Midterm Review

CMPUT 397
Fall 2020

Course Roadmap

Bandits (Ch2) ⟶ MDPs, returns, value functions (Ch3) ⟶ Dynamic programming (Ch4)

⟶ Monte Carlo learning (Ch5) ⟶ TD learning (Ch6) ⟶

Planning (Ch8)

Course Roadmap

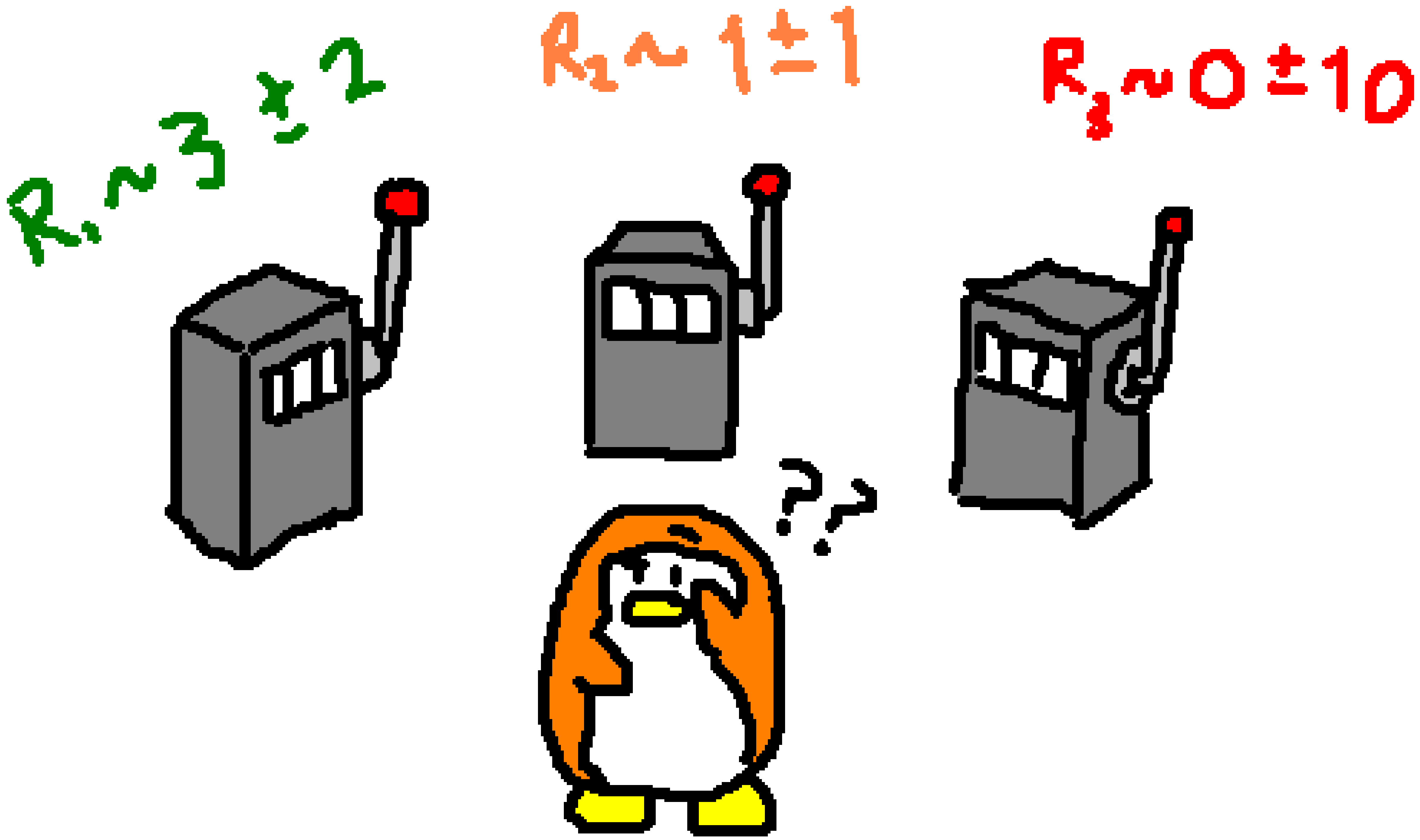
$\boxed{\text{Bandits (Ch2)}}$ ➡ MDPs, returns, value functions (Ch3) ➡ Dynamic programming (Ch4)

➡ Monte Carlo learning (Ch5) ➡ TD learning (Ch6) ➡

Planning (Ch8)

Bandits

- Simple decision making problem with 1 state



Bandits

- Know the exploration-exploitation tradeoff!
  - i.e. Why shouldn't you always be greedy? Why not constantly explore?
- Know about incremental averaging (and why we do it!)
  - NewEstimate←OldEstimate+StepSize[Target-OldEstimate]

Course Roadmap

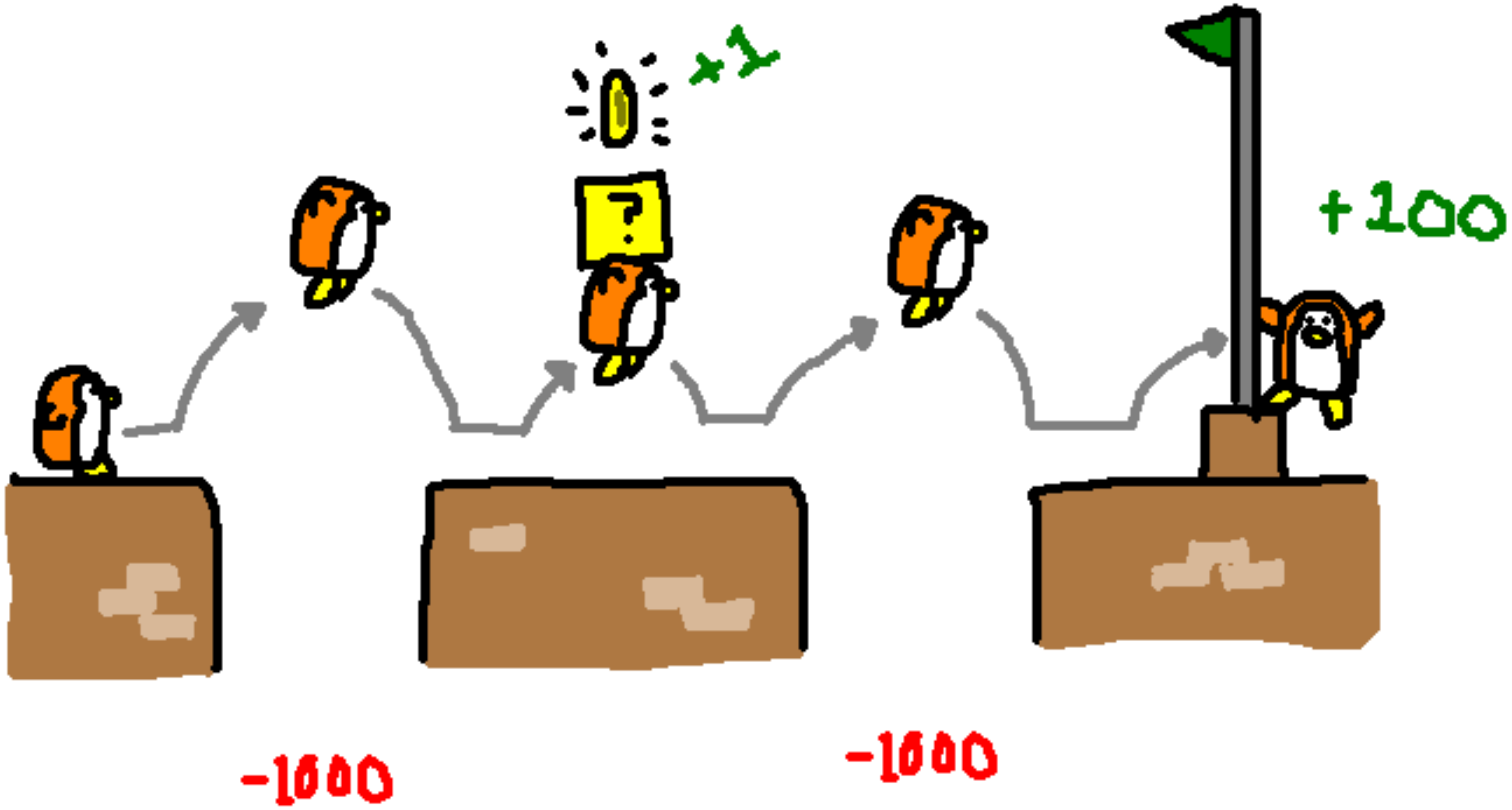
Bandits (Ch2) → MDPs, returns, value functions (Ch3) → Dynamic programming (Ch4)

→ Monte Carlo learning (Ch5) → TD learning (Ch6) →

Planning (Ch8)

MDPs, Returns, Value Functions

- Decision making problems with many states



MDPs, Returns, Value Functions

- Sequential decision making: must take many actions in a row to maximize reward
- Agent is concerned with **returns**:

$$\boxed{G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots + \gamma^{k-1} R_{t+k} + \ldots}$$

- Specifically, the agent estimates the **expected return**, which depends on the agent's **policy** and the **environment dynamics**

MDPs, Returns, Value Functions

- Value-based methods address this by learning to predict the **expected return**, i.e. learning **value functions**

- Value functions:

$$v_\pi(s) \doteq \mathbb{E}_\pi \left[G_t \mid S_t = s \right]$$ "How good is this state"

$$q_\pi(s, a) \doteq \mathbb{E}_\pi \left[G_t \mid S_t = s, A_t = a \right]$$ "How good is taking **this action** in this state"

MDPs, Returns, Value Functions

- Bellman Equations: write the value of a state in terms of the value of another state
- i.e. for all states:

$$
\begin{aligned}
v_\pi(s) &\doteq \mathbb{E}_\pi\left[G_t \mid S_t = s\right] \\
&= \sum_a \pi(a \mid s) \sum_{s'} \sum_r p(s', r \mid s, a)\left[r + \gamma v_\pi(s')\right]
\end{aligned}
$$

MDPs, Returns, Value Functions

- **Policy improvement**
  - If you derive a **greedy** policy with respect to the *action-values* of another policy, the new policy will be at least as “good” as the previous one
- If the new policy did not change from the previous policy, the policy is greedy with respect to its **own** value function, and is an optimal policy π^*
- Optimal value functions denoted $v^*(s)$ and $q^*(s,a)$

Self-test: Connections to Course 2

- We only talked about the policy improvement result in Course 1, when we did DP
- How is policy improvement relevant for the sample-based methods in Course 2?
    - how is it relevant for Sarsa?
    - how is it relevant for Q-learning?

Course Roadmap

Bandits (Ch2) ➡ MDPs, returns, value functions (Ch3) ➡ $\boxed{\text{Dynamic programming (Ch4)}}$

➡ Monte Carlo learning (Ch5) ➡ TD learning (Ch6) ➡

Planning (Ch8)

Dynamic Programming: Iterative Policy Evaluation

- Computes an approximate value function $V(s) \approx v_\pi(s)$
- Sweeps across all states and actions, and evaluates the Bellman equation using the current estimates in the value function

$$v_{k+1}(s) \leftarrow \sum_{a} \pi(a \mid s) \sum_{s'} \sum_{r} p(s', r \mid s, a)\left[r + \gamma v_k(s')\right]$$

Dynamic Programming

- Introduces the idea of **bootstrapping** - basing the update to a state's value on the agent's current value estimates of successor states
- Requires knowledge of the **environment dynamics** p(s',r | s,a)
  - this is a model-based method since it assume access to the environment model p

Dynamic Programming: Value Iteration

- Uses Bellman equation to iterate towards v^* (and so towards π^*)

- $v_{k+1}(s) \leftarrow \max_a \sum_{s',r} p(s', r \mid s, a)[r + \gamma v_k(s')]$

- Contrast with policy evaluation update

- $v_{k+1}(s) \leftarrow \sum_a \pi(a \mid s) \sum_{s',r} p(s', r \mid s, a)[r + \gamma v_k(s')]$

- Policy iteration uses greedy policy, and fully evaluates the values for that policy (multiple sweeps to do IPE)

- Value Iteration greedifies, after only one sweep of evaluating with the current greedy policy

Course Roadmap

Bandits (Ch2) → MDPs, returns, value functions (Ch3) → Dynamic programming (Ch4)

→ $\boxed{\text{Monte Carlo learning (Ch5)}}$ → TD learning (Ch6) →

Planning (Ch8)

Monte Carlo Learning

- Policy Evaluation: estimates the value function $V(s) \approx v_{\pi}(s)$
- Sample *returns* from states by **following policy π**, then average those *returns* for each state

Monte Carlo Learning

- Doesn’t need a model of the environment
- We only used it in episodic problems: learning only occurs **after** each episode

Monte Carlo Self-test

- Given a We talked about two versions of Monte Carlo for prediction. The first uses a sample average (sample mean) of returns from a state s. The second uses the following incremental update rule, for a constant stepsize $\alpha > 0$
- $V(S_t) \leftarrow V(S_t) + \alpha(G_t - V(S_t))$
- What is the primary difference between the values learned with the sample average and those with the incremental update rule?

Course Roadmap

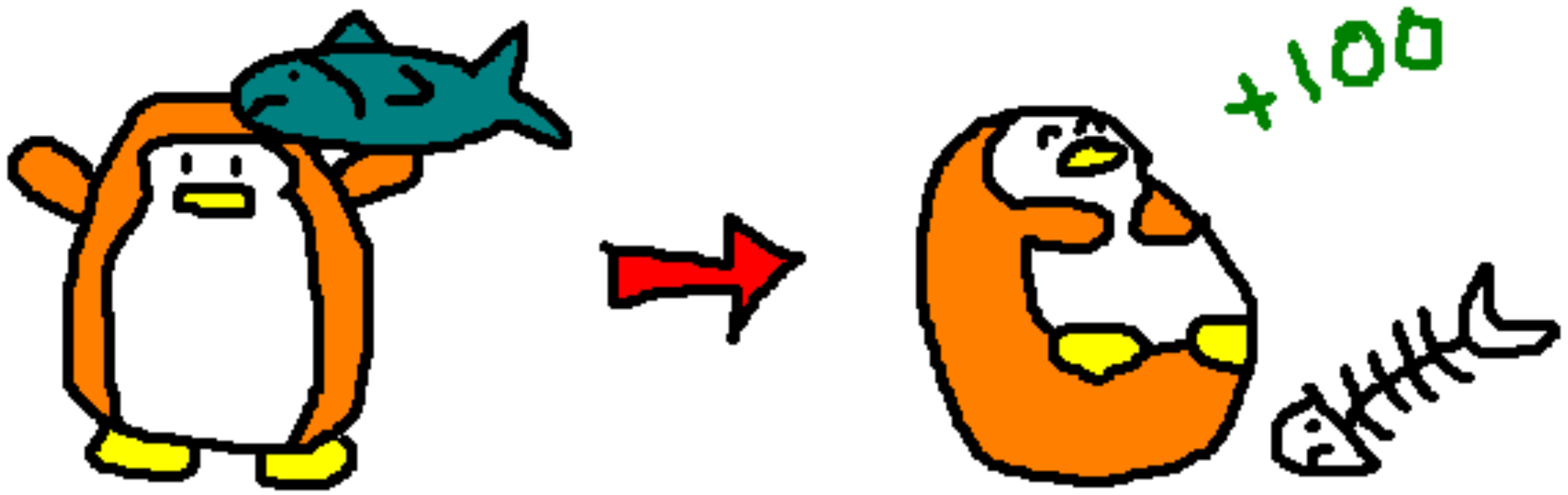
Bandits (Ch2) $\Rightarrow$ MDPs, returns, value functions (Ch3) $\Rightarrow$ Dynamic programming (Ch4)

$\Rightarrow$ Monte Carlo learning (Ch5) $\Rightarrow$ $\boxed{\text{TD learning (Ch6)}}$ $\Rightarrow$

Planning (Ch8)

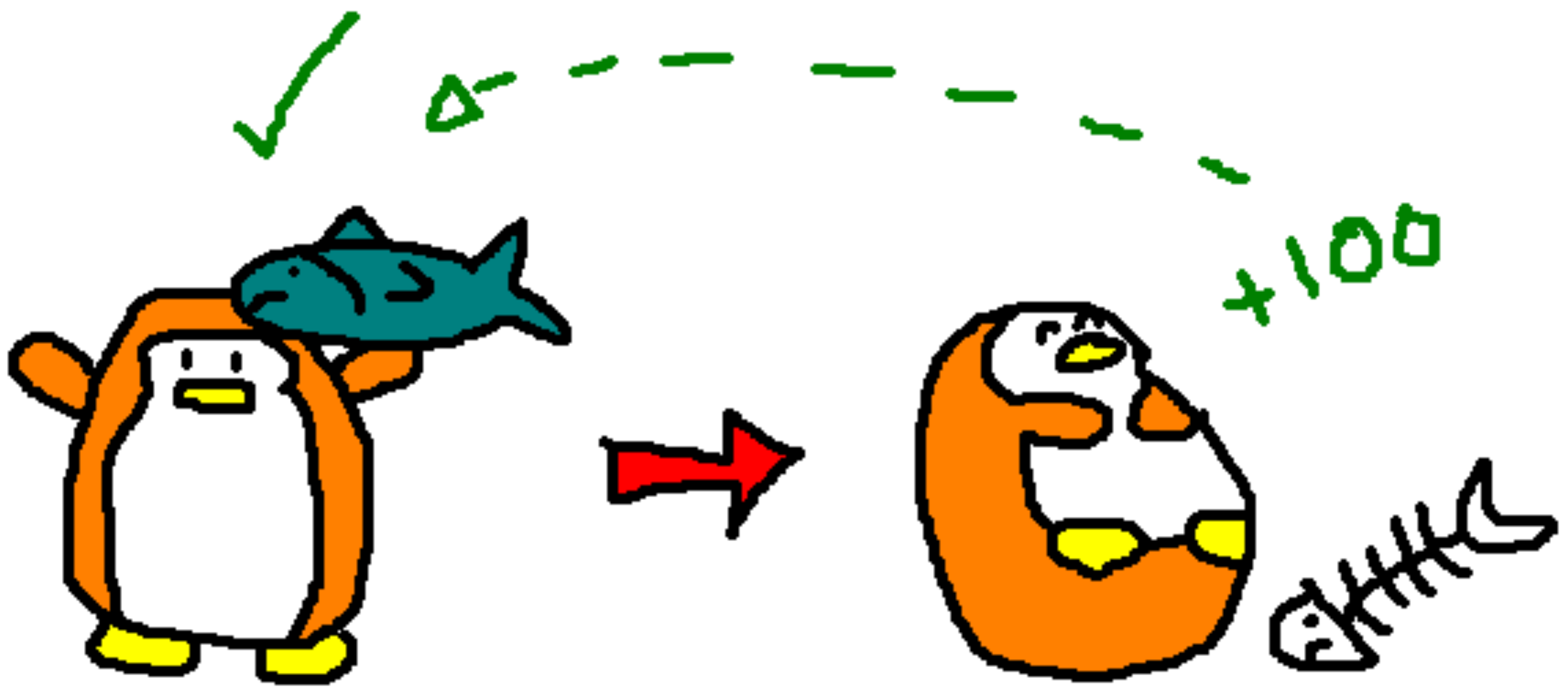
Temporal Difference Learning

Previous experience:



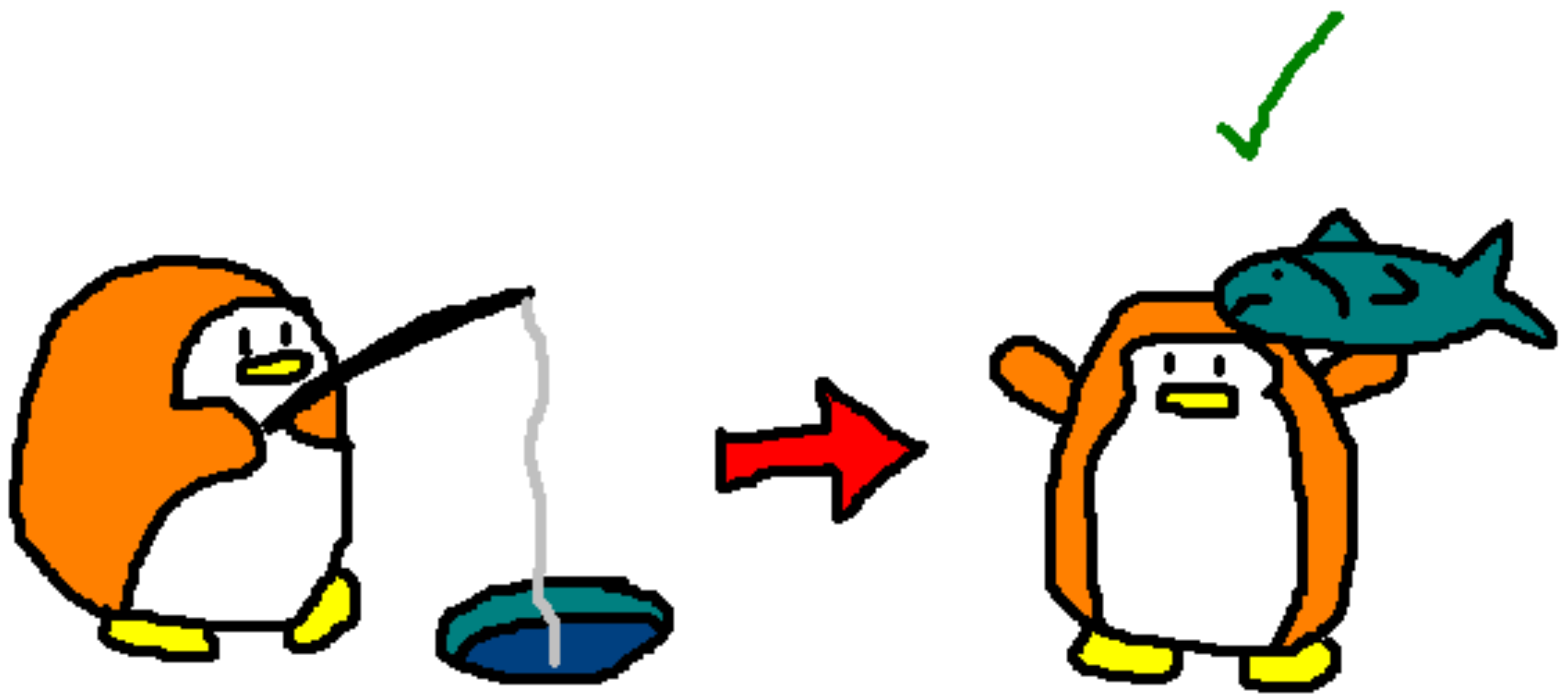
Temporal Difference Learning

Previous experience:



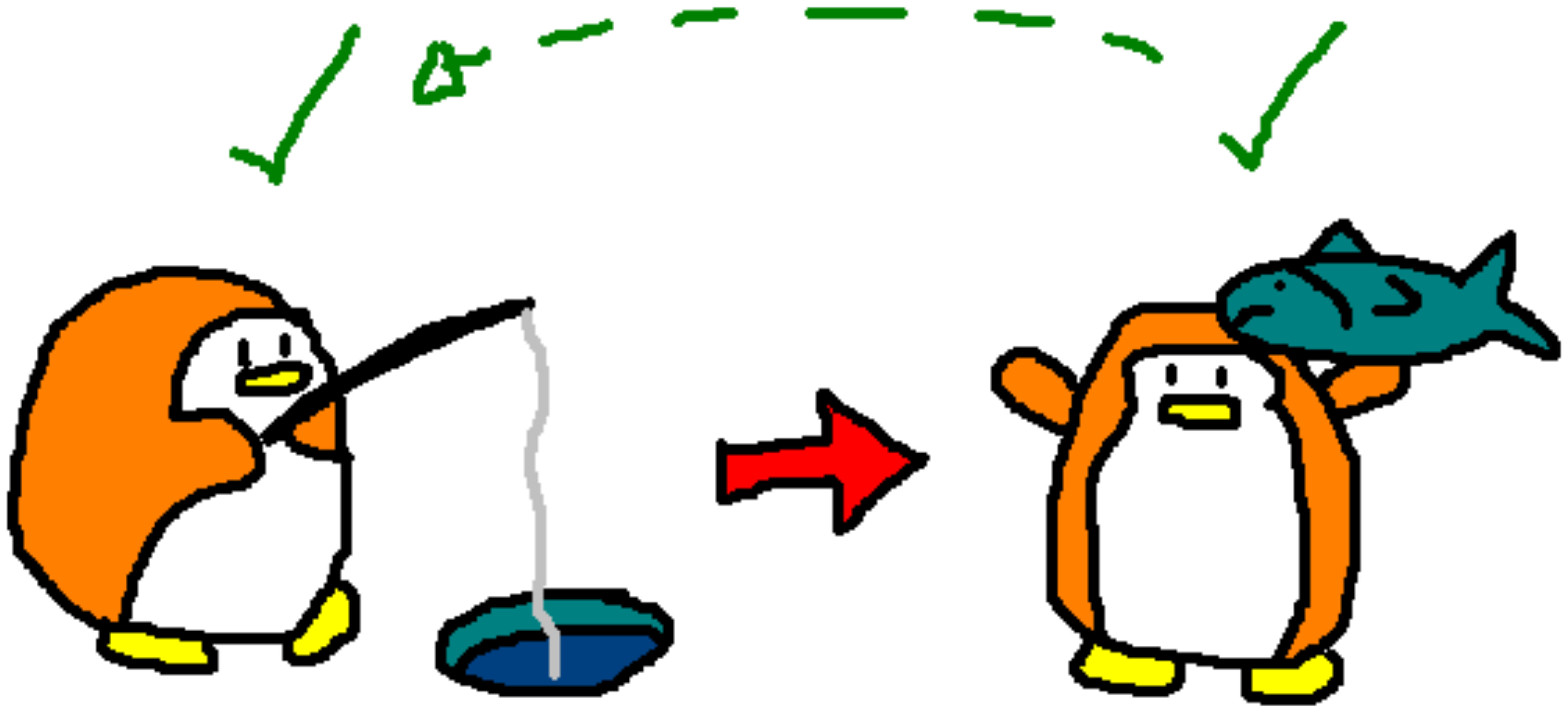
Temporal Difference Learning

New experience:



Temporal Difference Learning

New experience:



TD Learning

- Estimates value function $V(s) \approx v_\pi(s)$
    - TD is for Policy Evaluation (prediction), Sarsa and Q-learning are TD variants for control
- Combines ideas from Monte Carlo and Dynamic Programming - uses a mix of sampled information and bootstrapping off of current estimates
- Can learn *online*, without having to wait for the end of an episode

TD Learning Algorithms

One-step TD (or TD(0)):

$$V(S_t) \leftarrow V(S_t) + \alpha\left[\hat{G}_t - V(S_t)\right]$$

$$\hat{G}_t = R_{t+1} + \gamma V(S_{t+1})$$

One-step Sarsa (or Sarsa(0)):

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\left[\hat{G}_t - Q(S_t, A_t)\right]$$

$$\hat{G}_t = R_{t+1} + \gamma Q(S_{t+1}, A_{t+1})$$

Self-test: What makes Sarsa a control algorithm?

Sarsa (on-policy TD control) for estimating $Q \approx q_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

Loop for each episode:
  Initialize S
  Choose A from S using policy derived from Q (e.g., ε-greedy)
  Loop for each step of episode:
    Take action A, observe R, S'
    Choose A' from S' using policy derived from Q (e.g., ε-greedy)
    $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma Q(S', A') - Q(S, A)\big]$
    $S \leftarrow S'$; $A \leftarrow A'$;
  until S is terminal

Can you get a policy evaluation variant of Sarsa? (i.e., TD that estimates action-values)
We would call this Sarsa for Prediction. By default, Sarsa means Sarsa for Control

TD Self-test

- What is the difference between **online** and **offline** updating?
- What is the difference between Sarsa, Q-learning, and Expected Sarsa?

Self-test: Variance in Updates

- We talked a bit about the variance in updates
- Why does variance in the update matter?
- Which do you think will have a lower variance update: MC or TD, for prediction?
- Which do you think will have a lower variance update: Sarsa or Expected Sarsa?

Course Roadmap

Bandits (Ch2) → MDPs, returns, value functions (Ch3) → Dynamic programming (Ch4)

→ Monte Carlo learning (Ch5) → TD learning (Ch6) →

Planning (Ch8)

Planning, Learning and Acting

- Planning: a process which takes a **model** as input and produces or improves a **policy**

- Dyna uses a model to **simulate experience** and improve its value estimates, where greedifying with respect to these value estimates produces an improved **policy**

.

Dyna Self-test

- What is a model? (where model has a technical definition for RL and this course)
  - A procedure that produces a possible next state and reward, for a given state and action
- What is the difference between **simulated** and **real** experience?
- Explain the exploration / exploitation trade-off in model-based RL. How does it differ from the trade-off in the model-free setting?
- Describe at a high level how the Dyna-Q algorithm works?

Course Roadmap

Bandits (C2) → MDPs, returns, value functions (C3) → Dynamic programming (C4)

→ Monte Carlo learning (C5) → TD learning (C6) →

Planning (C8) → ~~Function approximation (C9)~~

Comments about Test Answers

- The goal is to see your thought process. Try to explain your answer clearly and give reasons, rather than just writing down an answer (but still be concise)

- Don’t vomit on the page: if you write many answers, and some of them are wrong, I will mark the wrong ones

- I don’t give partial marks for wrong answers, but if you demonstrate understanding (but its wrong) then I might give partial marks for that

- **One common problem**: answering a different question than is asked

  - after answering, re-read the question and check if you answered it

  - If you think: “this is a simple question, she must have meant this other more complicated thing”, you are probably wrong. I probably meant the simple thing.