
CS 232: Artificial Intelligence

Fall 2026

Prof. Carolyn Anderson
Wellesley College

Reminders

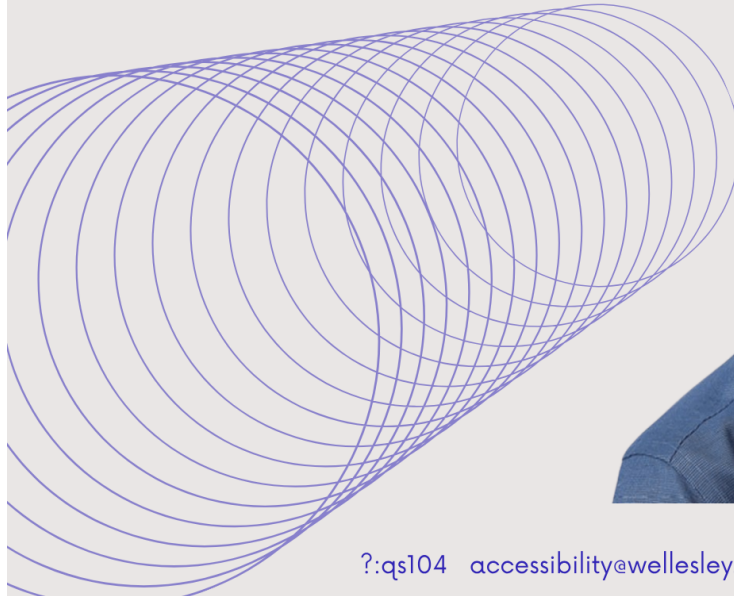
- ♦ I have help hours Friday from 3-4 in W423
- ♦ Midterm 1 is next Tuesday (10/6).
- ♦ It covers all concepts from our Search unit, but not Classification, which we start on Friday.
- ♦ No homework assignment this week.

THURSDAY, OCTOBER 8
12:45PM TO 2PM
H105



<https://bit.ly/3V2EJLd>

CS Colloquium:
**Tech Ethics and
College Chaplaincy:
Lessons from MIT
Radius**



Recap

Stochastic Transition Model

So far we have considered search problems where the outcomes of our actions was known.

Our transition function took our current state and an action, and told us which state we would end up in next:

$$T(s,a) \rightarrow s'$$

But what if we don't know what will happen next?

Transition function is stochastic:

$$T(s,a,s') \rightarrow p(s' | s,a)$$

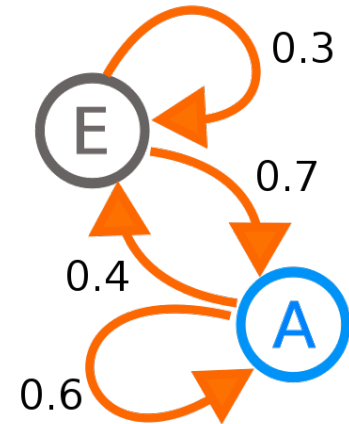
Markov Decision Processes

- ◆ Known state space
- ◆ Known transition function
- ◆ Transition function is stochastic:

$$T(s,a,s') \rightarrow p(s' | s,a)$$

- ◆ Reward function that tells us the value of being in each state

Markovian transitions depend only on the current state



Assumptions About Transitions

When the transition function is known and deterministic:

BFS, DFS, UCS, A*, minimax

When the transition function is known but stochastic:

Expectimax, Markov Decision Processes

When the transition function is unknown:

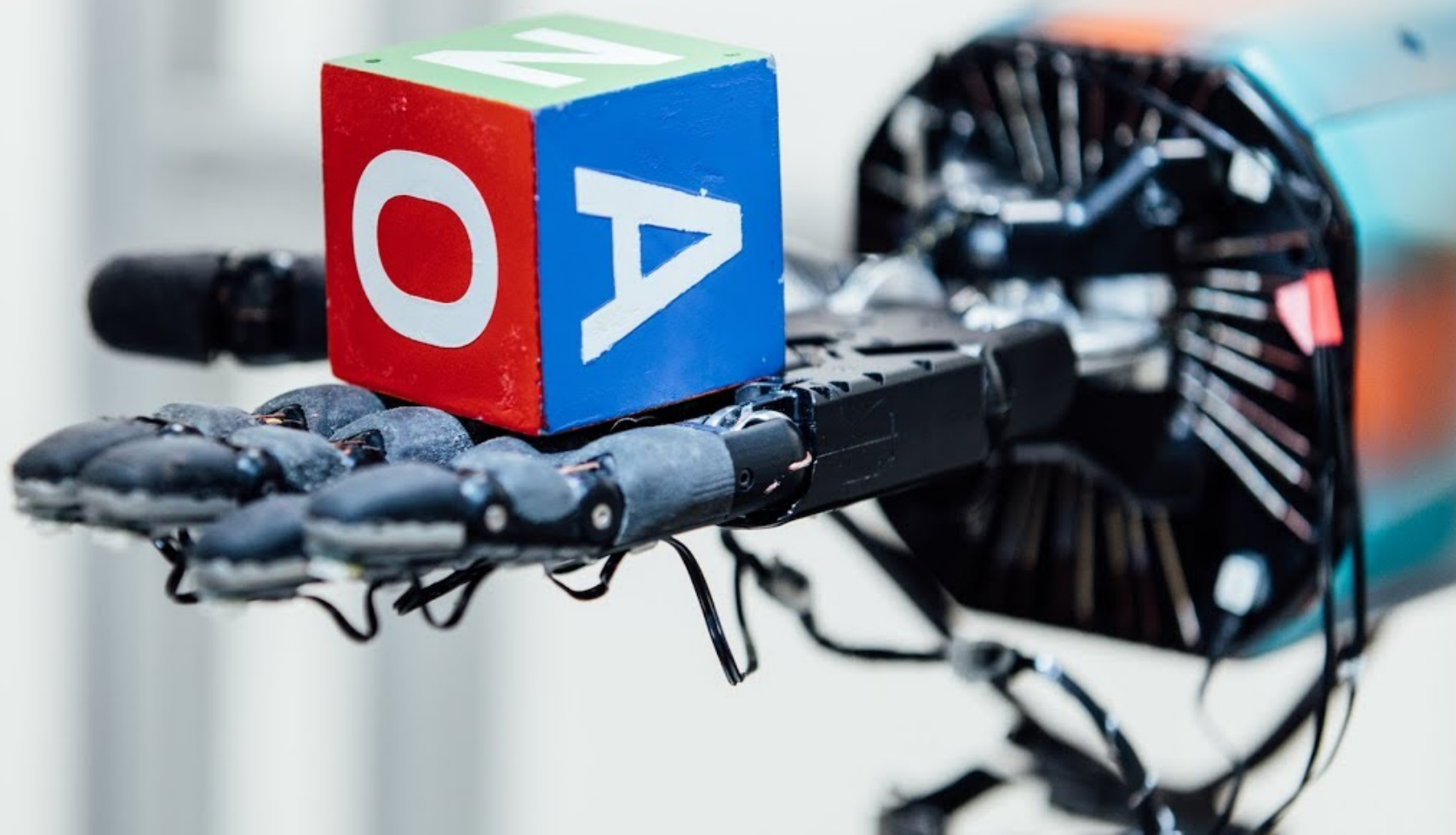
Reinforcement Learning

Reinforcement Learning

Transitions Are Unknown

In Markov Decision Processes, the outcomes were uncertain, but not completely unknown.

In Reinforcement Learning, we have to **discover** the transition function and the rewards associated with each state.



Reinforcement Learning

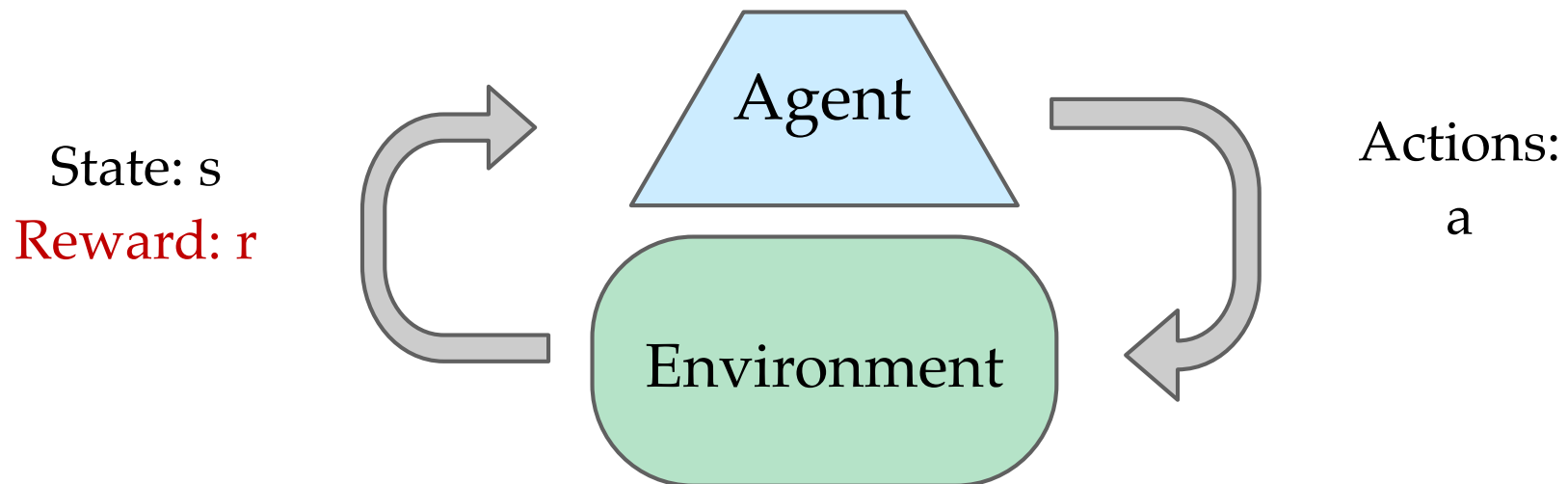
✦ We still assume the following components:

- States: S $\rightarrow$ Observed
- Actions: A
- Transition Function $\rightarrow$ Hidden
- Reward Function

Reinforcement Learning

Basic idea:

- ♦ Agent receives feedback in the form of rewards
- ♦ Agent tries to learn to act in order to maximize expected rewards
- ♦ Agent learns from samples of observed outcomes (AKA: try something out and see what happens!)



RL Components

- ♦ Policy selection: how do we pick actions?
- ♦ Policy evaluation: how do we learn from the results of our actions?

RL Demo



Policy Evaluation

Direct Evaluation

In Direct Evaluation, we repeatedly follow the same policy and observe its rewards.

Reminder: Policy is a function $\pi(s) \rightarrow a$

Direct Evaluation:

- Act according to policy π
- In each state we visit:
 - write down the observed reward + the sum of discounted future rewards
- Discounting: apply a fractional weight to future rewards so that we weight them less than immediate rewards
- Average samples

Direct Evaluation

Main Idea:

- ♦ Act according to **policy** π
- ♦ In each state, write down the **sum of discounted rewards**: the value of a state s is equal to the expected immediate reward at s plus the **expected future reward** from the successor state s'
- ♦ **Average multiple samples**

γ = discounting rate

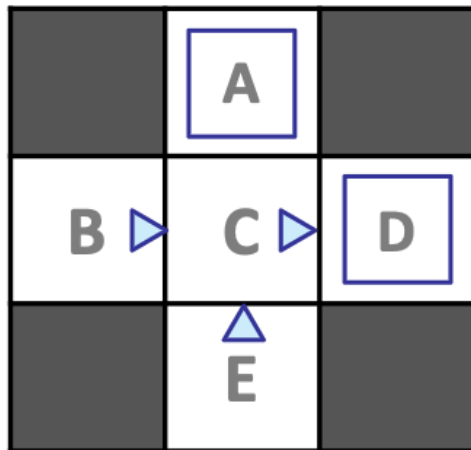
Sample of $V(s)$: $sample = R(s, \pi(s), s') + \gamma V^\pi(s')$

$R(s, \pi(s), s')$: reward from taking action $\pi(s)$ from s to s'

$V^\pi(s') = \text{future rewards} / \text{value of being in } s'$

Example: Direct Evaluation

Input Policy π



Assume: $\gamma = 1$

Observed Episodes (Training)

Episode 1

B, east, C, -1
C, east, D, -1
D, exit, x, +10

Episode 2

B, east, C, -1
C, east, D, -1
D, exit, x, +10

Episode 3

E, north, C, -1
C, east, D, -1
D, exit, x, +10

Episode 4

E, north, C, -1
C, east, A, -1
A, exit, x, -10

Output Values

	A ⁻¹⁰	
B ⁸	C ⁴	D ¹⁰
	E ⁻²	

Example: Direct Evaluation

Episode 1

B, east, C, -1
C, east, D, -1
D, exit, x, +10

Episode 2

B, east, C, -1
C, east, D, -1
D, exit, x, +10

Episode 3

E, north, C, -1
C, east, D, -1
D, exit, x, +10

Episode 4

E, north, C, -1
C, east, A, -1
A, exit, x, -10

↑

$$V(B) = -1 + 1(-1 + 10) = 8$$

$$V(E) = -1 + 1(-1 + 10) = 8$$

$$V(E) = -1 + 1(-1 + -10) = -12$$

	A	
B 8	C 9	D 10
	E	

	A	
B 8	C 9	D 10
	E	

	A	
B	C 9	D 10
	E 8	

	A -10	
B	C -11	D
	E -12	

$$V(C) = -1 + 1(10) = 9$$

$$V(C) = -1 + 1(10) = 9$$

$$V(C) = -1 + 1(-10) = -11$$

discounting

For simplicity, assume a weight decay of $\gamma = 1$ (no decay)

Weaknesses of Direct Evaluation

Direct Evaluation works.

But, we had to learn each state separately.

That isn't very efficient!

Output Values

	-10 A	
+8 B	+4 C	+10 D
	-2 E	

*If B and E both go to C
under this policy, how can
their values be different?*

Temporal Difference Learning

**Andrew Barto and Richard Sutton win
2024 Turing Award**



Andrew Barto and Richard Sutton. Image credit: Association for Computing Machinery.

Temporal Difference Learning

Main Idea: learn from every experience

TD Learning

- ♦ Update $V(s)$ every time we make a transition $T(s,a,s',r)$
- ♦ Most likely outcomes (s') will contribute more updates
- ♦ Use a *learning rate* to control how quickly we update values

α : learning rate

$\gamma V^\pi(s')$ = old info

Sample of $V(s)$:

$$sample = R(s, \pi(s), s') + \gamma V^\pi(s')$$

= new info

Update to $V(s)$:

$$V^\pi(s) \leftarrow (1 - \alpha)V^\pi(s) + (\alpha)sample$$

Same update:

$$V^\pi(s) \leftarrow V^\pi(s) + \alpha(sample - V^\pi(s))$$

TD: A Moving Average

In TD Learning, we average our observations in a way that weights more recent samples highly.

Over time, we forget our initial estimates (which probably weren't very good!)

$$\bar{x}_n = \frac{x_n + (1 - \alpha) \cdot x_{n-1} + (1 - \alpha)^2 \cdot x_{n-2} + \dots}{1 + (1 - \alpha) + (1 - \alpha)^2 + \dots}$$

We decrease the **learning rate α** over time.

Example: Temporal Difference Learning

States

	A	
B	C	D
	E	

Assume:

$$\gamma = 1,$$

$$\alpha = 1/2$$

Observed Transitions

B, east, C, -2

C, east, D, -2

	0	
0	0	8
	0	

$$V^\pi(s) \leftarrow (1 - \alpha)V^\pi(s) + \alpha \left[R(s, \pi(s), s') + \gamma V^\pi(s') \right]$$

Example: Temporal Difference Learning

States

	A	
B	C	D
	E	

Assume:

$$\gamma = 1,$$

$$\alpha = 1/2$$

Observed Transition:

B, east, C, -2

	0	
-1	<u>0</u>	8
	0	

$$V^\pi(s) \leftarrow (1 - \alpha)V^\pi(s) + \alpha [R(s, \pi(s), s') + \gamma V^\pi(s')]$$

$$V(B) \stackrel{0.5}{=} 0 + 0.5 \cdot (-2 + 1 \cdot \underline{0})$$

$$= -1$$

Example: Temporal Difference Learning

States

	A	
B	C	D
	E	

Assume:

$$\gamma = 1,$$

$$\alpha = 1/2$$

Observed Transition:

C, east, D, -2

	0	
-1	0	<u>8</u>
	0	

$$V^\pi(s) \leftarrow (1 - \alpha)V^\pi(s) + \alpha [R(s, \pi(s), s') + \gamma V^\pi(s')]$$

$$\begin{aligned} V(C) &= 0 + \underline{0.5} [-2 + 1 \cdot \underline{8}] \\ &= 3 \end{aligned}$$

Example: Temporal Difference Learning

States

	A	
B	C	D
	E	

Assume:

$$\gamma = 1,$$

$$\alpha = 1/2$$

Observed Transitions

B, east, C, -2

C, east, D, -2

	0	
0	0	8
	0	

initial estimator
of $V(s)$

	0	
-1	0	8
	0	

updated $V(B)$

	0	
-1	3	8
	0	

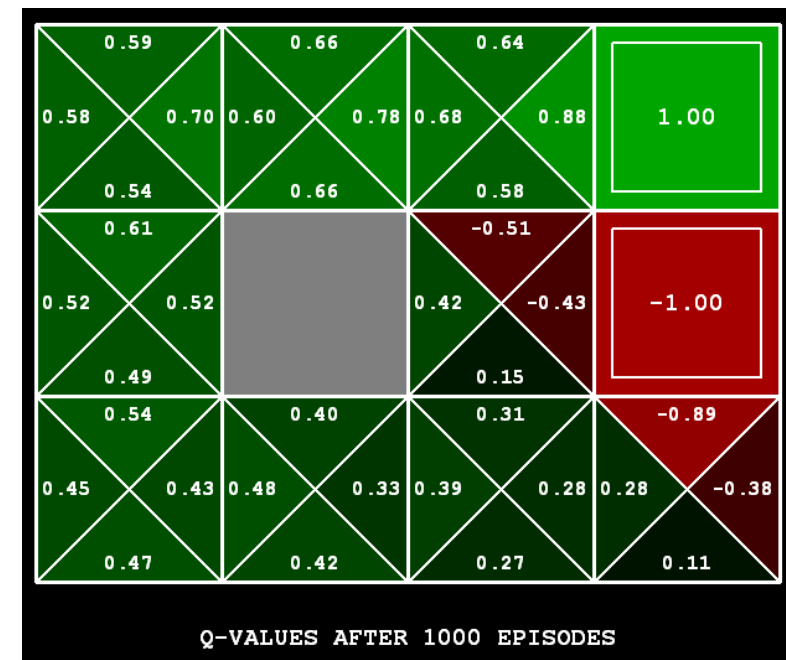
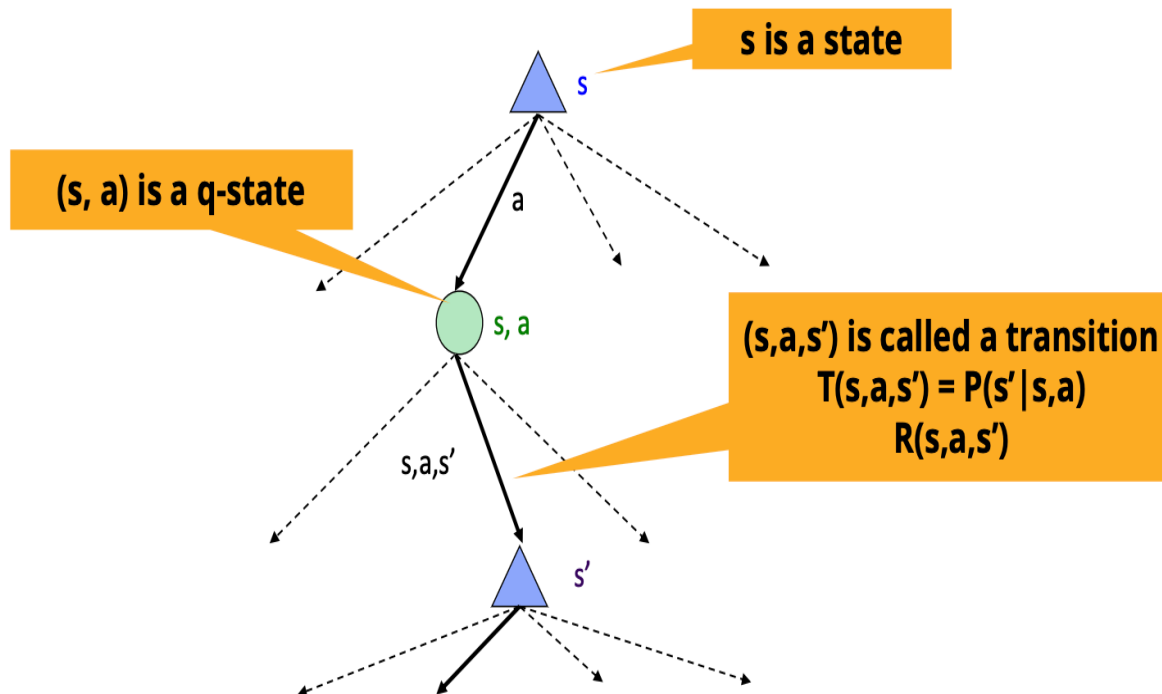
updated $V(C)$

$$V^\pi(s) \leftarrow (1 - \alpha)V^\pi(s) + \alpha [R(s, \pi(s), s') + \gamma V^\pi(s')]$$

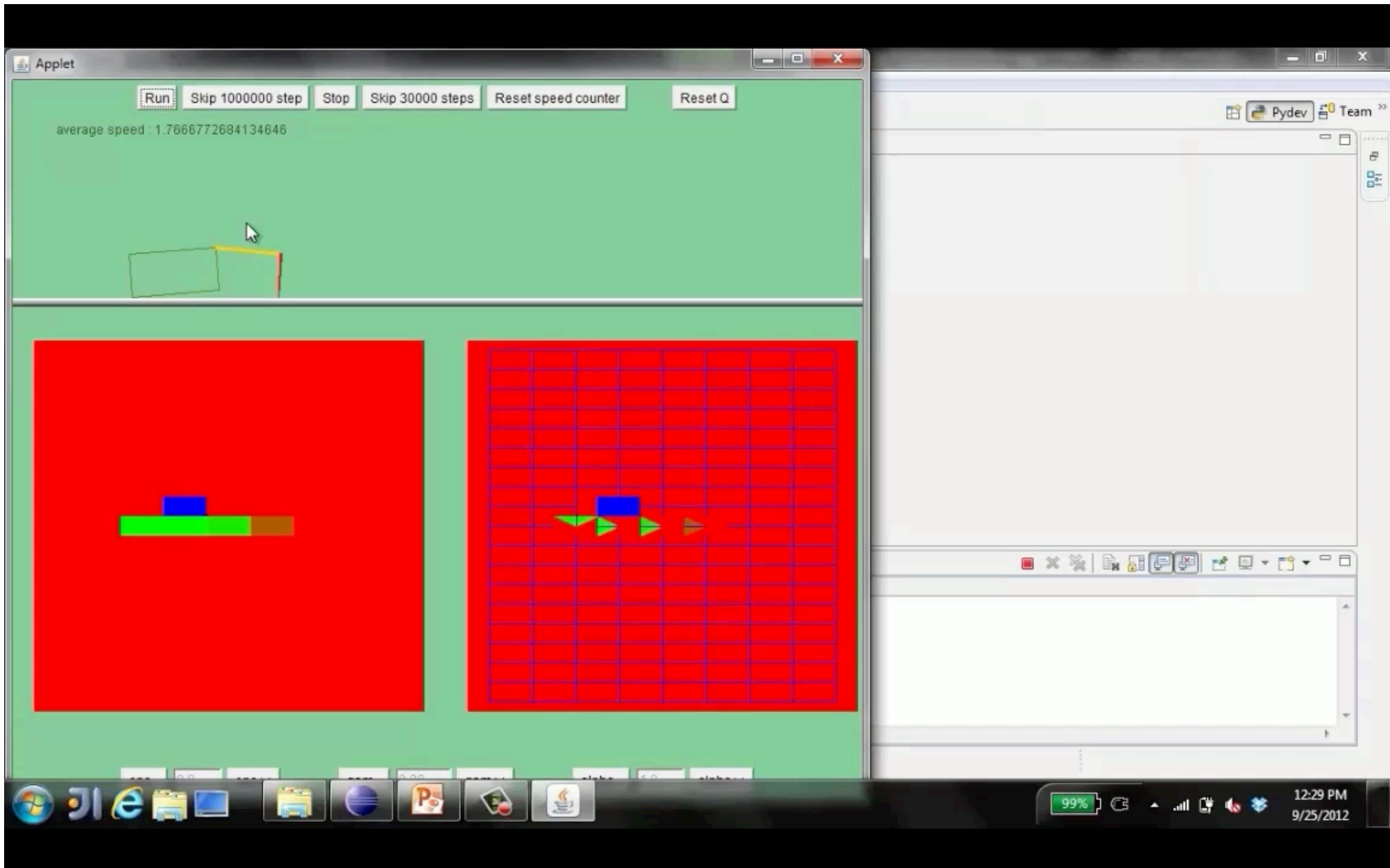
Q-Learning

In practice, we want to know the value of state-action pairs, not just states.

This version of TD Learning is called **Q-Learning**, because the value of a state-action pair is called a **q-value**.



Demo of Q-Learning



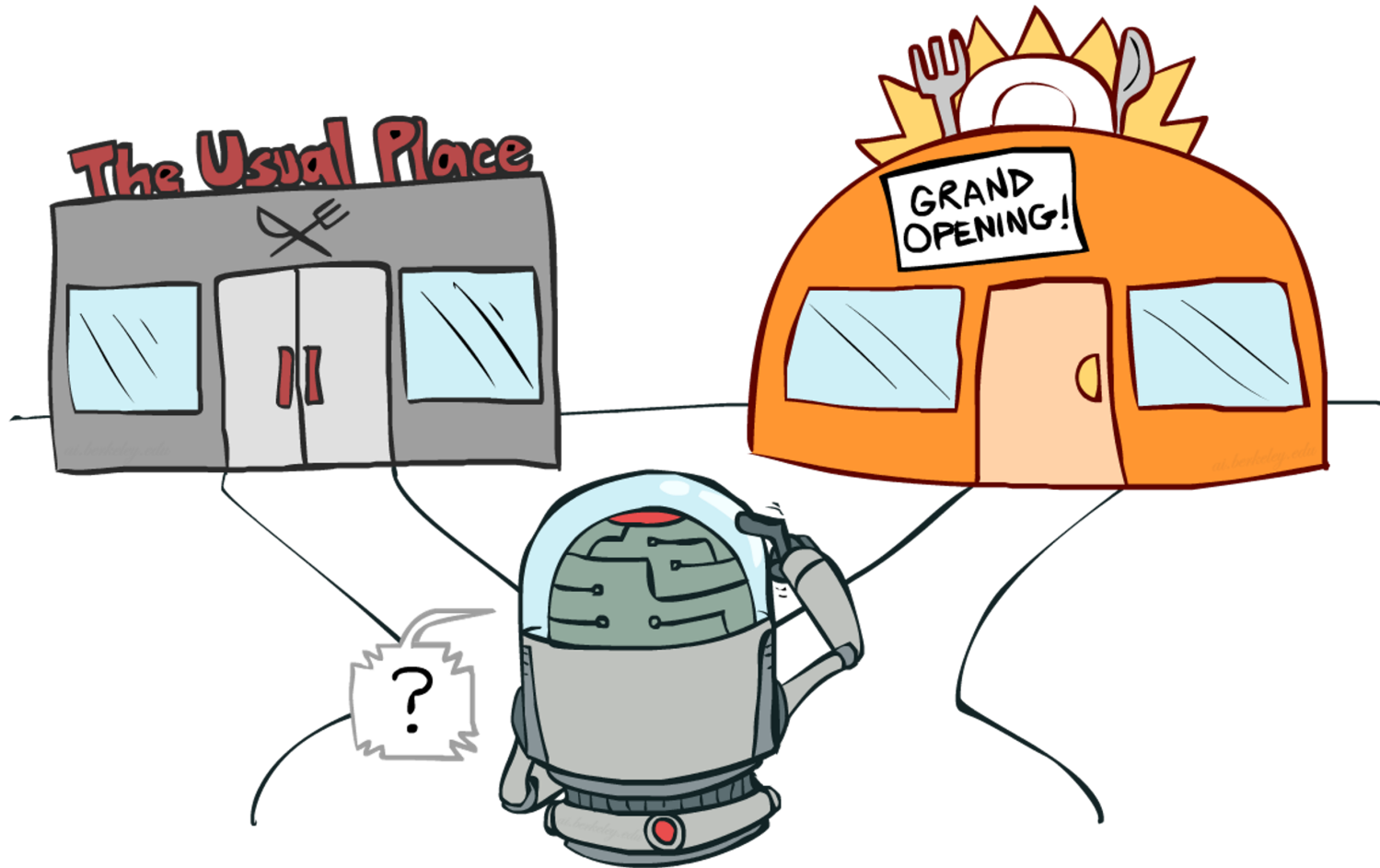
Policy Selection

Policy Selection

RL policy selection must balance two things:

- ♦ **Exploitation**: choose the action with highest reward (according to current information)
- ♦ **Exploration**: choose a new action to discover new information more quickly.

Policy Selection



Exploration Versus Exploitation

Policy Selection

RL policy selection must balance two things:

- ♦ **Exploitation**: choose the action with highest reward (according to current information)
- ♦ **Exploration**: choose a new action to discover new information more quickly.
- ♦ Different strategies for exploring:
 - Choose the action that has been tried least often in the current state.
 - Choose an action that leads to states that have been unexplored.
 - Choose a random action.

Policy Selection

Passive Reinforcement Learning

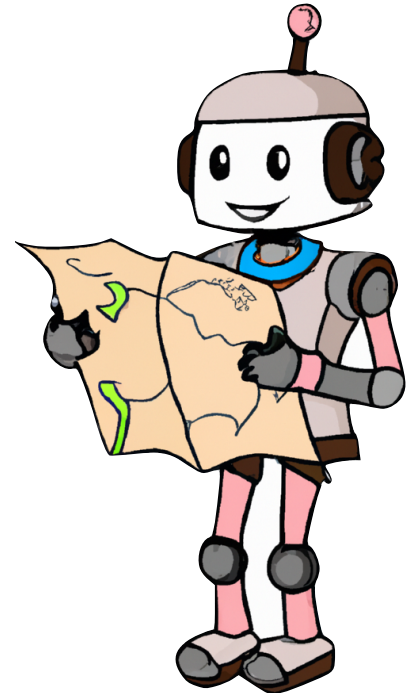
- ♦ Take a fixed policy and follow it

Random Exploration

- ♦ Pick actions randomly

Balancing Exploration and Exploitation

- ♦ Pick actions randomly initially, but gradually switch to mostly doing actions that you have already found to be valuable

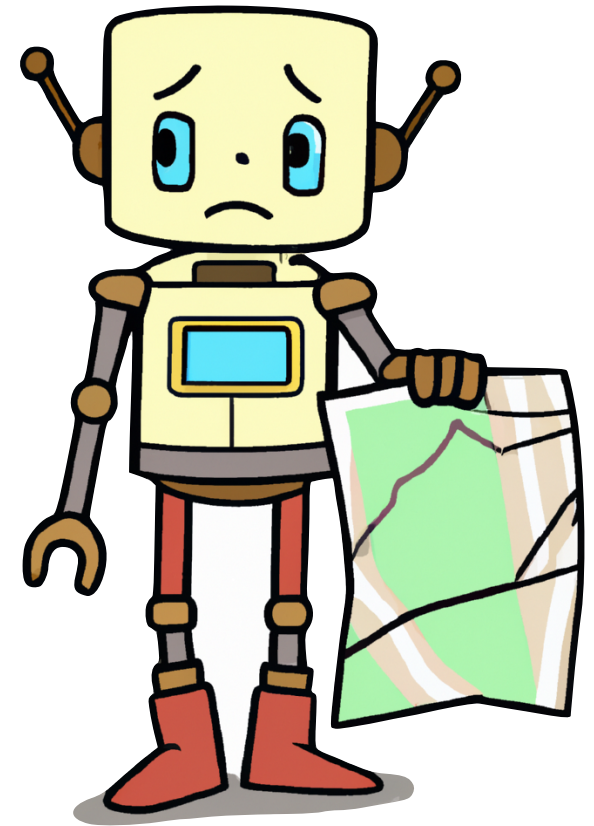


Regret

Even if our Reinforcement Learning agent learns an optimal policy eventually, it will still take sub-par actions along the way.

Regret is the total cost of all of those mistakes: the differences between optimal expected rewards and the agent's actual rewards.

The best RL agent is one that **minimizes regret**: learns optimally!

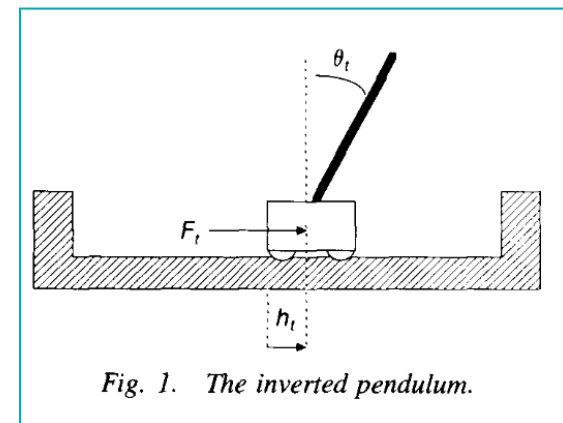


RL History: Games

- ◆ RL was first proposed in the context of game-playing. Arthur Samuel developed an RL-based checkers player in 1959.
- ◆ In 1992, TD learning was used to train TD Gammon, a backgammon agent that reached world-champion level.
- ◆ In 2016, DeepMind's AlphaGo system defeated Go champion Lee Sedol. It was trained on human expert games then refined by reinforcement learning via self-play.

RL History

- ◆ RL was first proposed in the context of game-playing. Arthur Samuel developed an RL-based checkers player in 1959.
- ◆ In 1968, Michie and Chambers had success applying RL to simple robot navigation.
- ◆ Most foundational work on RL is from the 1990s, but applications took off in the 2010s once deep neural networks could be integrated into evaluation.
- ◆ Since then, RL has become one of the most commonly used techniques for training robotics
- ◆ In the last couple of years, RL has been adopted as a common post-training strategy for LLMs



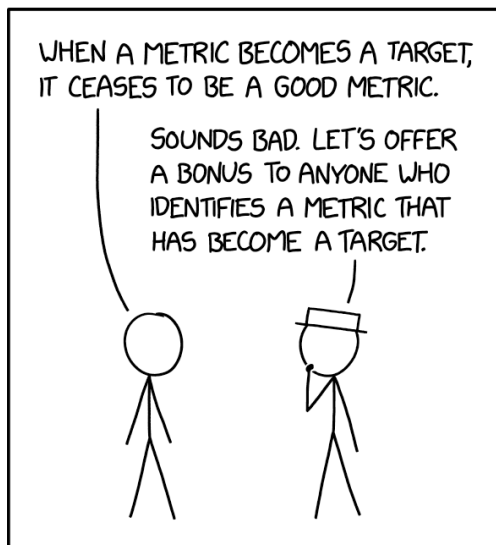
The Perils of Reward Maximization

RL models are trained to maximize their rewards and we know how to make them do this very effectively.

This makes it important to define the reward function *very carefully*.

GOODHART'S LAW

< < PREV RANDOM NEXT > >



xkcd

