Midterm Review

CMPUT 397
Fall 2019

- Link for questions:
  - **http://www.tricider.com/brainstorming/3Bjwvv5RURp**

Course Roadmap

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

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

Planning (Ch8)

Course Roadmap

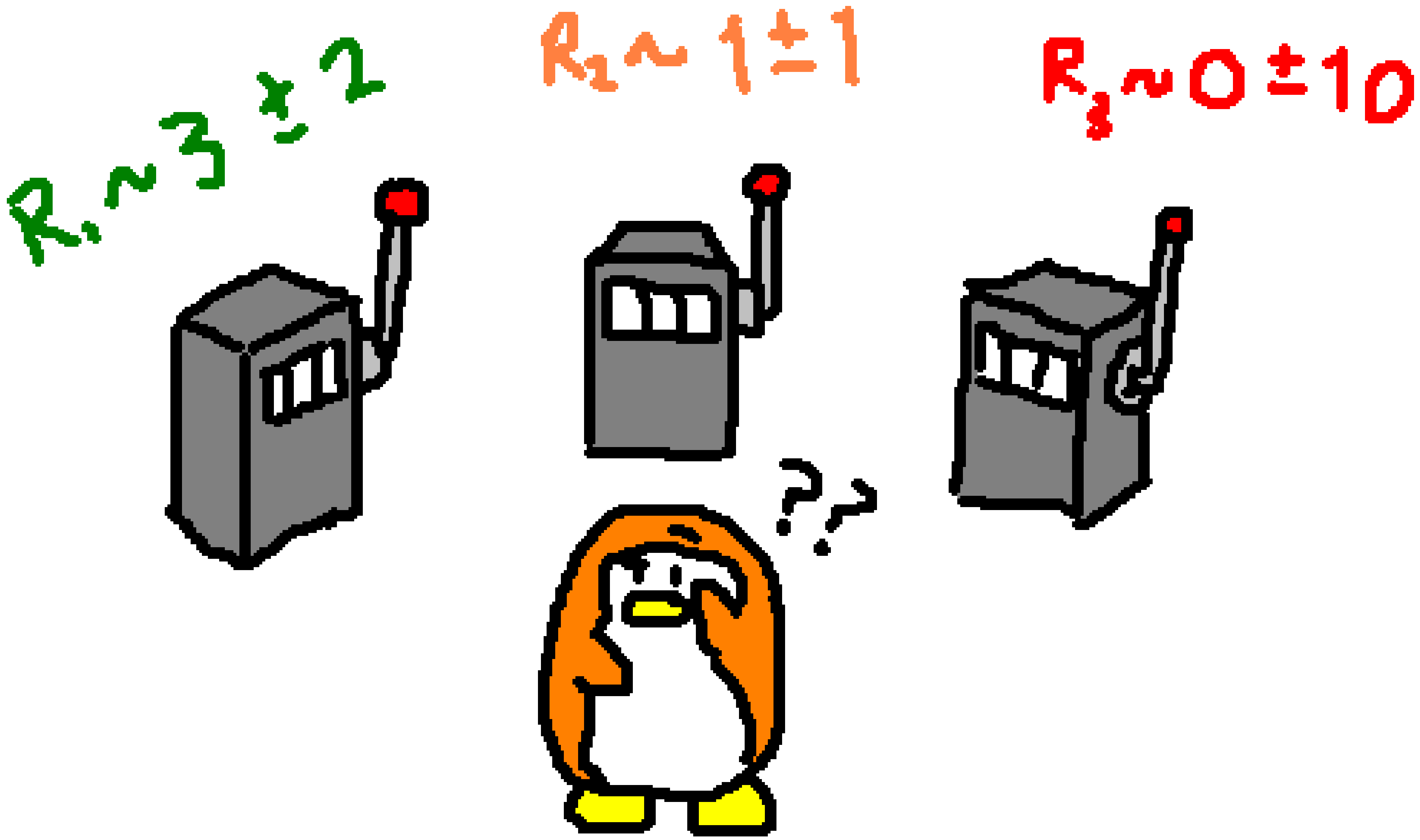
$\boxed{\text{Bandits (Ch2)}}$ → MDPs, returns, value functions (Ch3) → Dynamic programming (Ch4)

→ 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

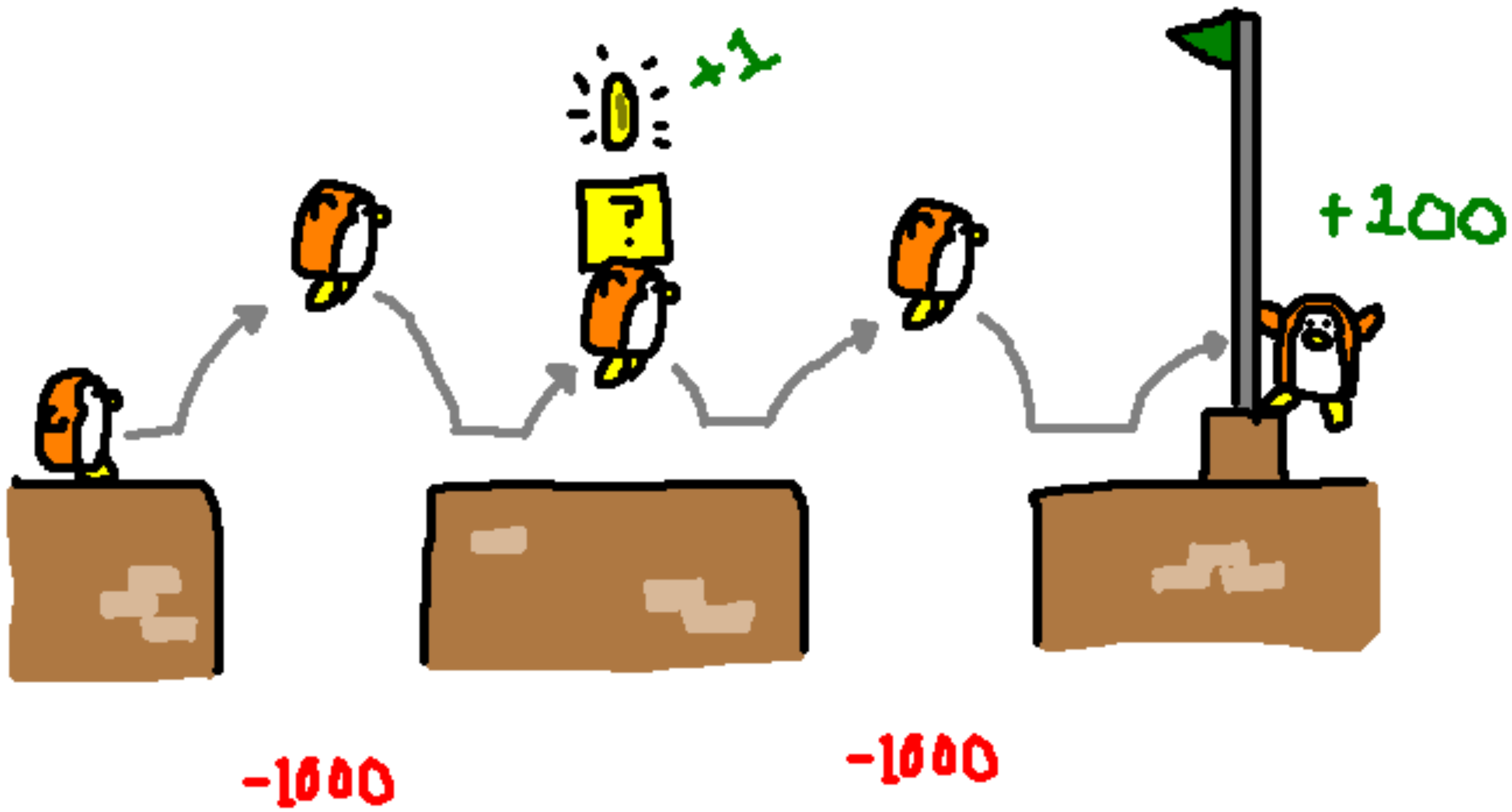
Bandits (Ch2) ➡ $\boxed{\text{MDPs, returns, value functions (Ch3)}}$ ➡ Dynamic programming (Ch4)

➡ 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** (estimating and/or maximizing):

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

- Specifically, the **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 what the **expected return: 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:

$$v_\pi(s) \doteq \mathbb{E}_\pi\left[G_t \mid S_t = s\right]$$

$$= \sum_a \pi(a \mid s) \sum_{s'} \sum_r p(s', r \mid s, a)\left[r + \gamma v_\pi(s')\right]$$

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 $q^*(s,a)$

Course Roadmap

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

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

Planning (Ch8)

Dynamic Programming

- Policy Evaluation
- Computes an approximate value function $V \approx v_\pi(s)$
- 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)

Dynamic Programming Self-test

- Given an MDP and a value function (i.e. all zeros), could you write out what the values would be after a sweep of a dynamic programming algorithm (i.e. policy evaluation)

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

- Implications of the policy improvement theorem:
  - i.e. if a policy is greedy wrt to it's own value function, what does that tell you?

Course Roadmap

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

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

Planning (Ch8)

Monte Carlo Learning

- Policy Evaluation: computes an approximate value function $V \approx v_\pi(s)$
- Sample *returns* from states by **following policy π**, then average those *returns* for each state
  - Self-test: Why do we use a sample average of the returns?

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 trajectory of experience (states, actions, rewards), some initial value function (i.e. all zeros), and parameters (step size, discount rate), can you write out the value updates that (every-visit) Monte Carlo would have performed?
-

Course Roadmap

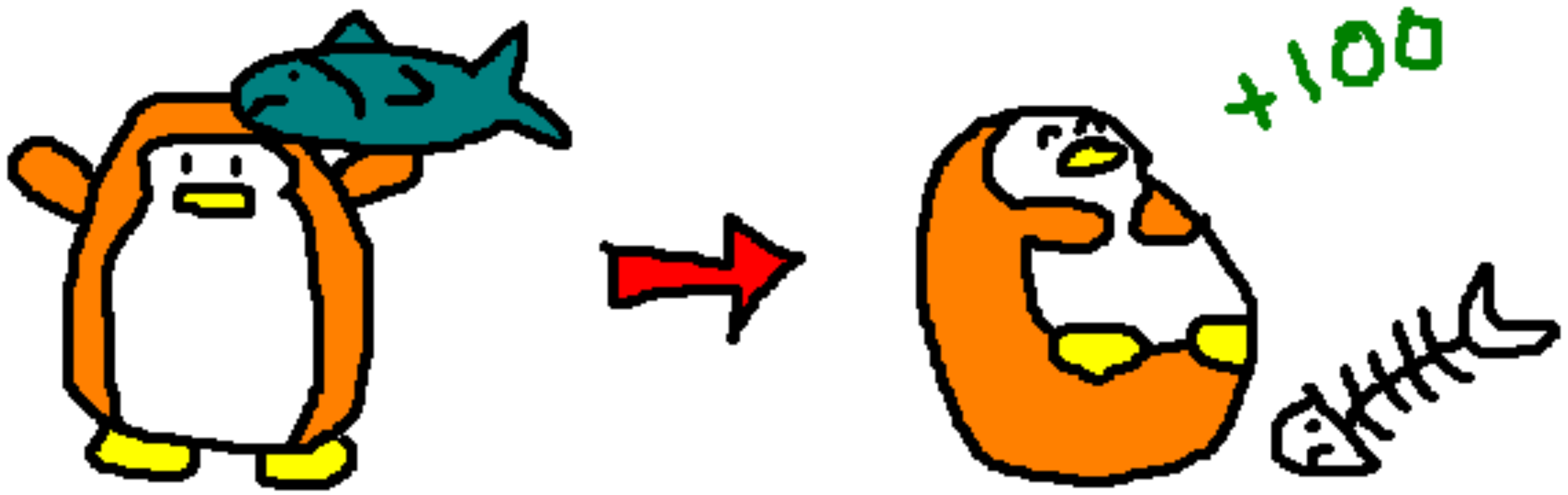
Bandits (Ch2) ➡ MDPs, returns, value functions (Ch3) ➡ Dynamic programming (Ch4)

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

Planning (Ch8)

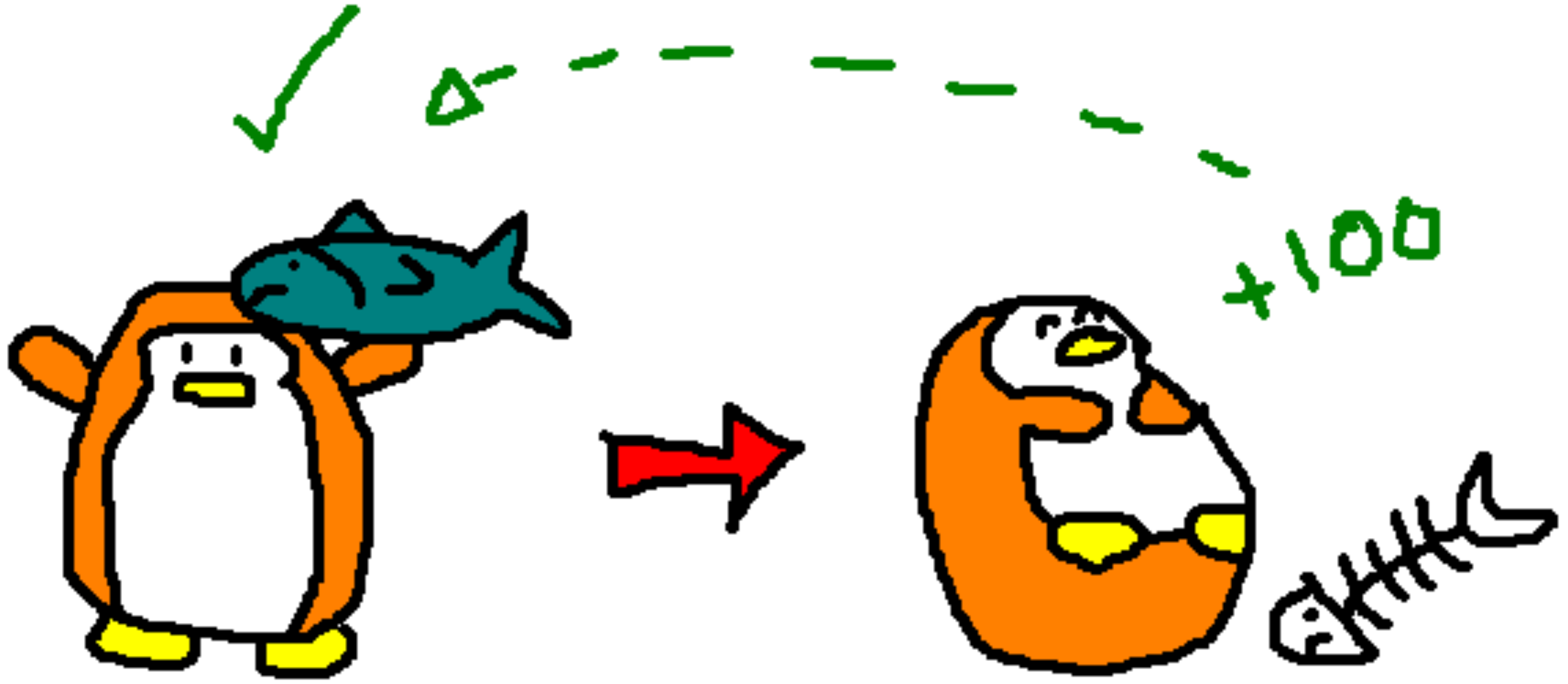
Temporal Difference Learning

Previous experience:



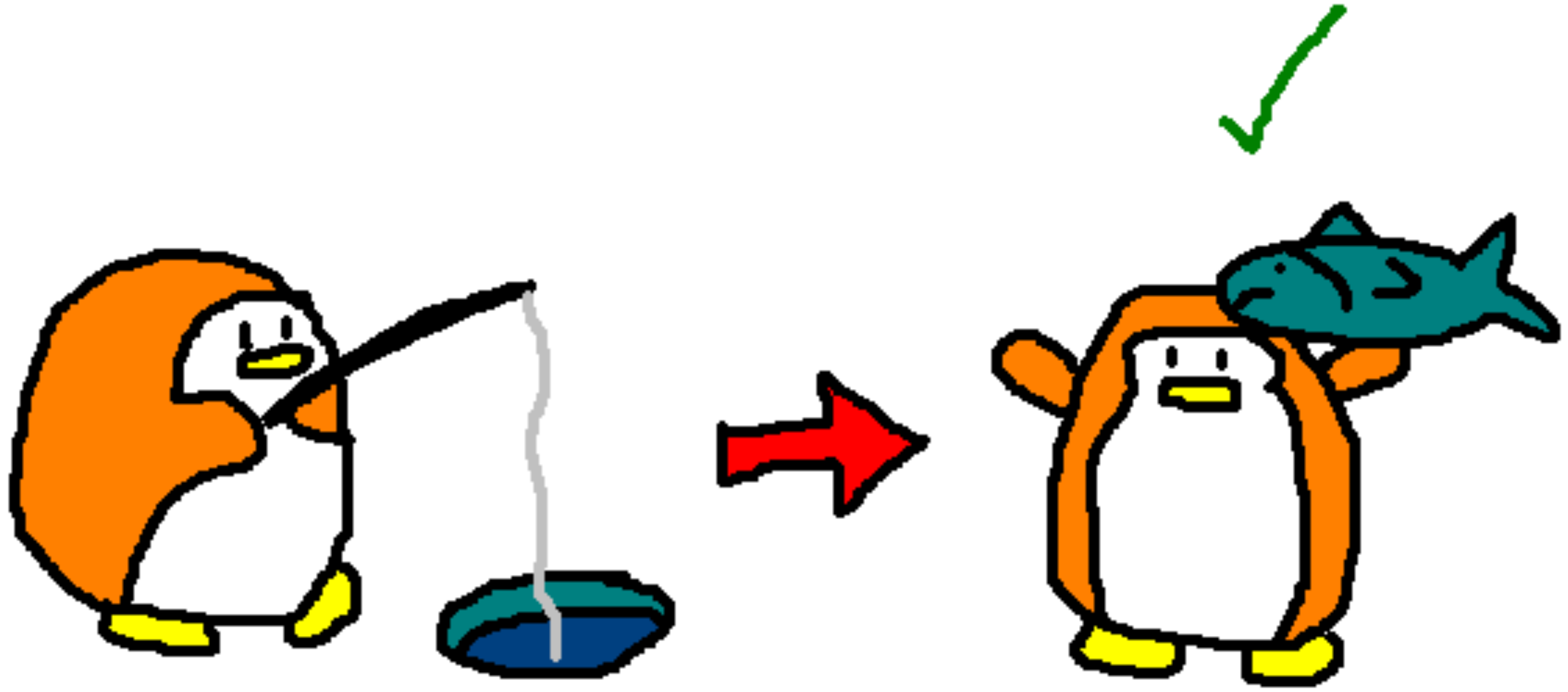
Temporal Difference Learning

Previous experience:



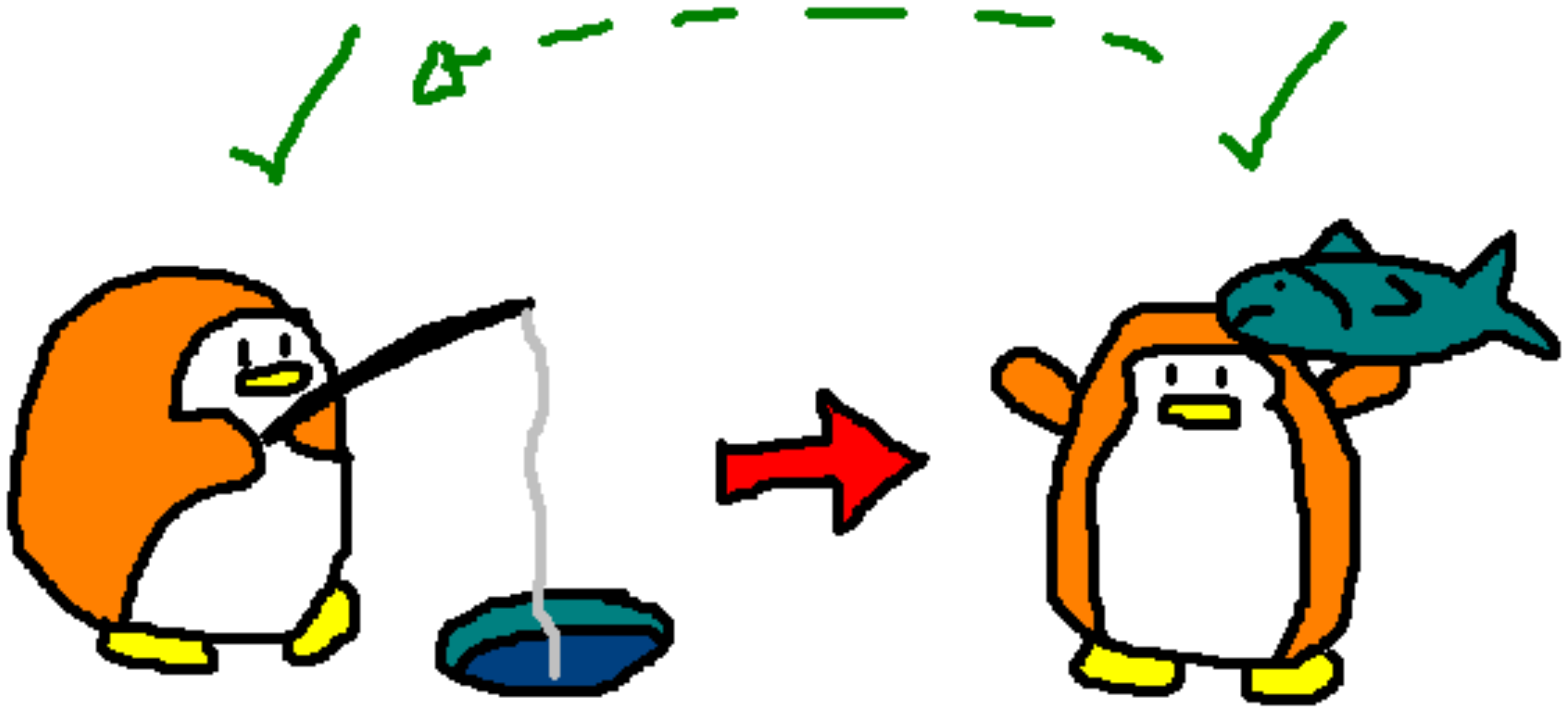
Temporal Difference Learning

New experience:



Temporal Difference Learning

New experience:



TD Learning

- Computes an approximate value function $V \approx v_\pi(s)$
  - Policy Evaluation
- 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

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)):

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

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

TD Self-test

- What is the difference between **online** and **offline** updating?
- What is the difference between Sarsa, Q-learning, and 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?
- 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 ap[illegible]mation (C9)