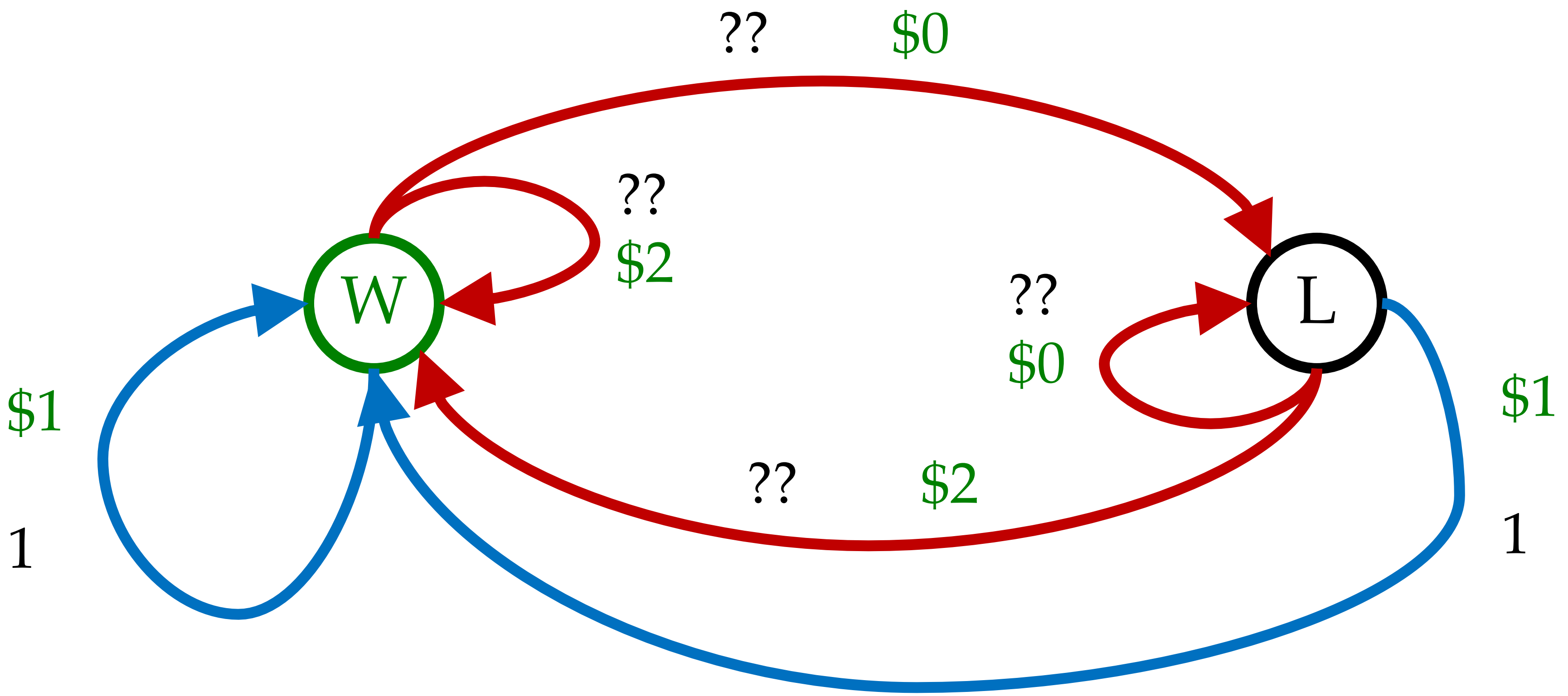
Let’s Play!

$2 $2 $0 $2 $2

$2 $2 $0 $0 $0

Online Planning

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



Let’s Play!

\$1 \$1 \$1

\$0 \$0 \$0 \$2

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

Reinforcement Learning

- Still assume a Markov decision process (MDP):
  - A set of states $s \in S$
  - A set of actions (per state) A
  - A model $T(s,a,s')$
  - A reward function $R(s,a,s')$
- Still looking for a policy $\pi(s)$

- New twist: don't know T or R
  - I.e. we don't know which states are good or what the actions do
  - Must actually try actions and states out to learn