Markov Decision Processes (MDPs)

Russell and Norvig: Chapter 17.1-17.3, 21
CSE 240: Winter 2023
Lecture 16

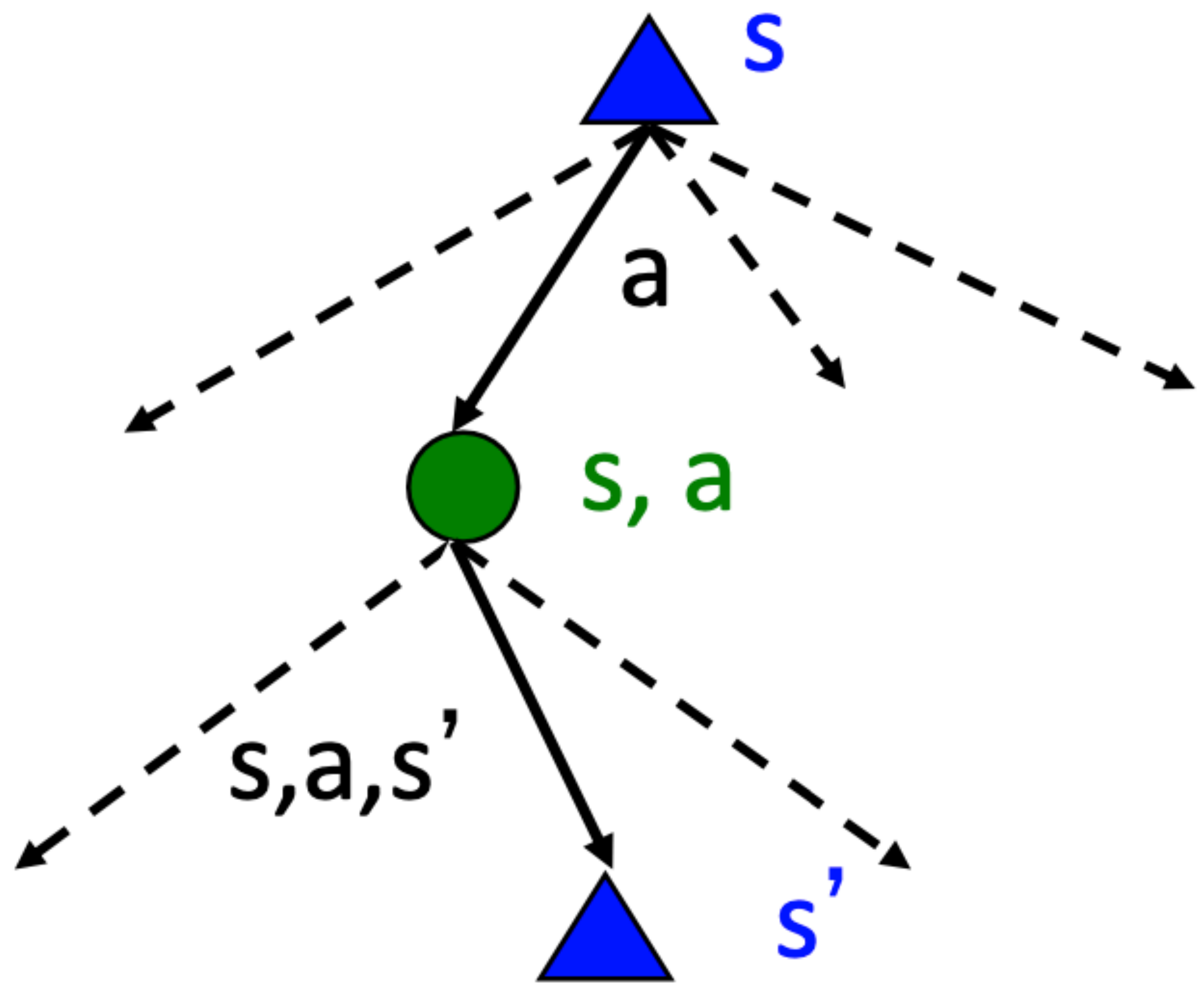
Announcements

- Assignment 4 is posted
- <u>We will *not* have class on Tuesday March 7.</u>
  - Quiz 3 will open on Tuesday and it is due Wednesday March 8th at 5pm.
  - Will post review materials on Friday (tomorrow).
  - I will hold additional office hours on Monday at 4pm.
- I will go over survey feedback on today.

Agenda and Topics

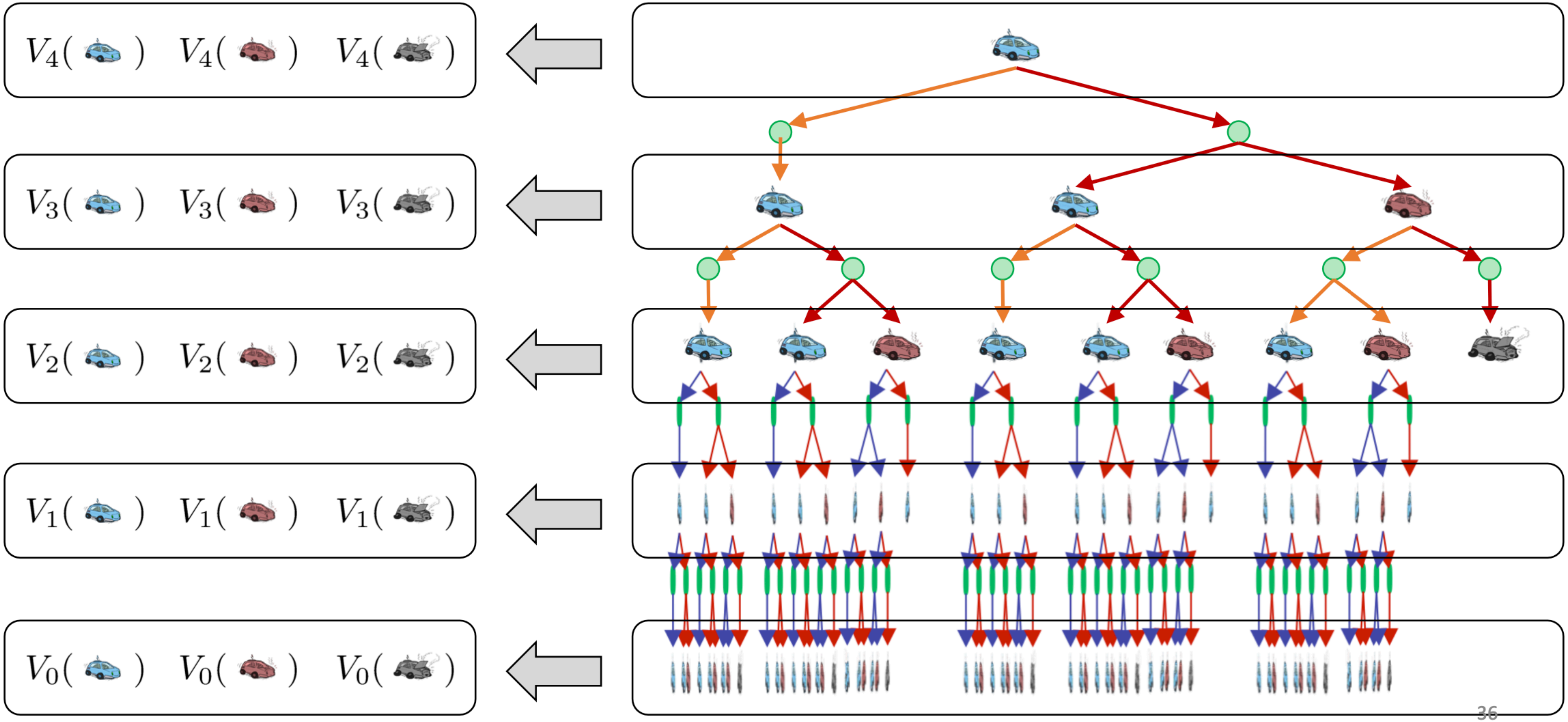
- Markov Decision Processes (MDP)
    - Value Iteration
    - Policy Iteration
- Go over class feedback
- Intro to RL (Not on the quiz)

Values of States



- Recursive definition of value:
  - $V^*(s) = \max_a Q^*(s, a)$
  - $Q^*(s, a) = \sum_{s'} T(s, a, s')\left[R(s, a, s') + \gamma V^*(s')\right]$
  - $V^*(s) = \max_a \sum_{s'} T(s, a, s')\left[R(s, a, s') + \gamma V^*(s')\right]$

Computing Time-Limited Values



Value Iteration

Value Iteration

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

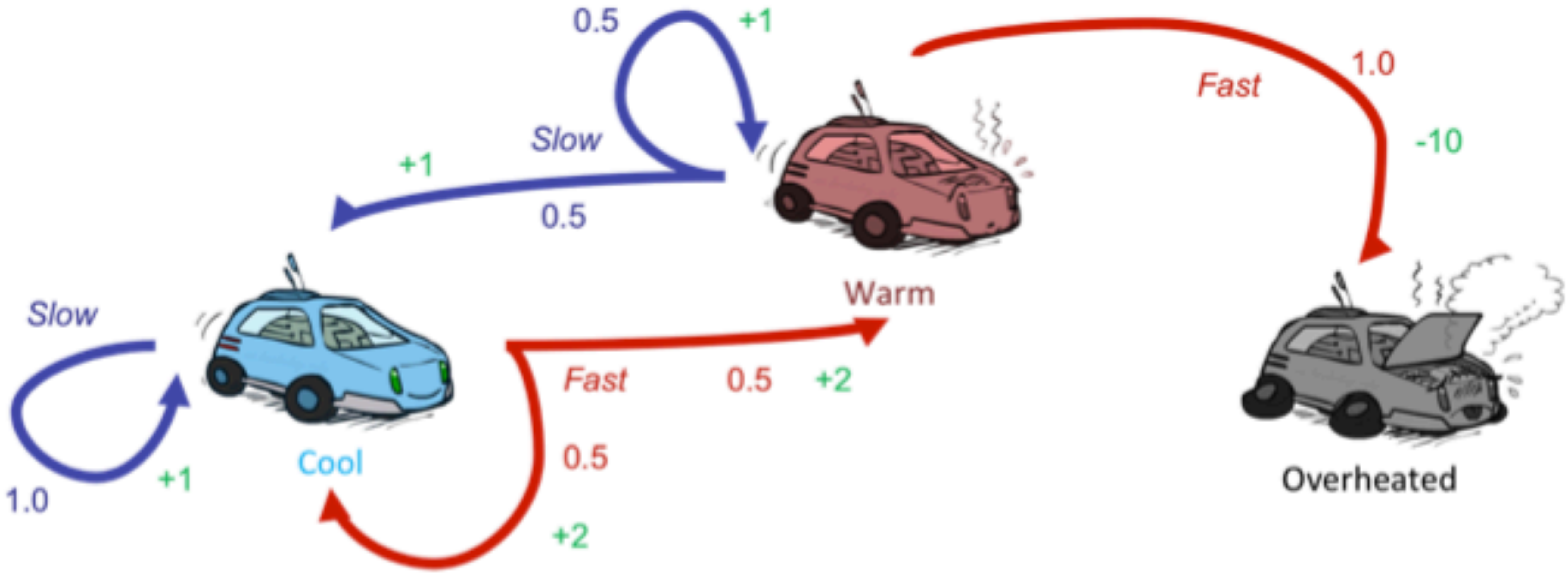
$$V_{i+1}(s) \leftarrow \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V_i(s')\right]$$

  - Throw out old V_i values
  - This is called a value update or Bellman update
  - 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			
V_1			
V_0	0	0	0

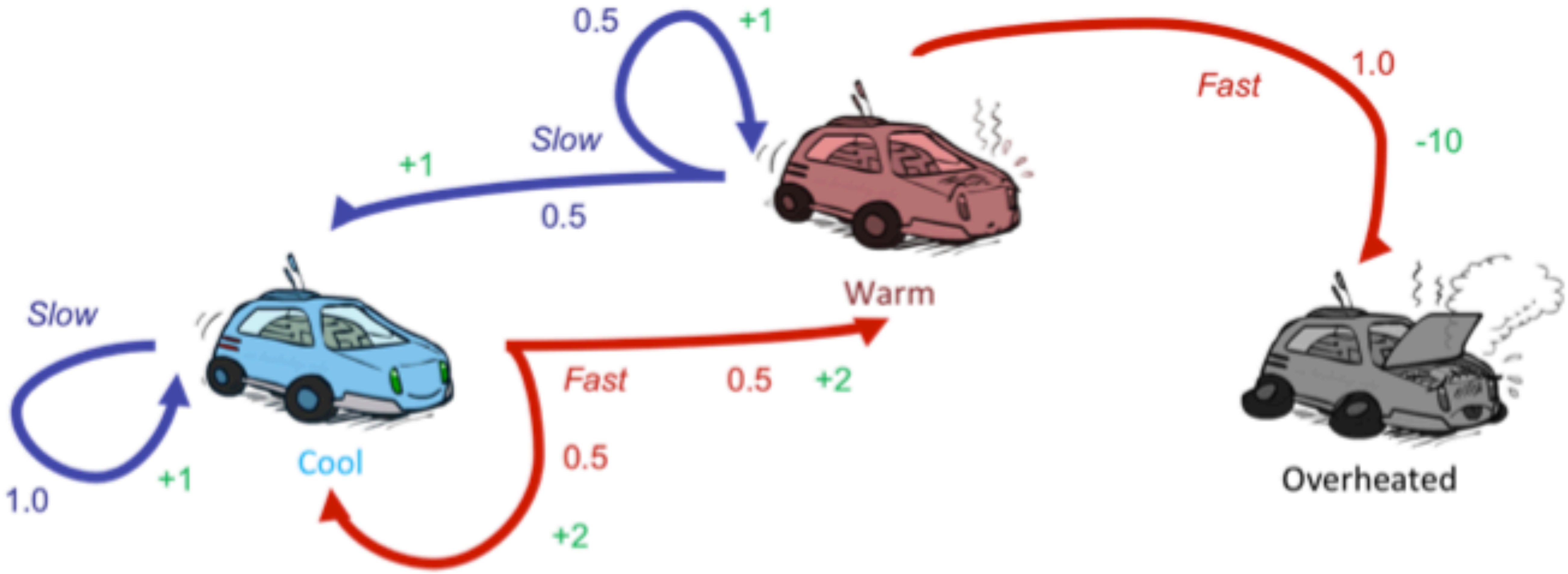


Assume no discount!

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

Example: Value Iteration

V_2			
V_1	2		
V_0	0	0	0

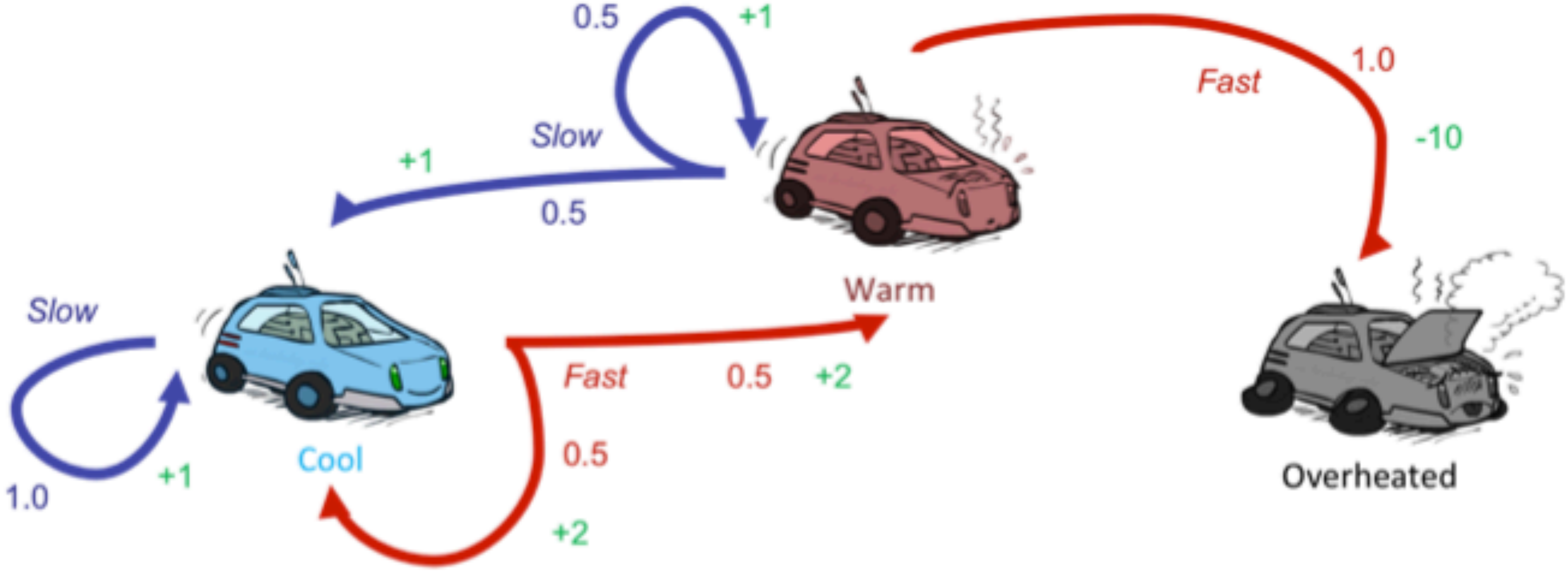


Assume no discount!

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

Example: Value Iteration

V_2			
V_1	2	1	0
V_0	0	0	0



Assume no discount!

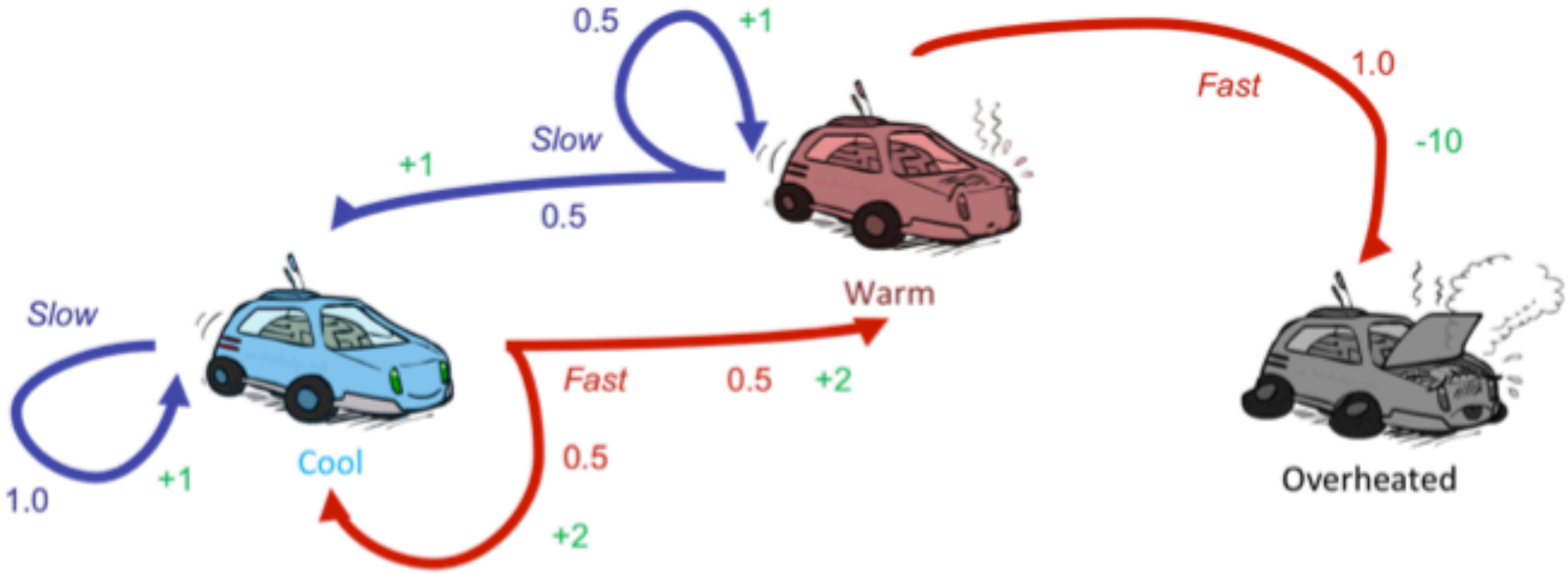
$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V_k(s') \right]$$

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

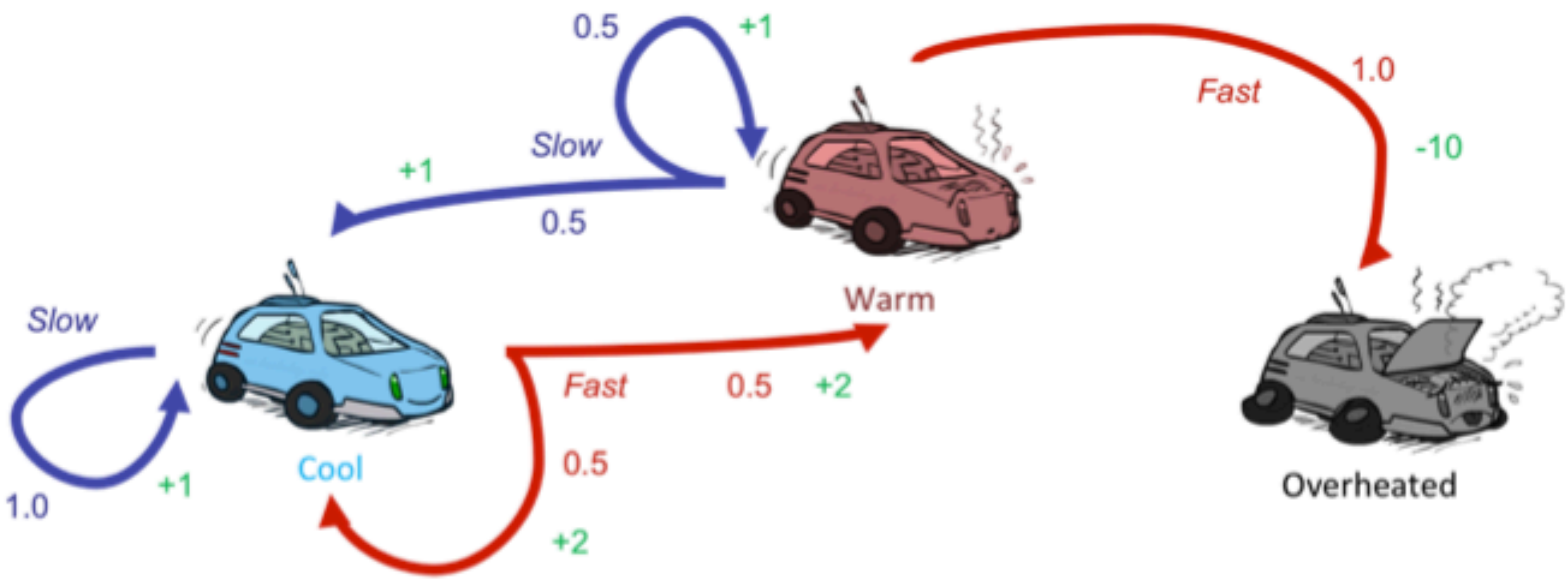


Assume no discount!

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

Example: Value Iteration

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



Assume no discount!

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

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)

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

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

Policy Methods

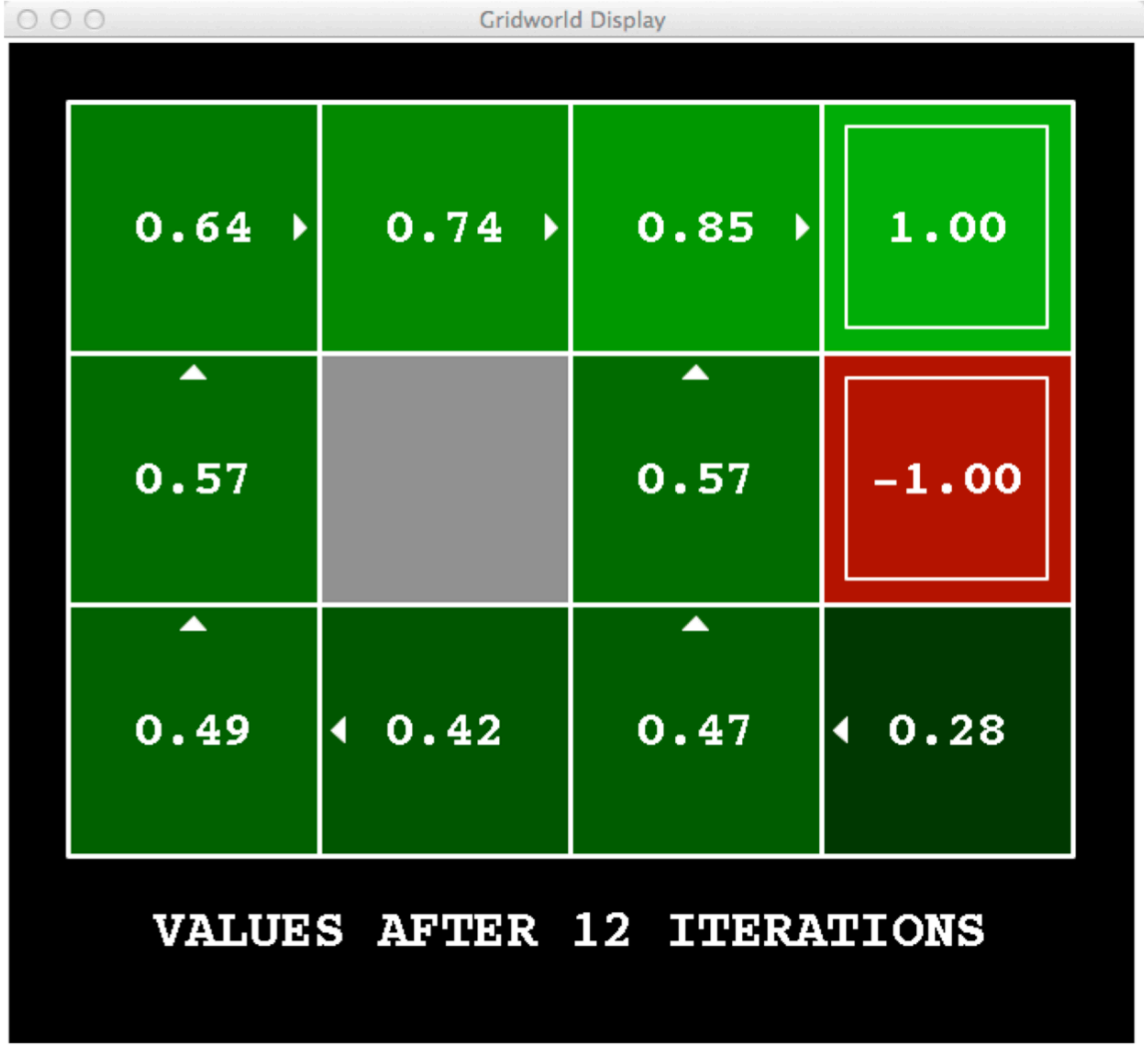
Problems with Value Iteration

- Value iteration repeats the Bellman updates:

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

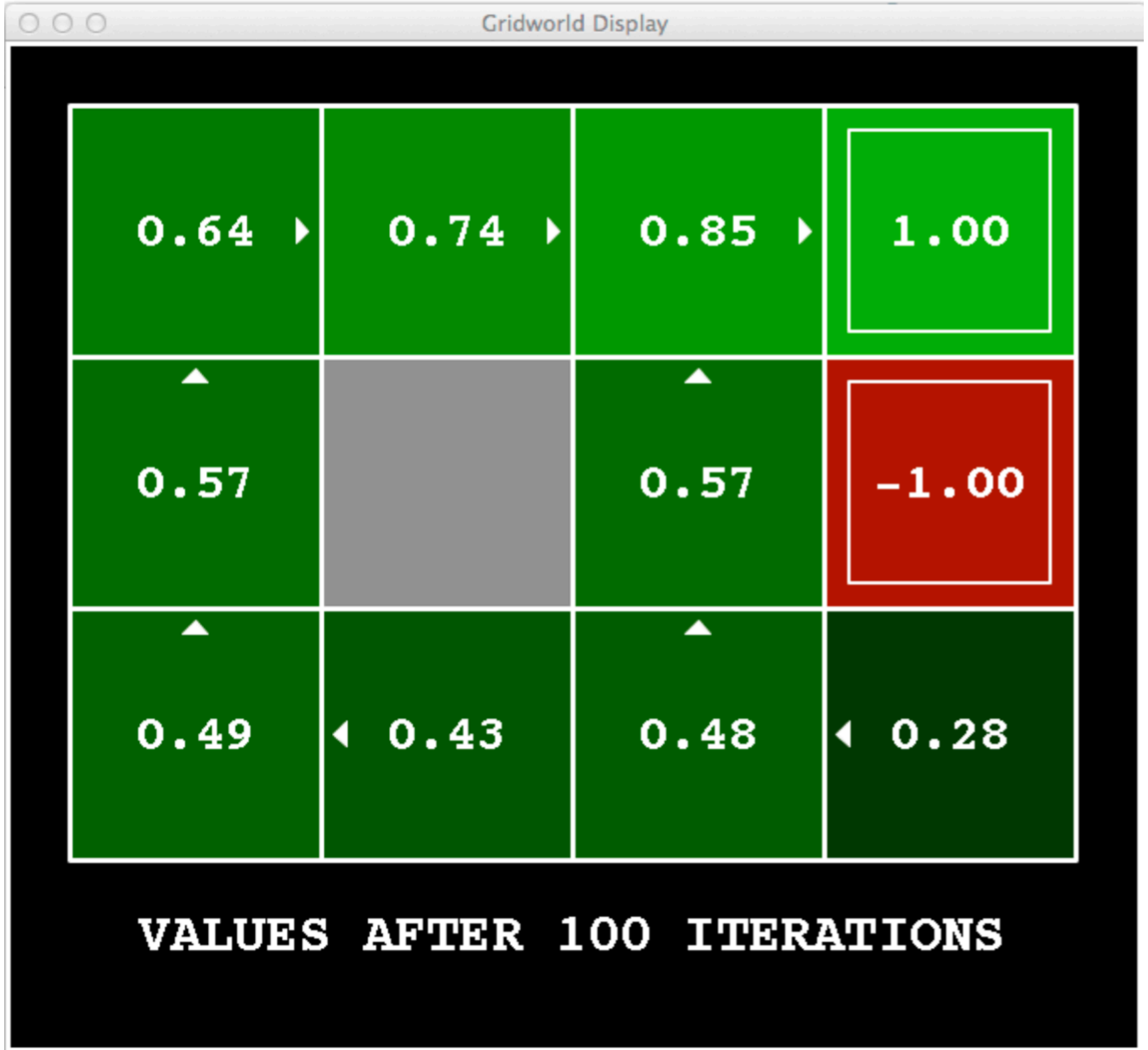
- Problem 1:
  - It's slow
  - $O(S^2 A)$ per iteration
- Problem 2:
  - The "max" at each state rarely changes
- Problem 3:
  - The policy often converges long before the values

K=12



Noise = 0.2
Discount = 0.9
Living reward = 0

K=100



Noise = 0.2
Discount = 0.9
Living reward = 0

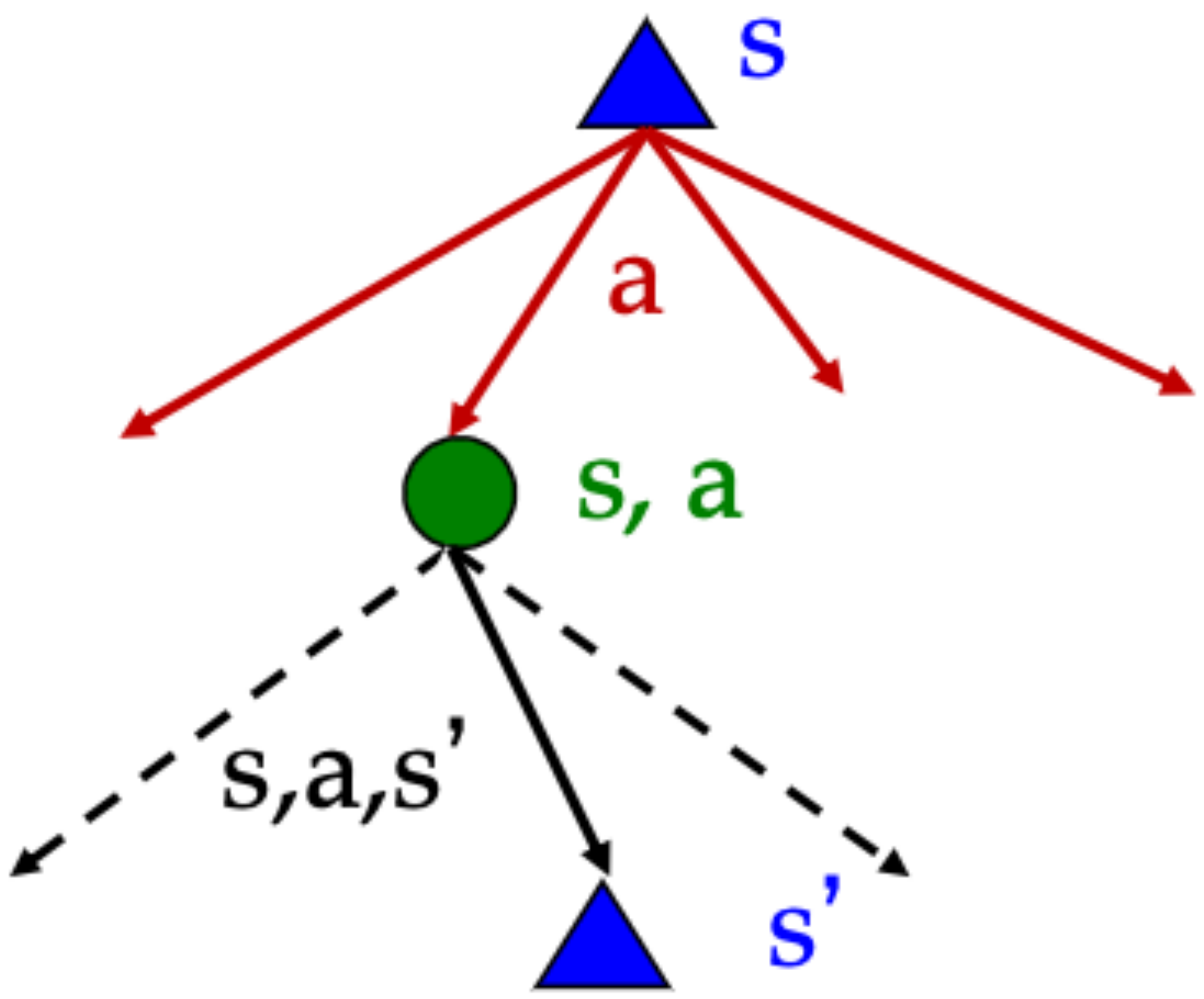
Policy Iteration

- Alternative to value iteration:
  - Step 1: Policy evaluation: calculate utilities for a fixed policy (not optimal utilities!) until convergence (fast)
  - Step 2: Policy improvement: update policy using one-step lookahead with resulting converged (but not optimal!) utilities (slow but infrequent)
  - Repeat steps until policy converges
- This is policy iteration
  - It's still optimal!
  - Can converge faster under some conditions

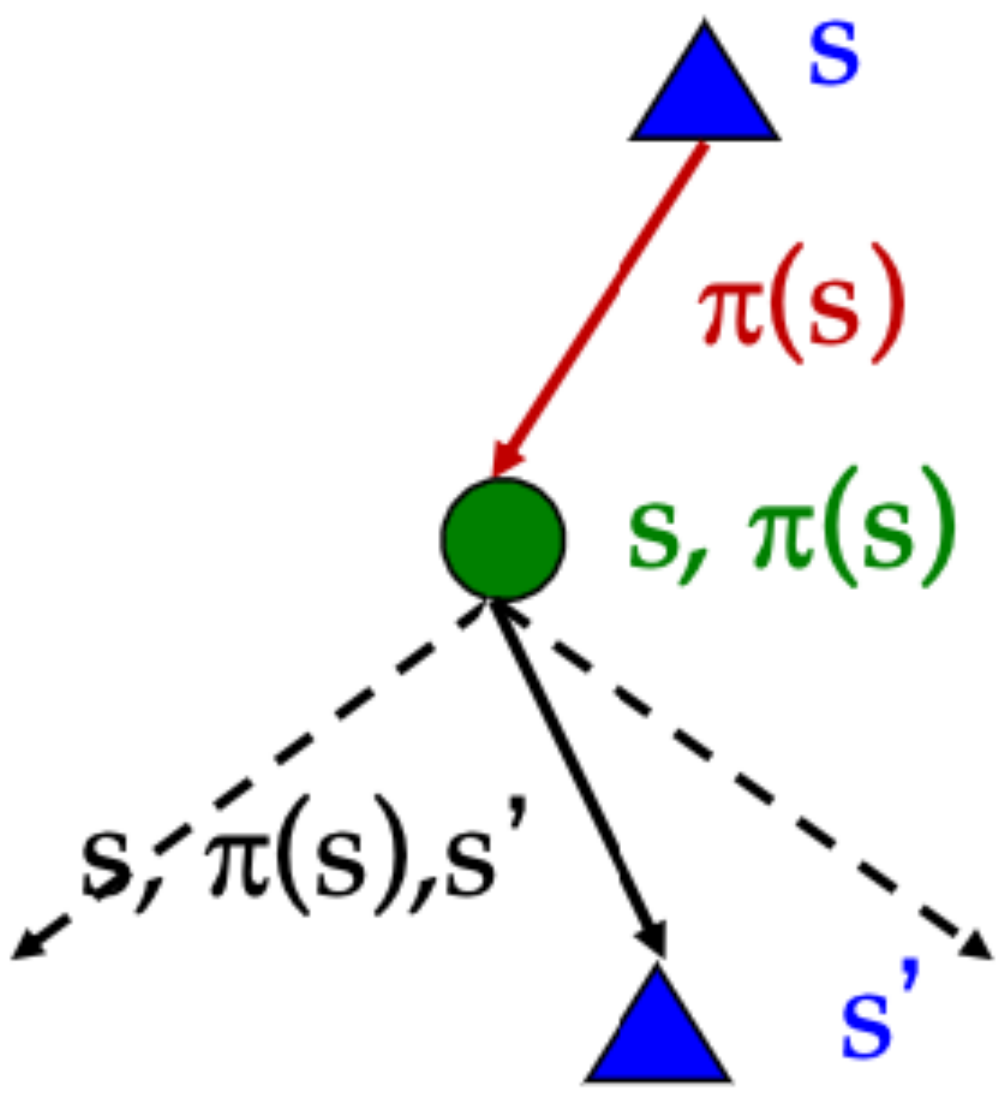
Policy Evaluation

Fixed Polices

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 π(s), then the tree would be simpler – only one action per state
  - … though the tree's value would depend on which policy we fixed

Fixed Polices

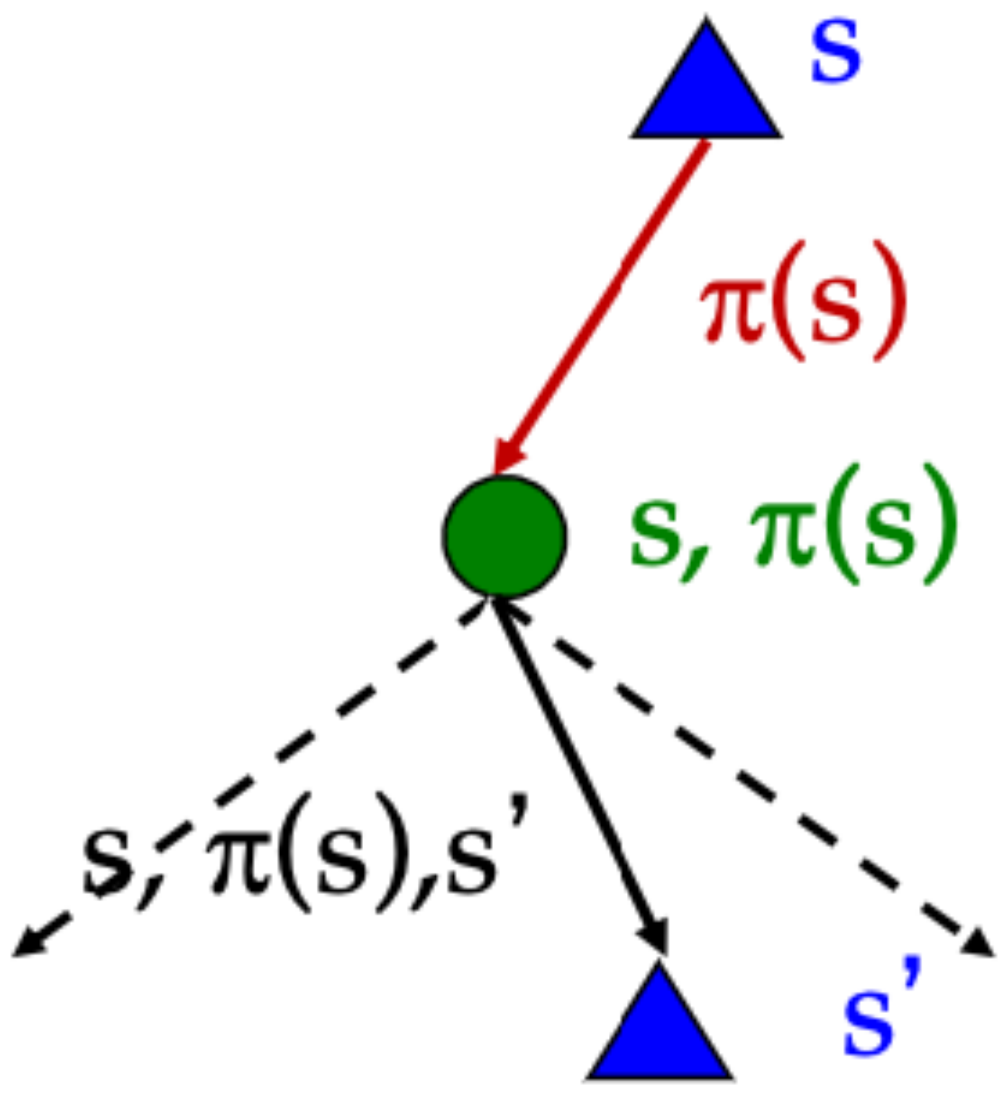
- Another basic operation: compute the utility of a state s under a fixed (generally non-optimal) policy

- 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.

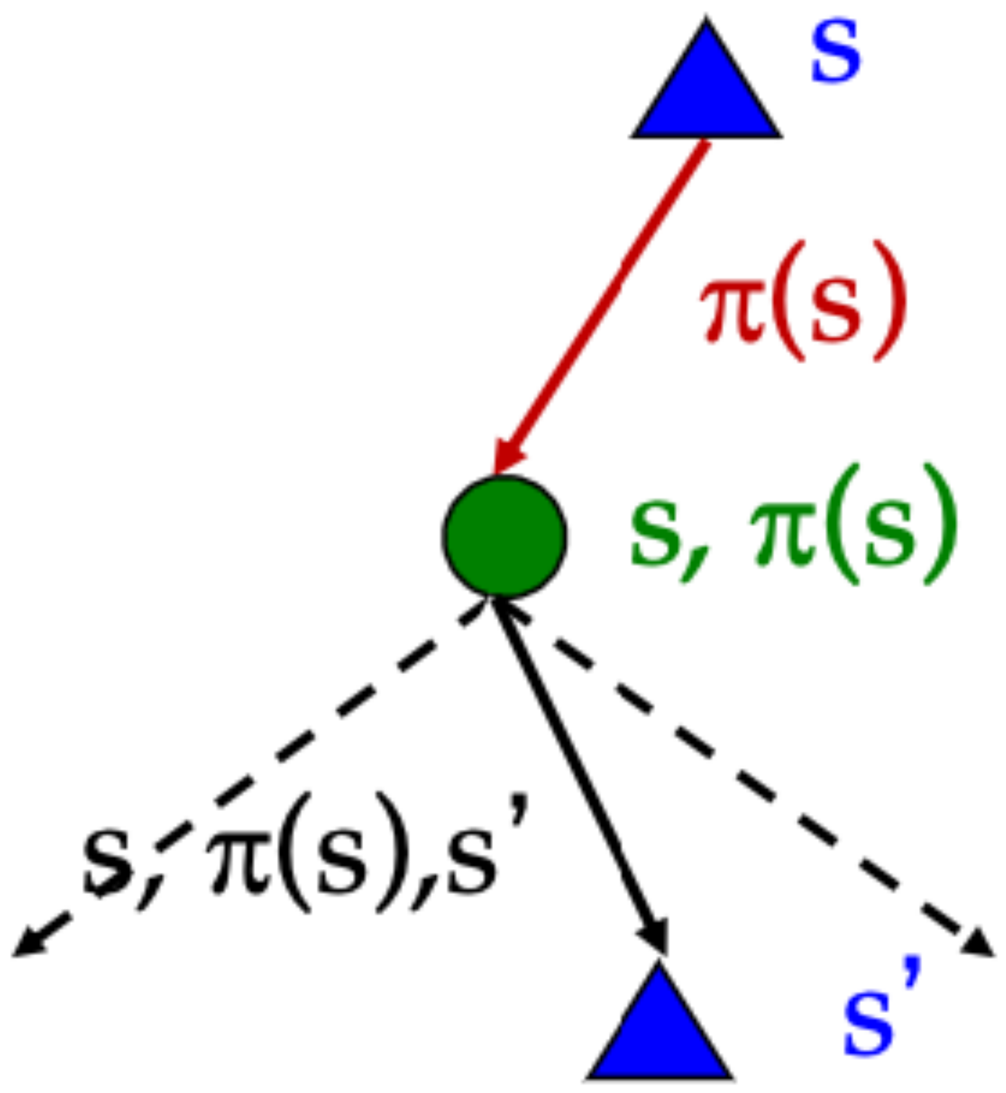
- Recursive relation (one-step look-ahead/Bellman equation):

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



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):
  - $V_0^\pi(s) = 0$
  - $V_{k+1}^\pi(s) \leftarrow \sum_{s'} T(s, \pi(s), s')\left[R(s, \pi(s), s') + \gamma V_k^\pi(s')\right]$
- Efficiency: $O(S^2)$ per iteration
- Idea 2: Without the maxes, the Bellman equations are just a linear system.



Policy Iteration

Policy Iteration

- **Evaluation:** with fixed current policy π, find values with simplified Bellman updates:
  - Iterate until values converge (or just solve the eqns directly!)

$$V_{i+1}^{\pi_k}(s) \leftarrow \sum_{s'} T(s, \pi_k(s), s') \left[R(s, \pi_k(s), s') + \gamma V_i^{\pi_k}(s')\right]$$

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

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

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

CE 16: Value Iteration

- Describe the Value Iteration algorithm and its rationale.

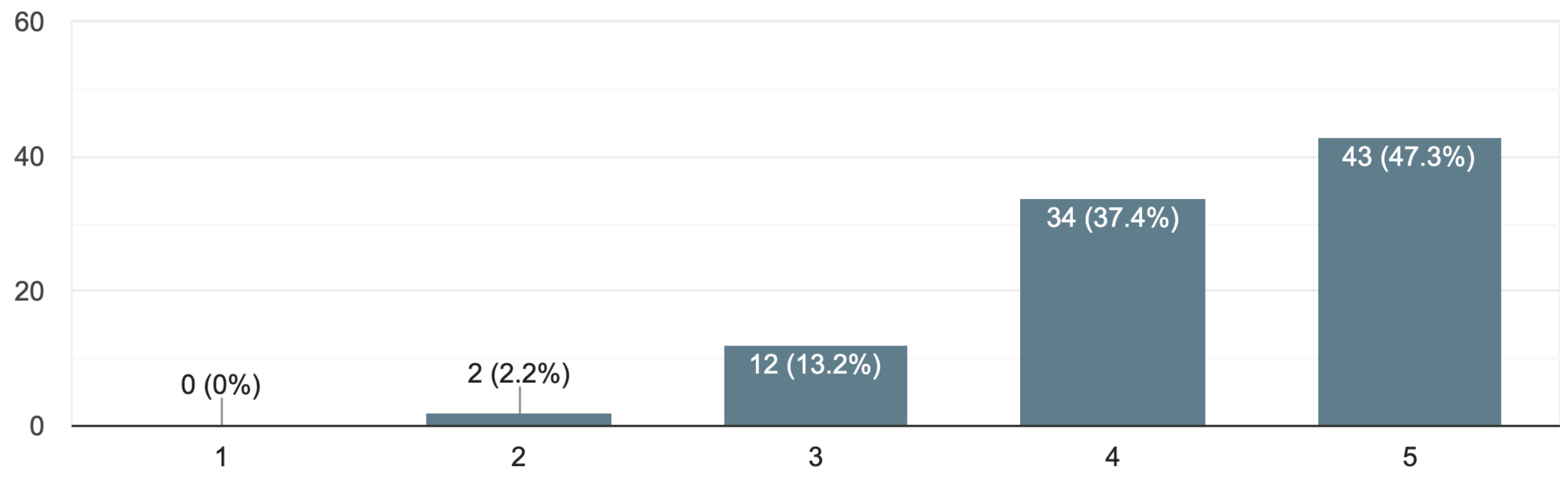
Survey

First of all; thanks!

- The teaching team and I are very open to feedback
- We appreciate you taking the time to help make the class better!
- Disclaimer
    - Large graduate class (we are doing our best!)
    - A lot of remote participations
    - Different abilities

1. Choose the appropriate representation for an AI problem or domain model (e.g., BFS versus DFS)

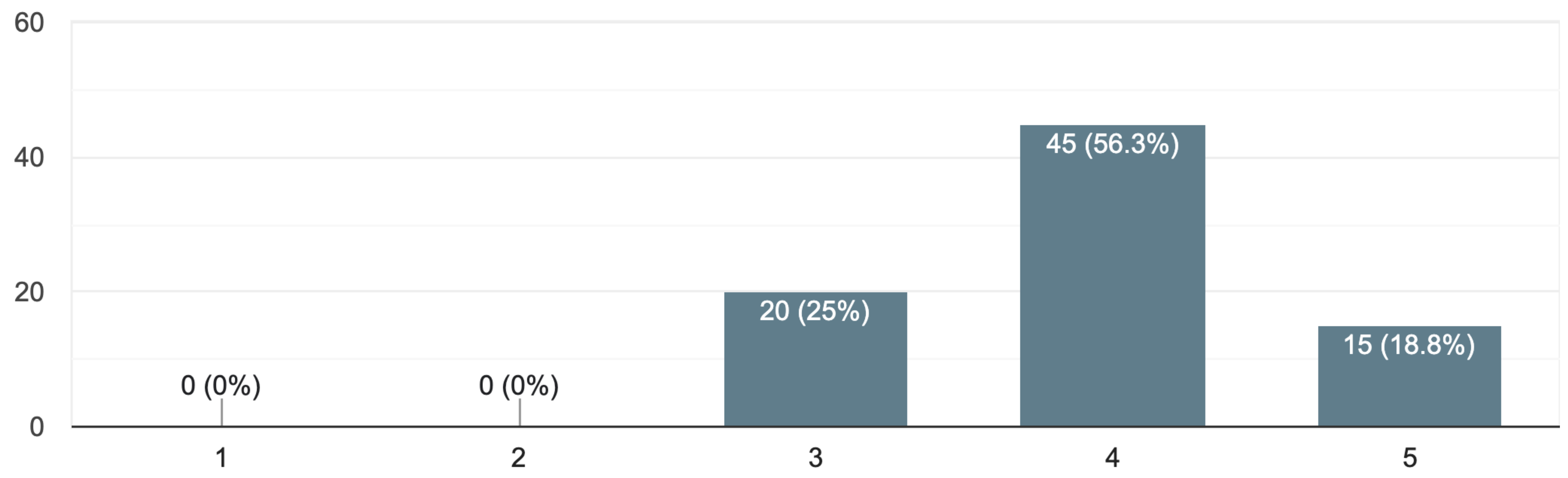
91 responses



Week 4

1. Choose the appropriate representation for an AI problem or domain model (e.g., uninformed search versus adversarial search)

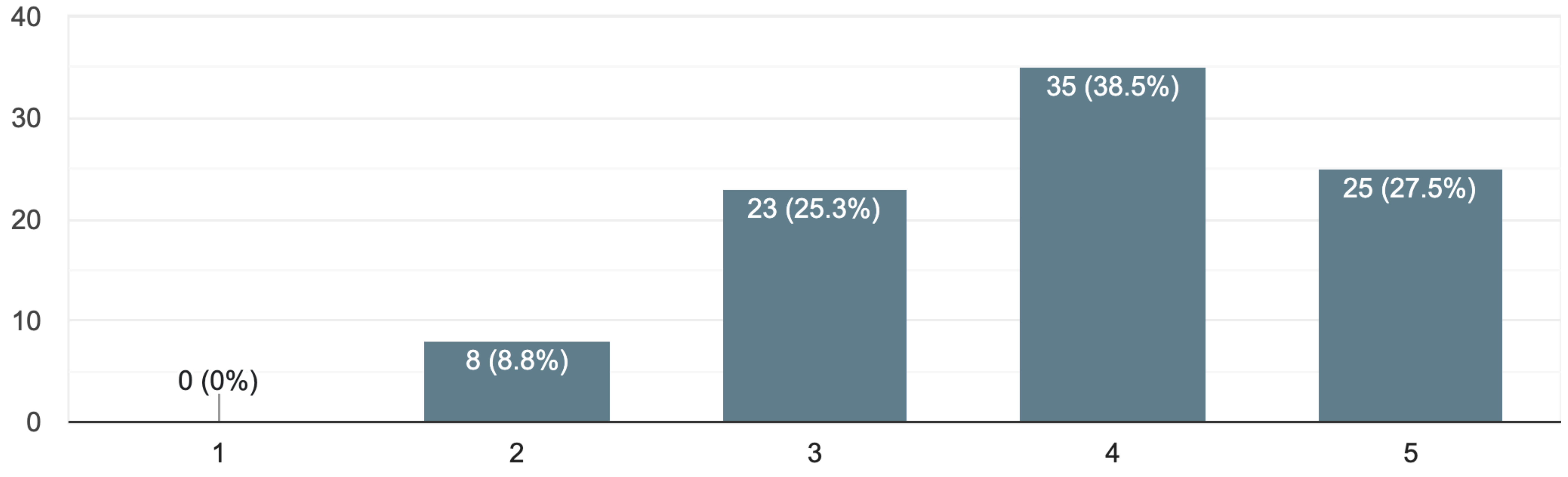
80 responses



Week 8

2. Implement and debug core AI algorithms in a clean and structured manner (e.g., alpha-beta pruning, A*, etc.)

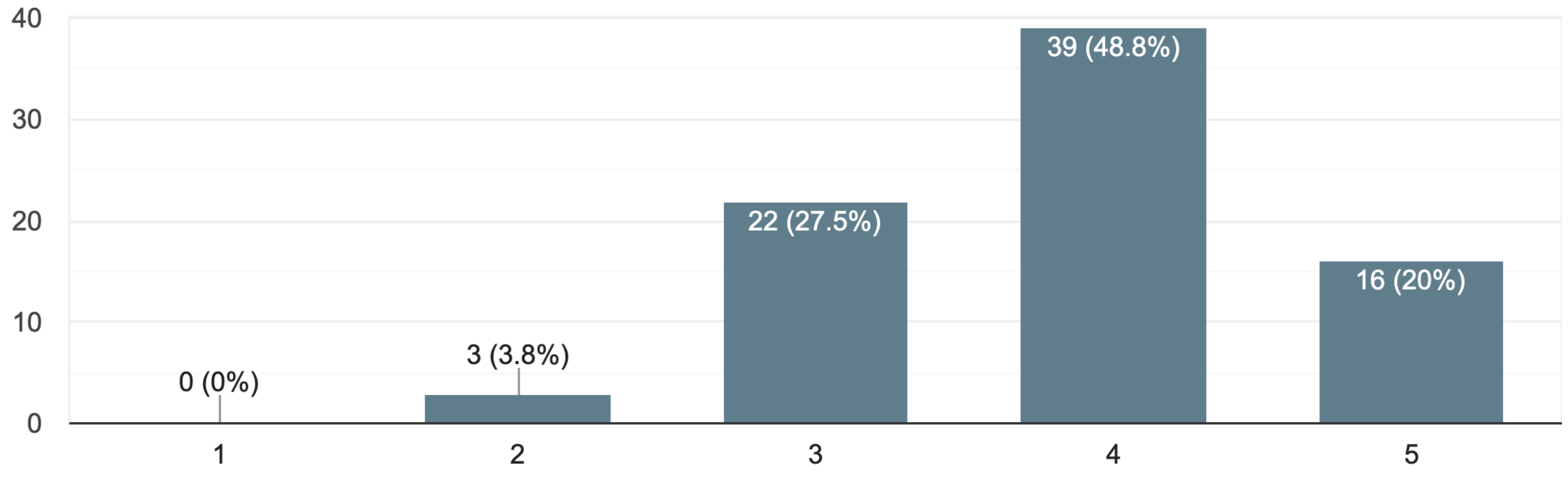
91 responses



Week 4

2. Implement and debug core AI algorithms in a clean and structured manner (e.g., CSPs, minimax, A*, etc.)

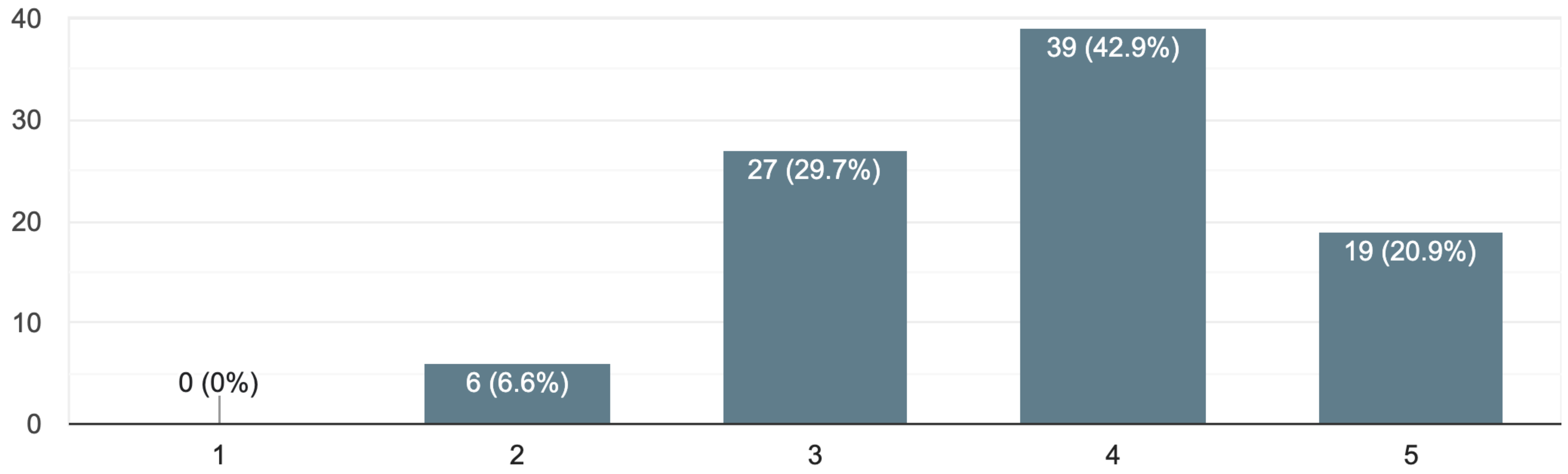
80 responses



Week 8

3. Design and analyze the performance of an AI system or component

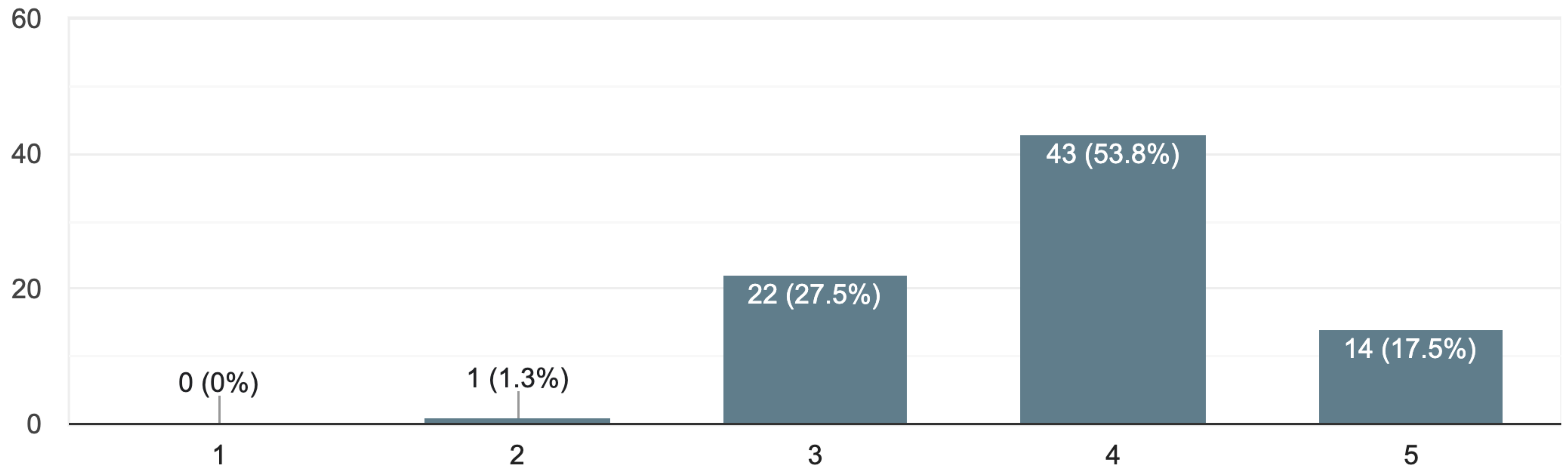
91 responses



Week 4

3. Design and analyze the performance of an AI system or component (e.g., examine the local search traces)

80 responses



Week 8

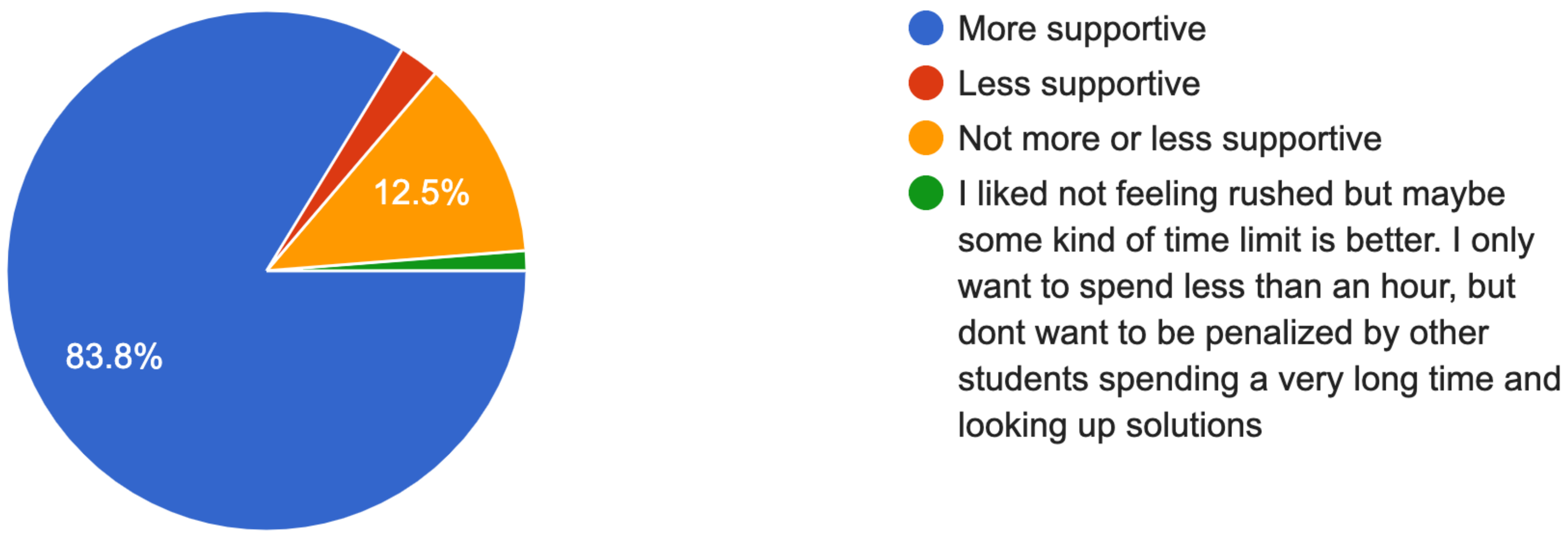
4. What modality of examination would be most supportive to test your conceptual knowledge of core concepts?

91 responses

Week 4

4. Did the latest changes to the examination (untimed quizzes) provide more/less/no change in support your conceptual knowledge of core concepts?

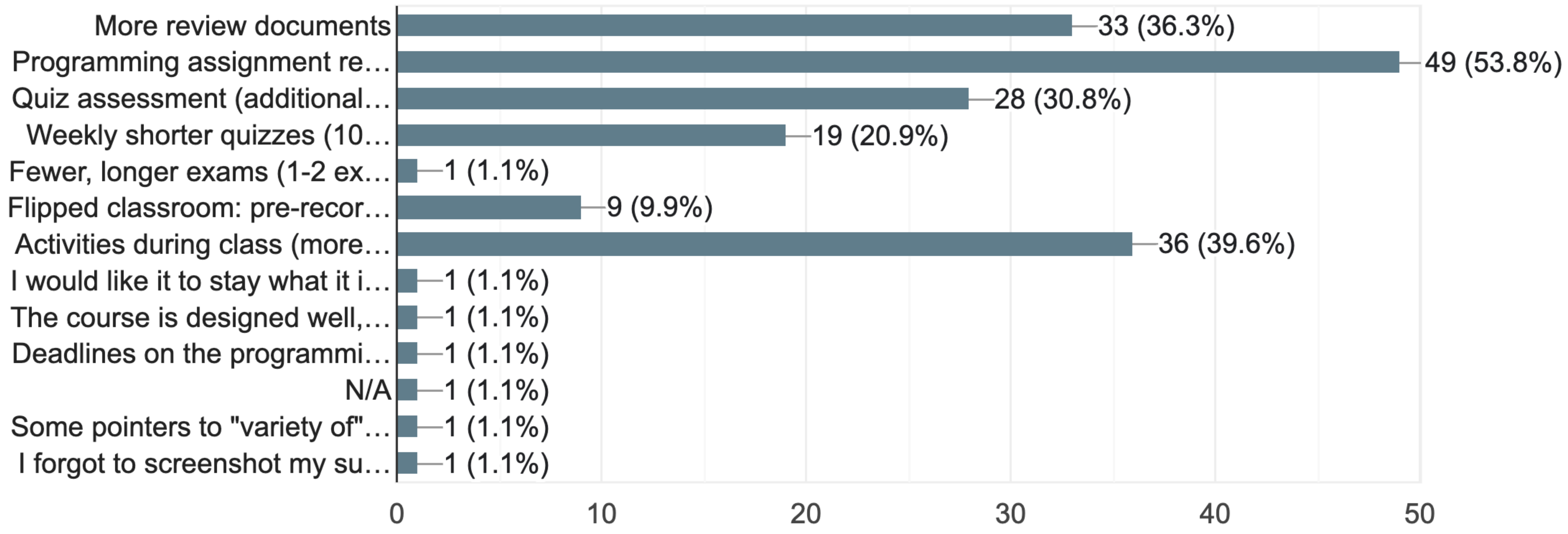
80 responses



Week 8

6. What changes could be made in the first weeks of the course to support learning? (Choose up to 2):

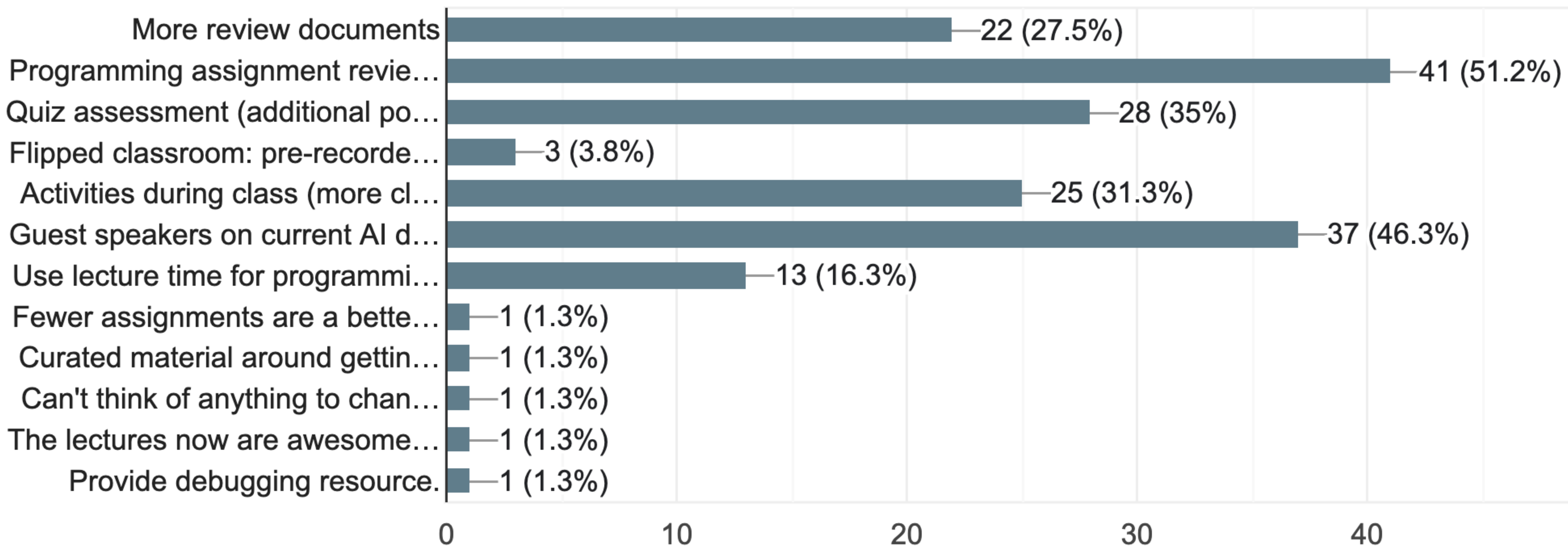
91 responses



Week 4

6. What changes could be made in the last few of the course to support learning? (Choose up to 2):

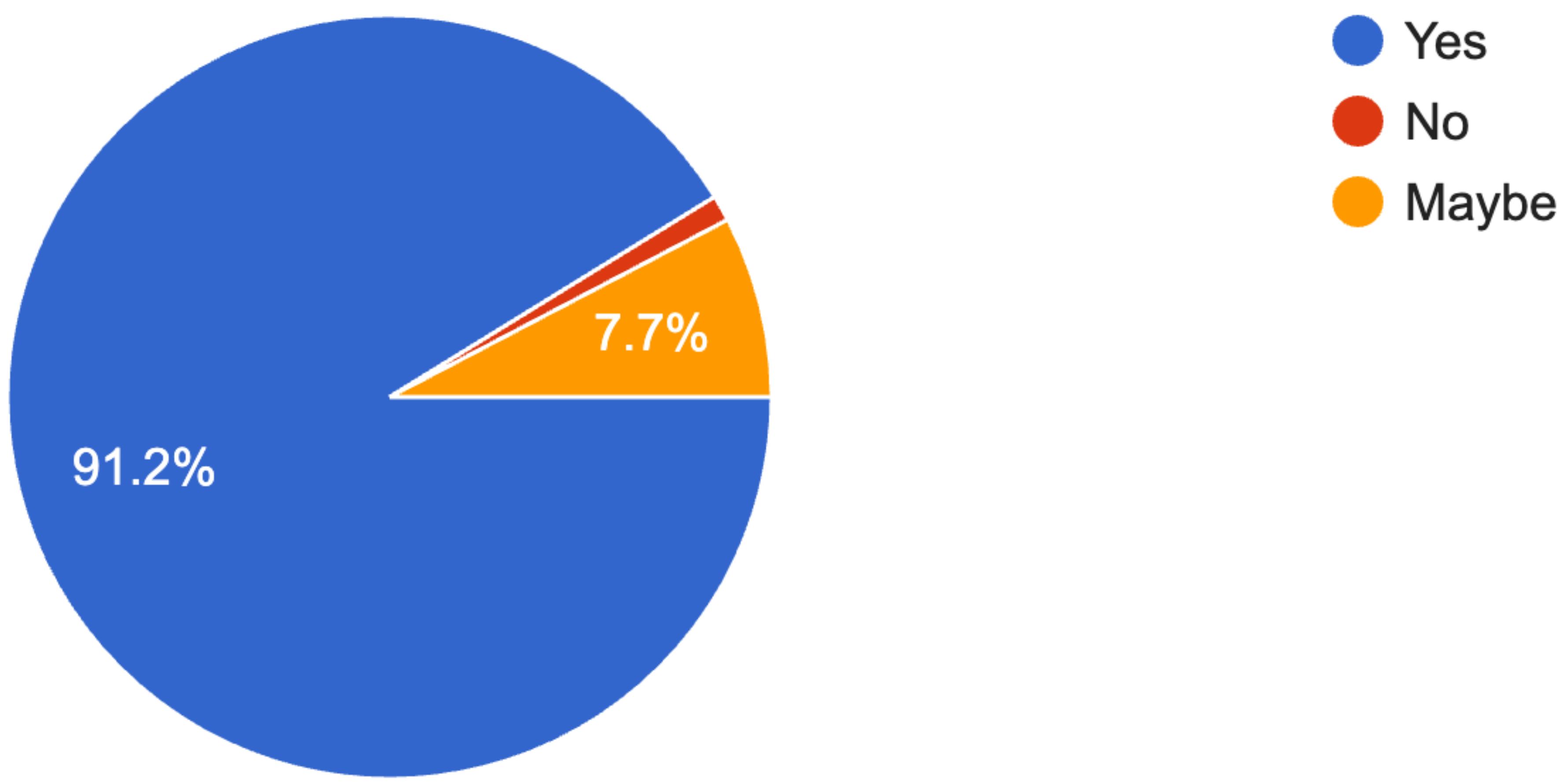
80 responses



Week 8

8. Do you feel that you have enough support and accessibility from the teaching team?

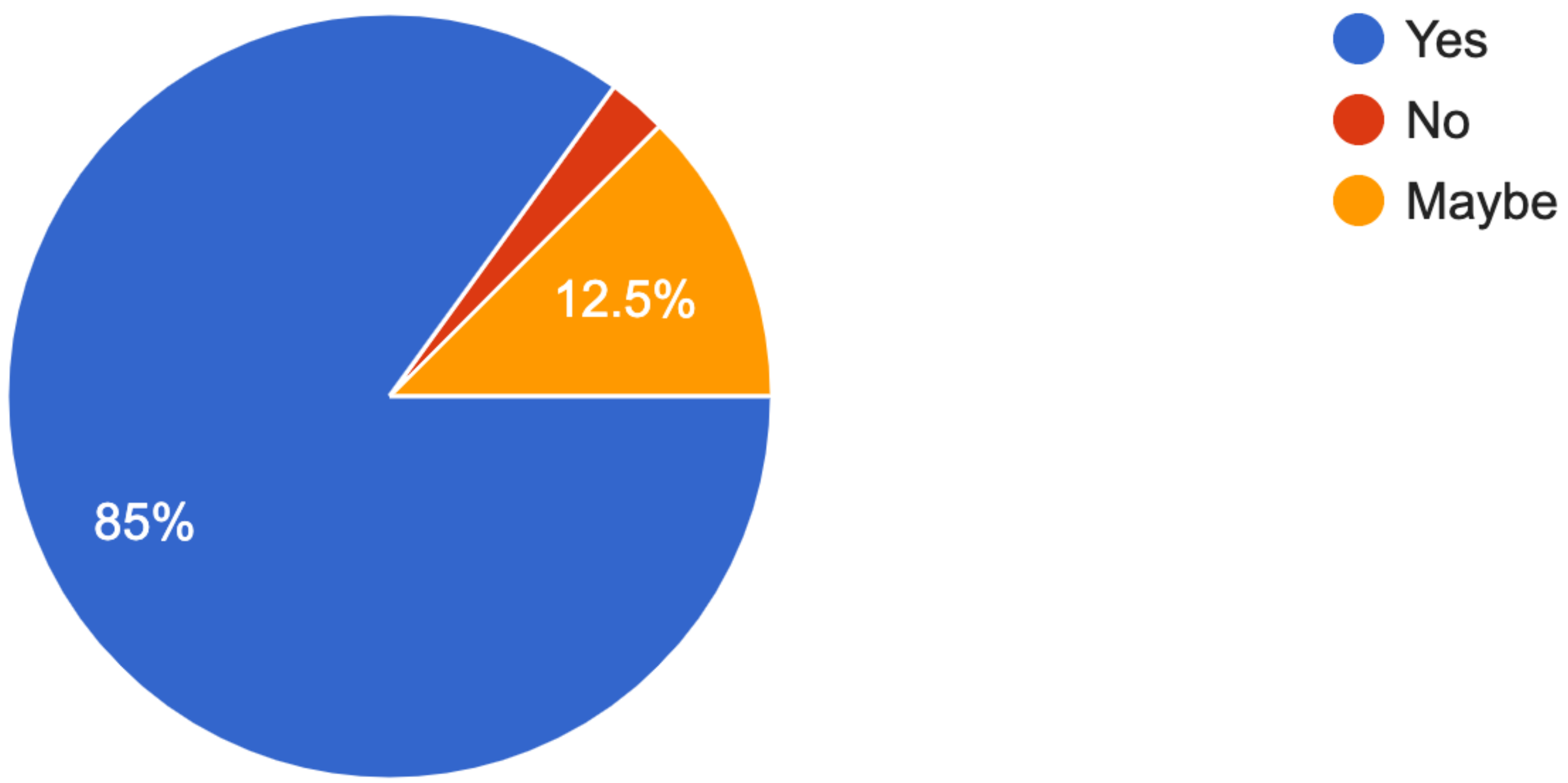
91 responses



Week 4

8. Do you feel that you have enough support and accessibility from the teaching team?

80 responses



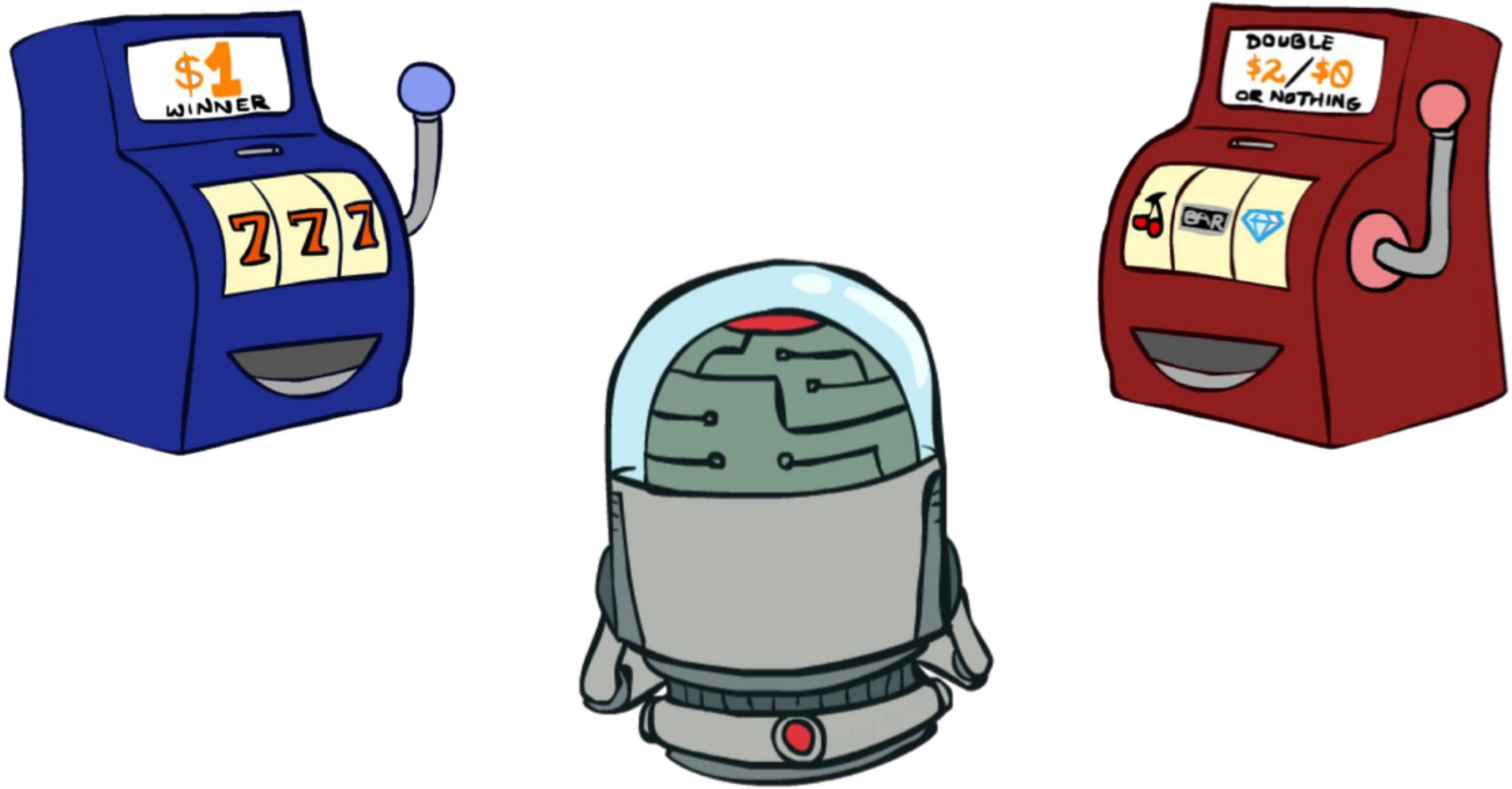
Week 8

Changes

- Go over assignments in class (tradeoff with advanced material).
- Concerns about workload.
- Post answers to the programming assignments.
- Concerns about regrades
  - Please come to me/TAs.
  - Unfortunately we don't have an auto grader, so we do make mistakes.
- You can make appointments with me (if you have questions, cannot attend office hours) by calendly (on the syllabus).
- We hear you about ambiguity on the assignment. Assignment 4/5 are very clear.
- I'll post some review materials/video before the next quiz.

Reinforcement Learning

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

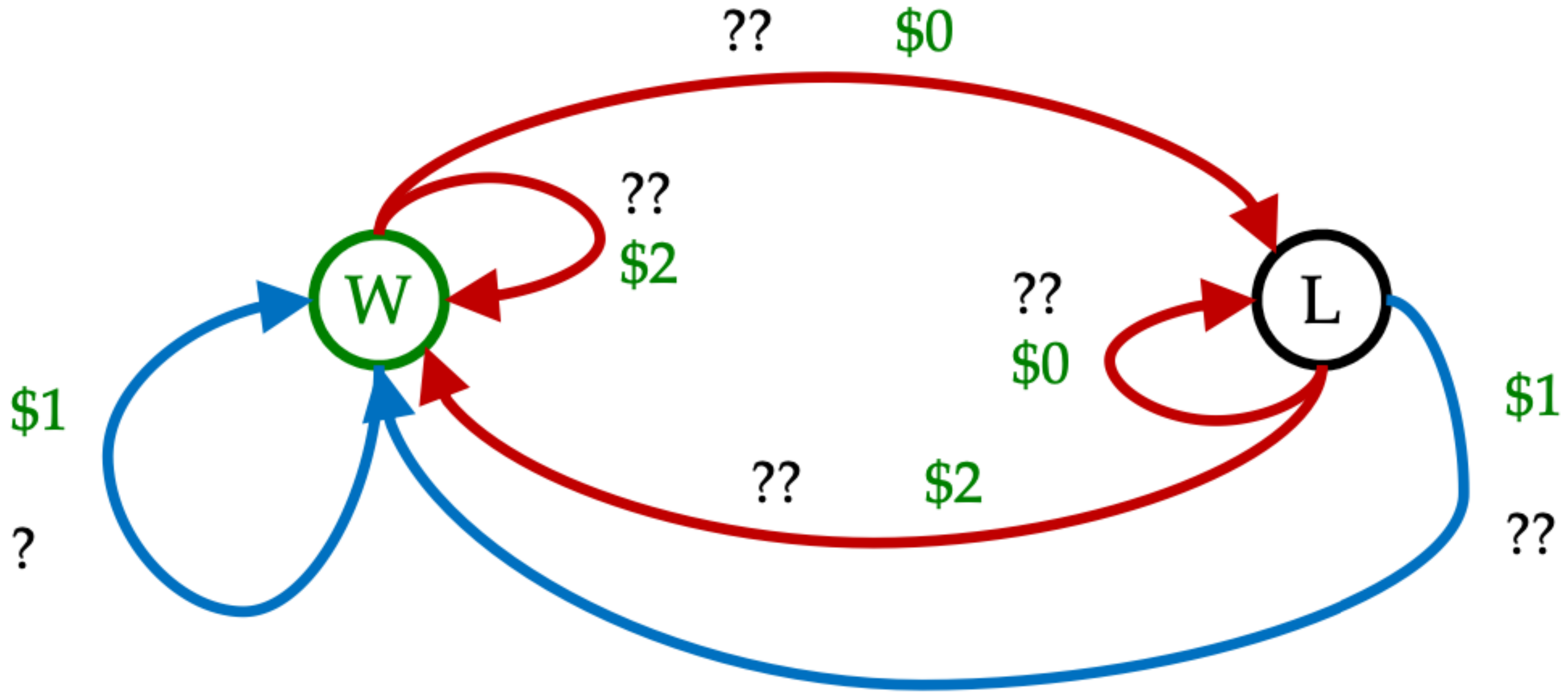
Let's Play!

$2 $2 $0 $2 $2

$2 $2 $0 $0 $0

Online Planning

- Rules changed! Win chance is different.



Let's Play!

$1 $1 $1

$0 $0 $0 $2

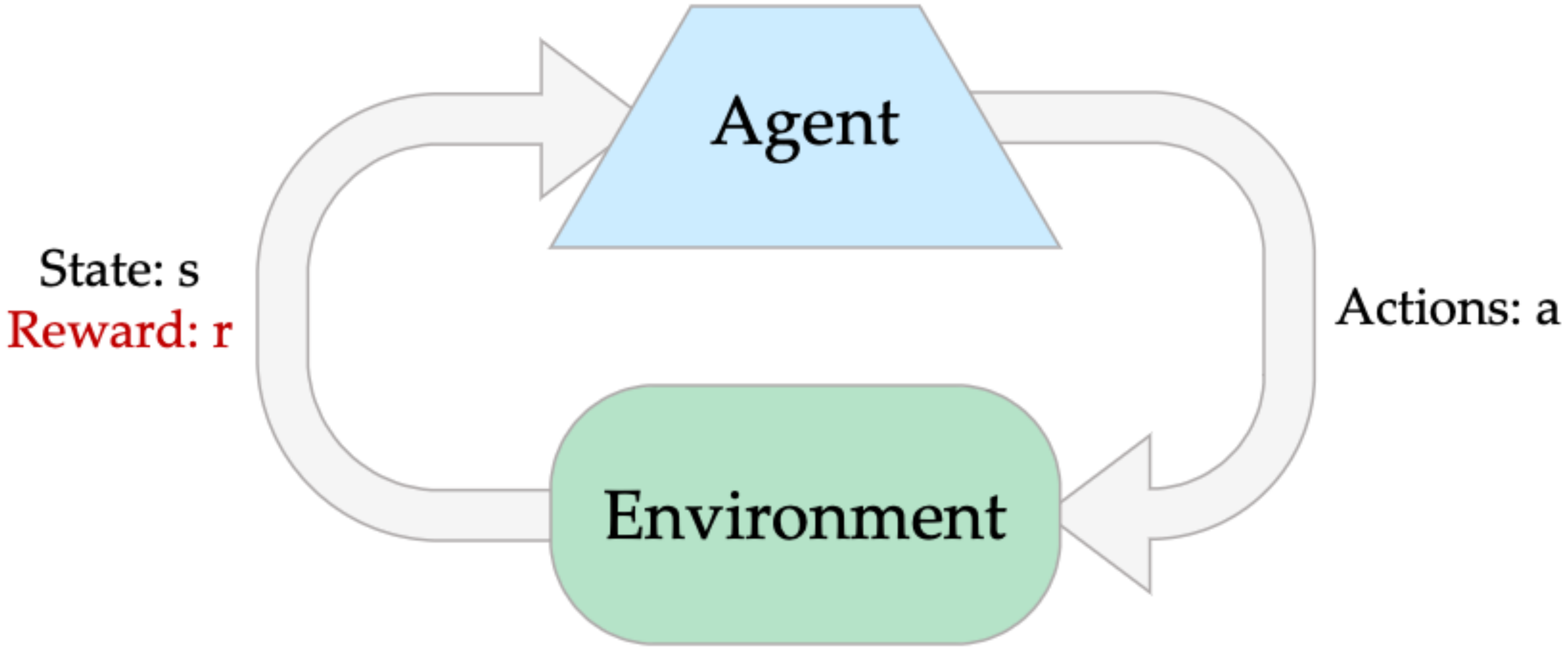
What Just Happened?

- 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

- Reinforcement learning:
  - Still assume an 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. don't know which states are good or what the actions do
    - Must actually try actions and states out to learn

Reinforcement Learning



- Basic idea:
  - Receive feedback in the form of rewards
  - Agent's utility is defined by the reward function
  - Must (learn to) act so as to maximize expected rewards
  - All learning is based on observed samples of outcomes!

Example: Pancake Flipping Robot

https://www.youtube.com/watch?v=W_gxLKSsSIE