Week 8 — Decisions, MDPs & Reinforcement Learning

Friday, June 19, 2026

Prof. Joseph Austerweil

Agenda

From beliefs to actions: decision theory	0:00
One decision → a sequence of decisions	0:21
Markov Decision Processes	0:29
Solving MDPs: value iteration	0:46
Break	1:04
Learning by doing: Q-learning	1:09
Interactive: reward shaping & cycles	1:27
Where RL is now	1:42

From beliefs to actions

Admin

Active deadlines:

- **Generalization assignment** — due **tonight, Fri Jun 19, 8:00 PM**
- **Final project proposal — Sun Jun 28, 8:00 PM** *(grab 5 min with me to talk through an idea)*
- **Monte Carlo assignment — Fri Jul 10, 8:00 PM**

The **Reinforcement Learning assignment** (Q-learning on today's GardenPath) releases **next week**. Questions before we start?

Seven weeks of *beliefs*. Today: *actions*.

Weeks 1–7 asked one question: **given data, what should I believe?** The normative answer was **Bayes** — update the posterior.

Today the question changes: **given my beliefs, what should I *do*?**

An agent has to *act*. Decision theory is the bridge from a posterior to an action — and the brain seems to use it too (Körding & Wolpert, 2004).

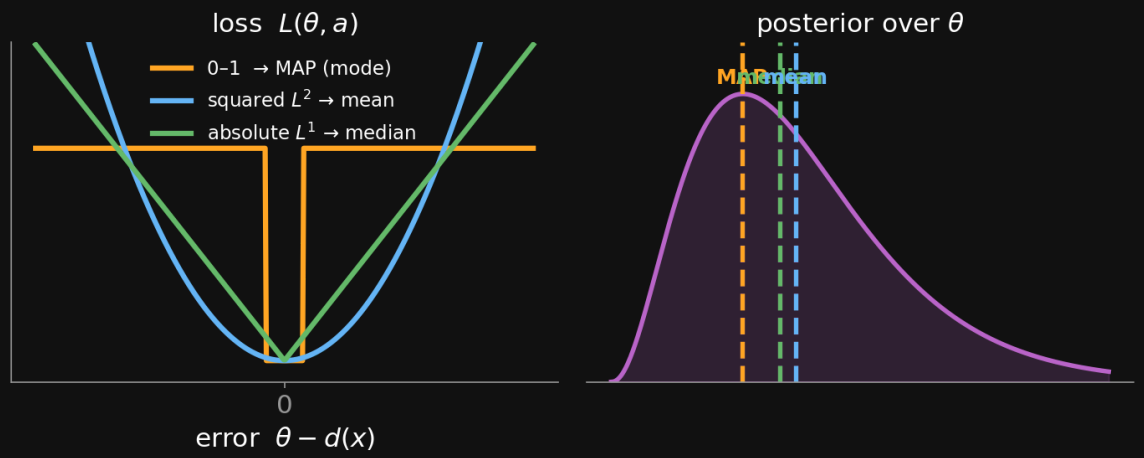
Statistical decision theory

A decision problem under uncertainty has five pieces:

- **state of the world** θ — what's true but unknown (is the tonkatsu fresh today?)
- **observation** x — what you see (the shop looks busy)
- **actions** A — what you can do (order tonkatsu / play it safe)
- **decision rule** $d(x)$ — your strategy: which action, given x
- **loss** $L(\theta, a)$ — what it costs to take action a when the world is θ

Which loss → which estimate?

Suppose Chibany must report a single number — his estimate of his bento's **weight** (a posterior from Week 2). The loss you'll be scored by decides the best estimate:



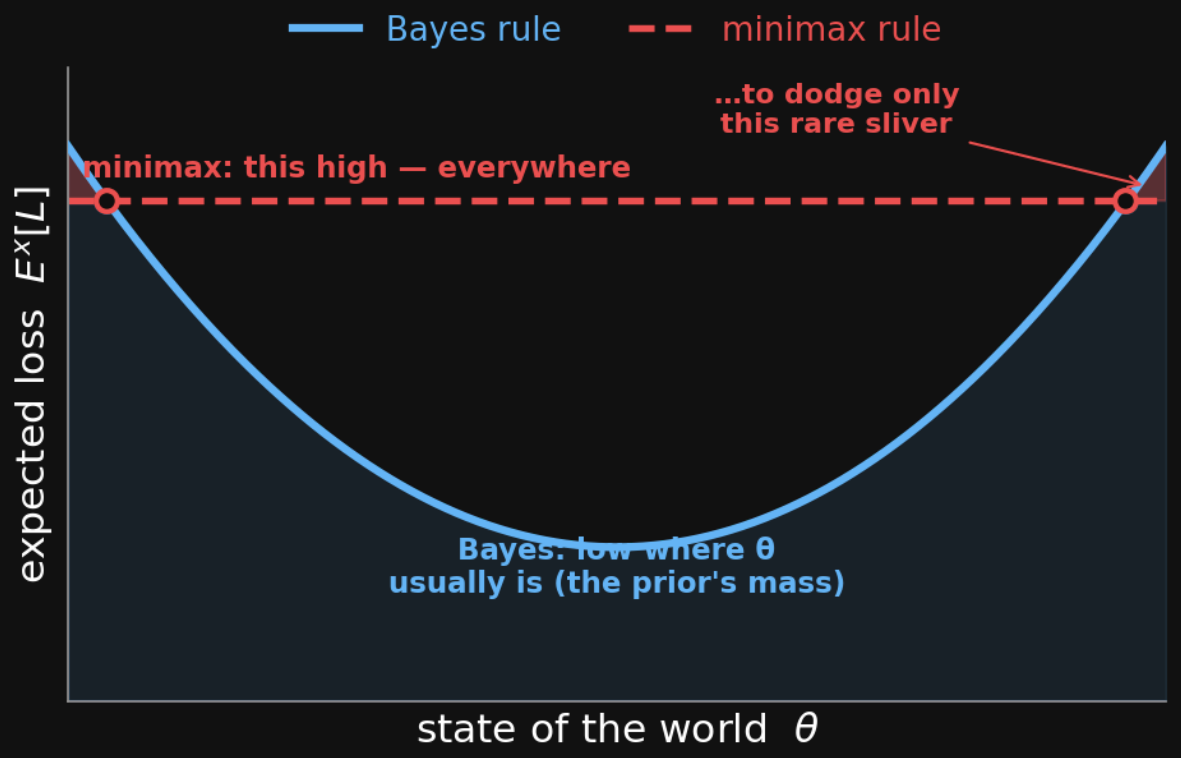
- **0–1 loss** (right/wrong) → report the **mode** = **MAP** (the posterior's peak)
- **squared loss** $(\theta - a)^2$ → report the **mean**
- **absolute loss** $|\theta - a|$ → report the **median**

The "best action" is *not* fixed — it depends on how you'll be judged.

How good is a rule? Bayes vs. minimax

The **risk** of a rule is its *expected* loss: $R(\theta, d) = \mathbb{E}_x[L(\theta, d(x))]$ — where $\mathbb{E}_x[\cdot]$ just means "the average over the data x." Two ways to choose a rule:

- **Bayesian** — have a prior over θ, minimize the **average** risk: $\arg\min_d \mathbb{E}_\theta \mathbb{E}_x[L]$
- **Minimax** — no prior, minimize the **worst case**: $\arg\min_d \max_\theta \mathbb{E}_x[L]$



Poll — which estimate?

You must report one number for an unknown quantity. You'll be scored by **absolute error**, $|\theta - a|$. Which summary of your posterior should you report?

- **A. the mode (MAP)**
- **B. the mean**
- **C. the median**

Poll — answer

C — the median.

Absolute (L^1) loss is minimized by the **median**; squared (L^2) by the mean; 0–1 by the mode. The loss you're judged by picks the estimate — that's the whole point of decision theory.

One decision → a sequence

What about a *sequence* of actions?

Decision theory tells you how to take **one** action. But life is a *sequence* — today's action changes tomorrow's situation.

"In theory, no difference": let the action variable be the **whole sequence**. But over a horizon of H steps there are $|A|^H$ sequences to compare — a combinatorial explosion. **Intractable.**

The rescue: assume the world is **Markov** — the next state depends only on the current state and action. Then the problem factorizes, and we can solve it step by step (dynamic programming).

Discount the future

We won’t just *sum* future rewards. Would you rather have **¥1000 now or ¥1000 in 1000 years?**

We **discount** the future by a factor $0 \leq \gamma \leq 1$ per step. The **return** from time t is

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}.$$

Small γ = short-sighted; γ near 1 = far-sighted. We’ll see γ decide a real choice in a few slides.

Where we are

From beliefs to actions: decision theory	✓
One decision → a sequence of decisions	✓
Markov Decision Processes	0:29
Solving MDPs: value iteration	0:46
Learning by doing: Q-learning	1:09
Where RL is now	1:42

Markov Decision Processes

Start where Week 6 left off

In Week 6, Chibany's bento was a **Markov chain**: one transition matrix P, no choices. Today we turn that chain into an agent.

Markov chain + reward = the simplest MDP

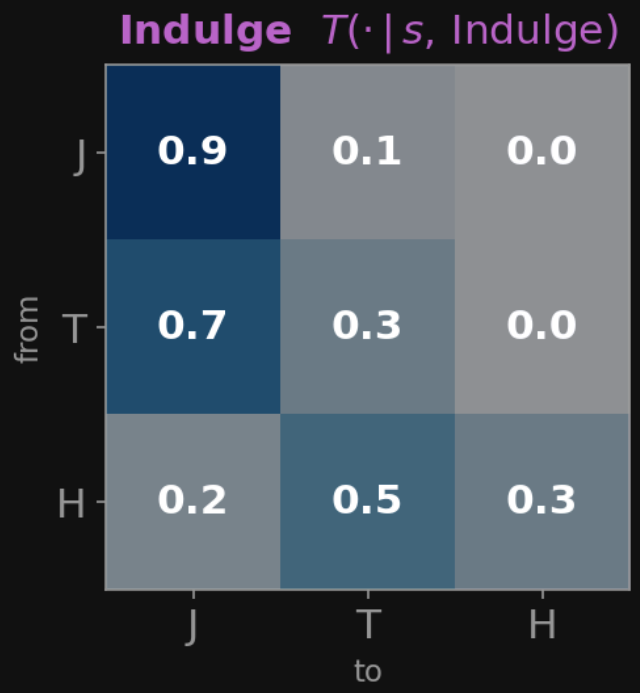
Attach a **reward** $R(s)$ to each state — how good it is to be there. A Markov chain with a reward is a **one-action MDP**: the dynamics just run; you collect (discounted) reward.

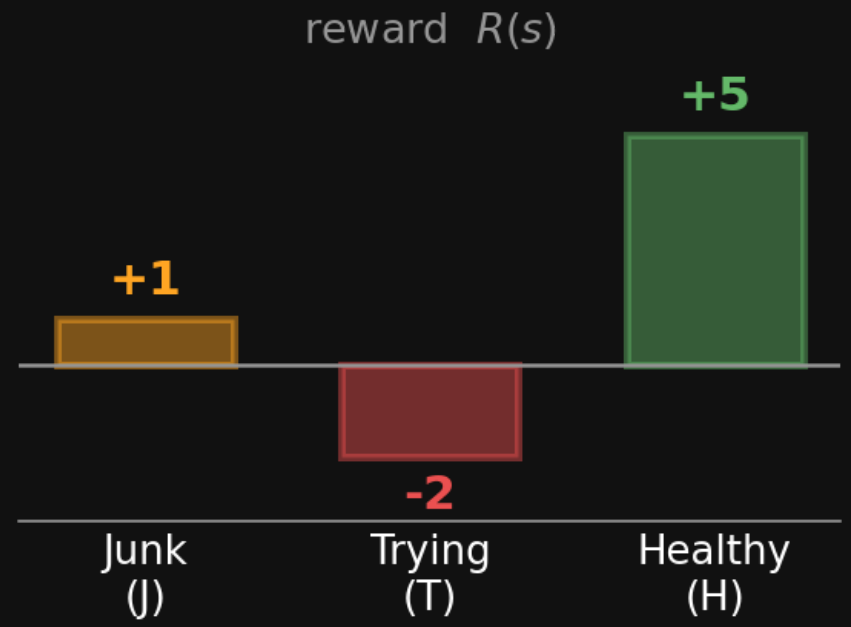
Chibany's wellbeing has three states:

- **Junk rut** ($R = +1$) — cheap, comfortable, going nowhere
- **Trying** ($R = -2$) — effortful, no payoff *yet*
- **Healthy & happy** ($R = +5$)

An action = which transition matrix to use

Now give Chibany a **choice** — each action is its own transition matrix: **Indulge** (order out) drifts *down*, **Invest** (cook, exercise) climbs *up*.





Picking an action = picking which matrix governs tomorrow. (Rewards **J / T / H** repeated as a reminder.)

A Markov Decision Process, formally

Markov chains + decisions + rewards = MDPs. An MDP is (S, A, T, R, γ):

- S — set of **states**
- A — set of **actions** (the decision rules of decision theory)
- T — **transition** $P(s_{t+1} \mid s_t, a_t)$: one matrix per action *(a Markov chain is the 1-action case)*
- $R(s, a)$ — **reward** (= $-$ loss)
- γ — **discount** factor

Maximizing reward is minimizing loss, flipped. Chibany's reward sits on the *state*, $R(s, a) \equiv R(s)$; in general it can depend on the action too.

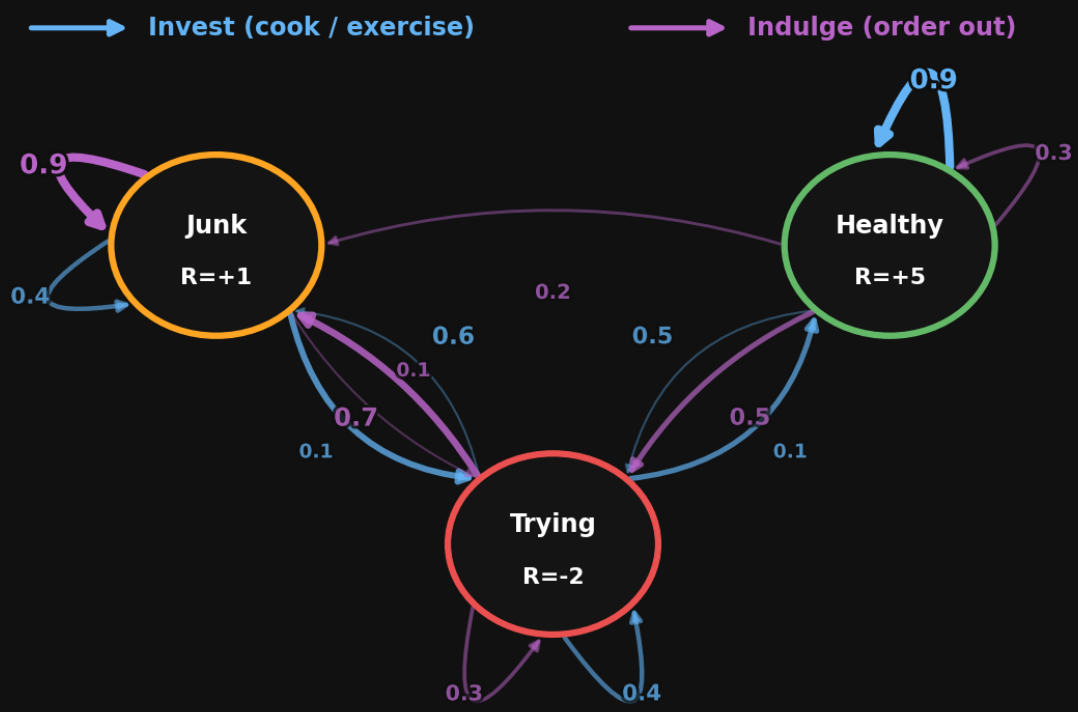
A policy is *how you choose*

A **policy** π says which action to take in each state:

$$\pi(a \mid s) = P(a_t \mid s_t).$$

Because the world is Markov, the policy only needs the **current** state — not the whole history. The goal: find the policy that **maximizes expected discounted reward**.

The Chibany MDP



every transition for both actions — each state's out-arrows sum to 1 per action

The chokepoint: the only road to **+5** runs through the **−2** trough.

What is the optimal policy? — that's what we solve next.

Notation lock-in

Before the math, the symbols we'll use from here on:

- $s \in S$ state; $a \in A$ action; $\pi(a \mid s)$ policy
- $T(s' \mid s, a)$ transition; $R(s, a)$ reward; γ discount
- $V(s)$ value of a state; $Q(s, a)$ value of an action
- math shorthand: $\mathbb{E}$ = average; $\sum$ = add up; $\arg\max_a$ = the *action* that maximizes; prime $'$ = "next"

Poll — should Chibany invest?

Chibany is stuck in the **Junk rut** ($R = +1$). To ever reach **Healthy** ($R = +5$) he must pass through **Trying** ($R = -2$). If he **barely weighs the future** (small γ), what's his optimal action?

A. Invest — push through the trough

B. Indulge — stay in the rut

Solving MDPs

The value of a state

Under a policy π, two value functions:

- **state value** — expected return from s: $v_\pi(s) = \mathbb{E}_\pi[G_t \mid S_t = s]$
- **action value** — expected return from taking a in s, then following π: $q_\pi(s, a) = \mathbb{E}_\pi[G_t \mid S_t = s, A_t = a]$

The **best** policy is the one with the highest value everywhere: $\pi^* = \arg\max_\pi v_\pi(s)$.

(Lowercase v, q = the *true* values; the algorithm's running estimates we'll write as V, Q.)

The Bellman equation

Value is **recursive** — now's value = immediate reward + discounted value of where you land:

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

This *recursion* is the door: a value defined in terms of itself can be solved by **dynamic programming**.

Value iteration

Turn the Bellman equation into an **algorithm** — just apply it as an update, over and over:

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

Start from $V_0 = 0$; sweep all states; repeat. It **converges** to v^*. Then read off the policy: in each state, take the action that achieved the max. We *know* T and R, so we can just **compute** the answer.

(It converges because the Bellman update is a γ-contraction — so it has one fixed point, v^*, reached at rate γ.)

Value iteration by hand

$$V_{k+1}(s) = \max_a \Big[R(s) + \gamma \sum_{s'} T(s' \mid s, a)\, V_k(s') \Big], \qquad \gamma = 0.9$$

Start $V_0 = 0$. **Sweep 1:** the future is still 0, so $V_1(s) = R(s)$. **Sweep 2:** back up with $V_1 = [1, -2, 5]$ — for **Junk**, compare both actions:

- **Indulge**: $1 + 0.9\,(0.9{\cdot}1 + 0.1{\cdot}(-2)) = 1 + 0.9(0.7) = \mathbf{1.63}$
- **Invest**: $1 + 0.9\,(0.4{\cdot}1 + 0.6{\cdot}(-2)) = 1 + 0.9(-0.8) = 0.28$

→ Junk still picks **Indulge** — investing leads to *Trying* (still −2), not yet worth it.

after sweep k	V(J)	V(T)	V(H)
0	0	0	0
1	+1.0	−2.0	+5.0
2	**+1.6**	**−0.4**	**+8.9**

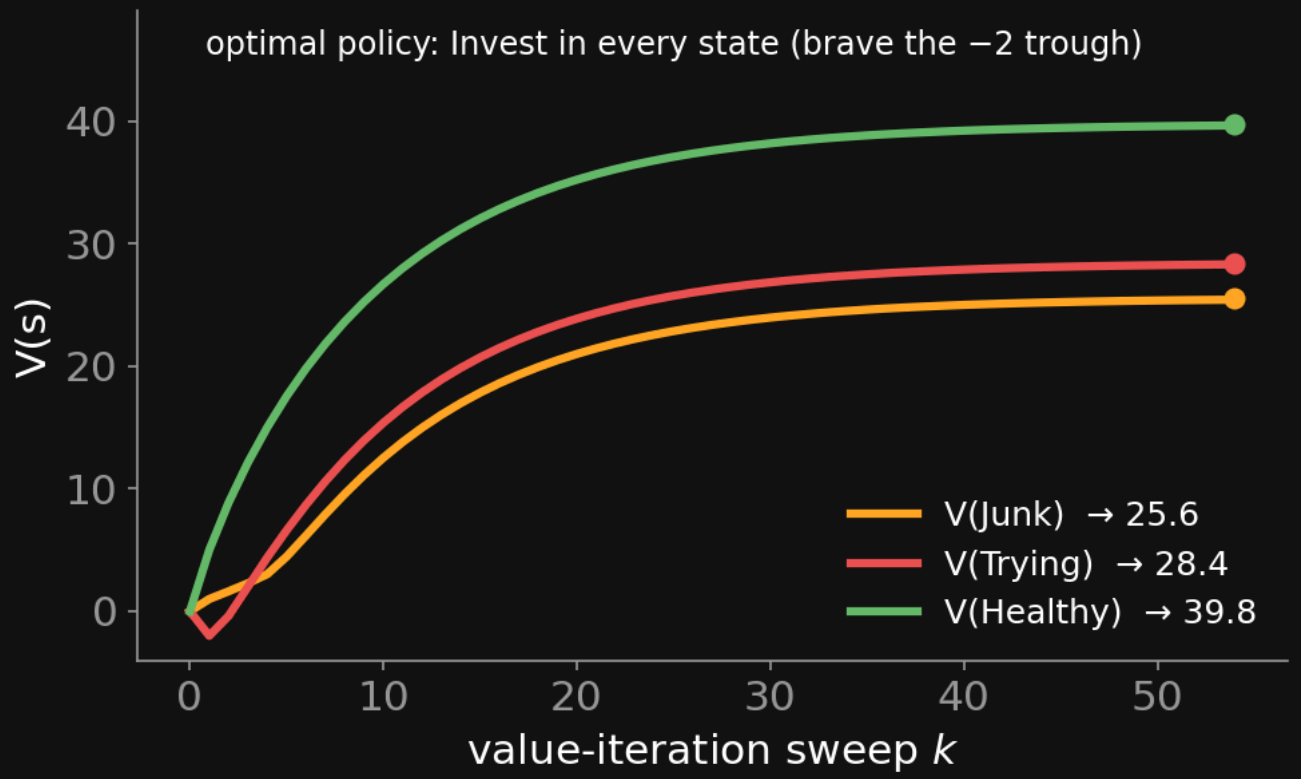
Keep sweeping — the policy flips

As **Healthy's +5** propagates back through Trying, investing starts to pay. By **sweep 5**, Junk **flips from Indulge to Invest** — and the values climb to their fixed point:

after sweep k	V(J)	V(T)	V(H)	Junk picks
2	1.6	−0.4	8.9	Indulge
4	3.0	4.4	15.0	Indulge
5	**4.5**	6.6	17.6	**Invest ✓**
…	…	…	…	Invest
∞	**25.6**	**28.4**	**39.8**	**Invest**

Next: watch all three values converge.

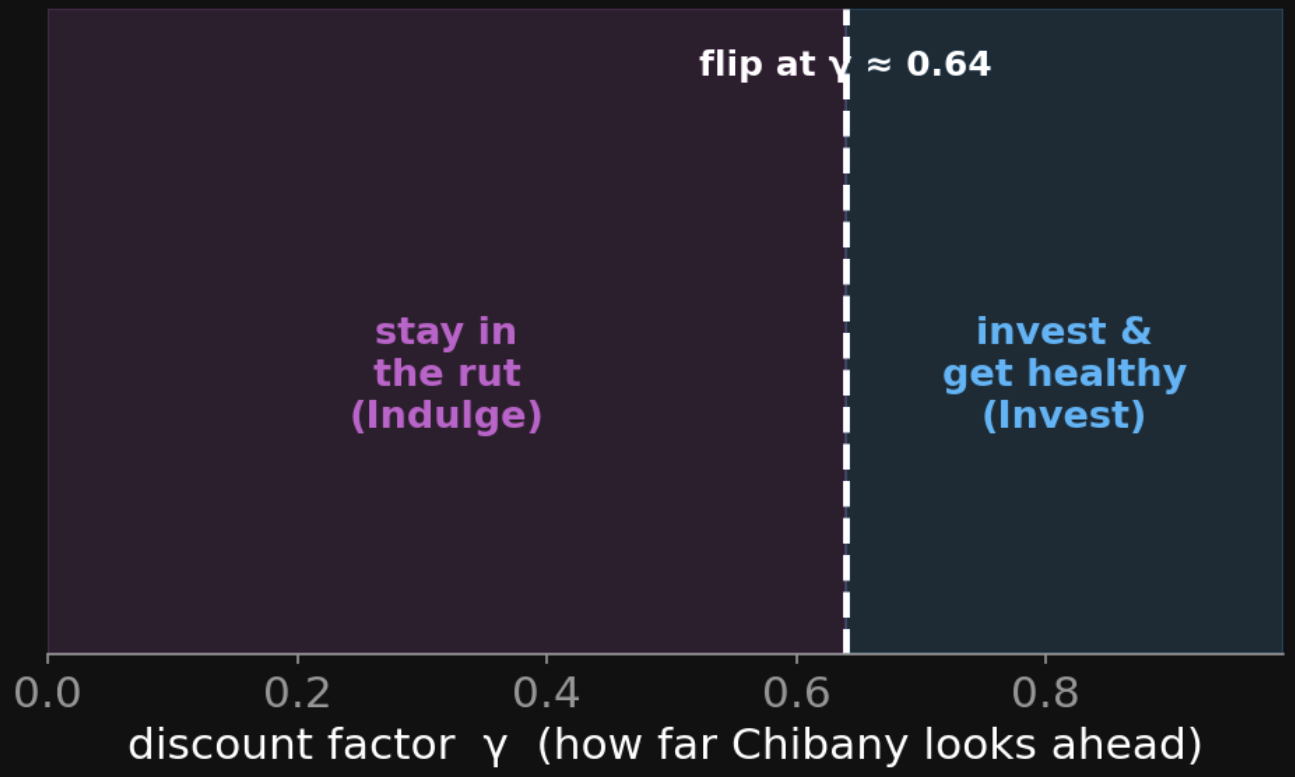
Back to Chibany: value iteration



With $\gamma = 0.9$, the values climb and settle, and the optimal policy is **Invest everywhere** — Chibany braves the **−2** trough because **+5** down the line is worth it.

This is what we never did in past years: actually **solve the MDP we drew.**

How far ahead? γ decides



Poll answer: B (Indulge) for small γ.

Sweep γ: below **≈0.64** (for *these* transition probabilities) the optimal action in the rut is **Indulge** — the future is too discounted to be worth the trough; above it, **Invest**. How far Chibany looks ahead decides whether he escapes.

The Chibany MDP in GenJAX

The transition is a **generative model** — the *action* picks which distribution governs the next state:

```
import jax.numpy as jnp
from genjax import gen, categorical

# states J,T,H;  actions Indulge=0, Invest=1
T = jnp.array([[[.9,.1,0.],[.7,.3,0.],[.2,.5,.3]],   # Indulge  T[0,s,s']
               [[.4,.6,0.],[.1,.4,.5],[0.,.1,.9]]])  # Invest   T[1,s,s']
R = jnp.array([1.,-2.,5.]);  gamma = 0.9

@gen                              # action a selects the next-state distribution
def transition(s, a):
    return categorical(jnp.log(T[a, s])) @ "s_next"

transition.simulate(key, (0, 1)).get_retval()   # sample s' from (Junk, Invest)
```

Solve it: compute *or* simulate

We know T and R, so **value iteration computes** the answer:

```
def bellman(V):
    Q = R[None,:] + gamma * (T @ V)          # Q[a,s] = R(s) + γ Σ T(s'|s,a) V(s')
    return jnp.max(Q, 0), jnp.argmax(Q, 0)

V, policy = value_iteration()                # → V = [25.6, 28.4, 39.8]
                                             #   policy = Invest everywhere
```

Or **simulate** that same model and average the returns → $\hat{V}(\text{Junk}) = 25.7 \approx 25.6$. *Compute or simulate — both agree.*
Runnable: `genjax_chibany_mdp.py`.

If you know the model, you can just simulate

Value iteration needed the **whole model** — every $T(s' \mid s, a)$ and $R(s, a)$. If you have that, you can **sit and simulate** the optimal policy *before acting*. No learning required.

But in the real world, you usually **don't know** T or R. You have to **learn by doing.**

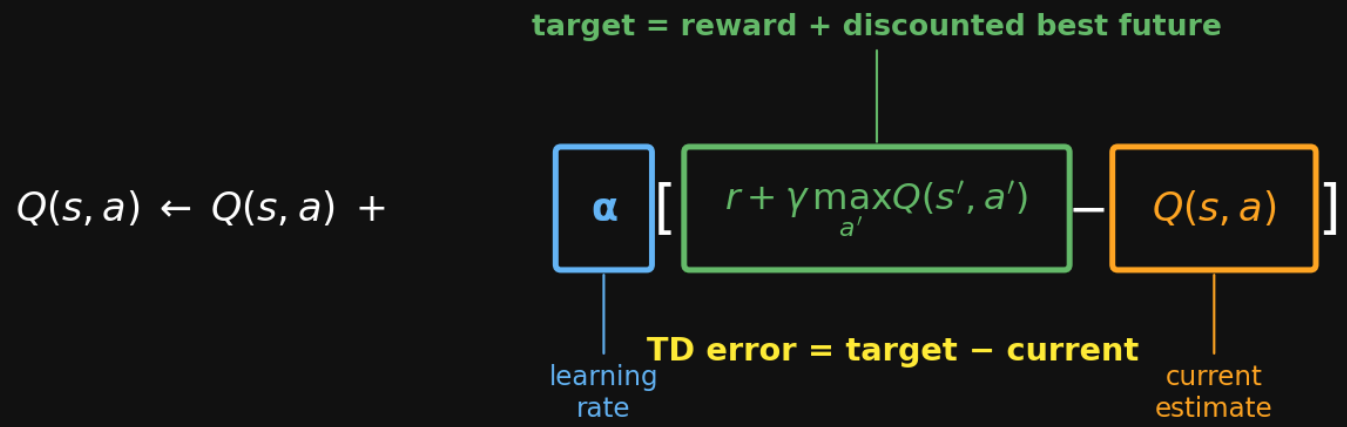
Learning by doing

No model? Learn from experience.

Chibany was easy: we *knew* his two transition matrices, so we could **compute** the answer. Now meet a new agent that **doesn't know its world** — no T, no R. It only gets to **act, see what happens, and adjust** — trial, feedback, update.

Q-learning learns the action-value $Q(s, a)$ directly from experience, with no model of the world. Once Q is learned, the policy is easy: in each state, take $\arg\max_a Q(s, a)$.

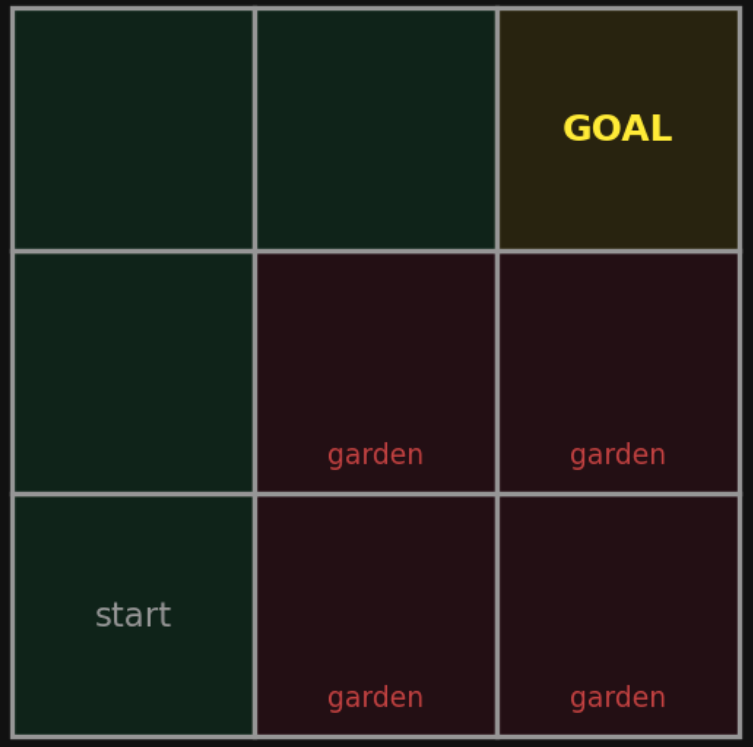
The Q-learning update



Move $Q(s, a)$ toward a **target** = reward + discounted best future; the gap is the **TD (temporal-difference) error** δ — *update whenever your prediction differs from what you see*. One step:

1. **select** a by **ε-greedy** — usually the best action, but random with prob. ε (keep **exploring**)
2. take a; observe r and next state s' (prime $'$ = “next”)
3. **target** $= r + \gamma \max_{a'} Q(s', a')$ · **TD error** $= target − current
4. **update**: $Q(s, a) \leftarrow Q(s, a) + \alpha\, \delta$ (α = **learning rate**)

The GardenPath



Path = left column + top row; Garden = bottom-right 2×2

A 3×3 world (Ho, Littman, Cushman & Austerweil, 2015). Get from **start** (bottom-left) to the **goal** (top-right) along the **path** (left column + top row), avoiding the **garden** (bottom-right).

The agent doesn't see this picture — it only knows it's in a grid, can move, and gets feedback. $\alpha = 0.9,\ \gamma = 0.95,\ \varepsilon = 0.1.$

Moves are deterministic; reaching the goal ends the episode.

One step, by hand

At the start, $Q = 0$ everywhere. The agent (ε-greedy) tries **RIGHT**, into the garden, and gets feedback -10. The next state's best value is still 0. Update:

$$Q(\text{start}, \text{R}) \leftarrow 0 + 0.9\,(\,-10 + 0.95 \cdot 0 - 0\,) = -9.$$

That -9 teaches the agent: "going right from the start is bad." Do this thousands of times and the values propagate back from the goal — let's watch it **live.**

Poll — will it reach the goal?

Teaching is hard, so you decide to help: give the agent **+10 every time it takes a "good" move toward the goal**, -10 for a bad move. Train Q-learning with this feedback. Will it learn to reach the goal?

A. yes — rewarding good moves obviously helps

B. no — it can get stuck looping forever

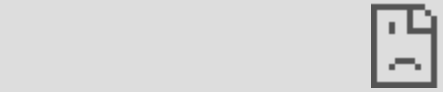
Interactive: reward shaping

What to watch

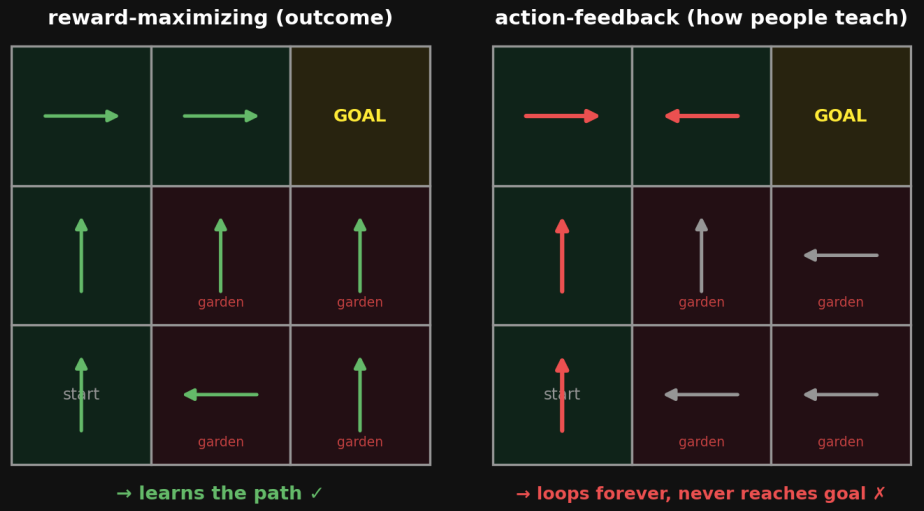
We'll run Q-learning on the GardenPath live and try four ways of giving feedback. For each, **watch the verdict** — does the *learned* policy actually reach the goal?

- **action-feedback** — reward each "good" move (the way people naturally teach)
- **reward-maximizing** — reward only at the goal, penalty in the garden
- **potential-based shaping** — reward progress, but carefully
- **human** — *you* give the feedback

(The cat still stumbles onto 🏠 while *exploring* — watch the **learned policy**, not luck.)



If the live demo isn’t available



Reward-maximizing learns the path (left). **Action-feedback** — rewarding each “good” move — makes the optimal policy a **positive reward loop** worth +20 per lap, so the agent circles forever and never reaches the goal (right). Open `week08.html` to drive it live, or be the teacher yourself.

What we saw

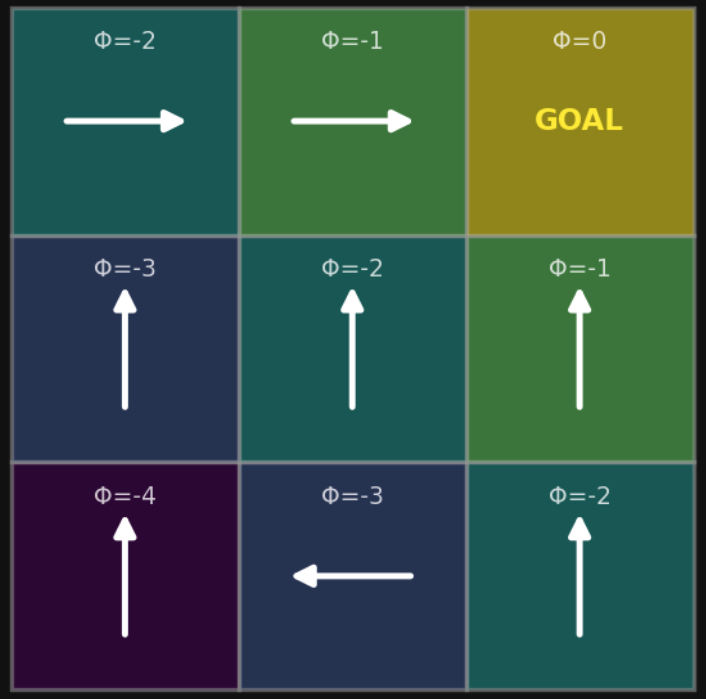
Action-feedback — the intuitive way — loops forever. (Poll: B.)

- **action-feedback** (reward good moves) → a **positive cycle**: the cat circles the garden (+20 a lap), never reaching the goal — led *down the garden path*.
- **reward-maximizing** (reward the outcome) → learns the correct path. ✓
- **potential-based shaping** → keeps helpful guidance but **can't create a cycle** (next slide).

The lesson: *how* you give feedback can quietly break the task — even when each reward seems helpful. We'll meet it again as **reward hacking** in modern AI.

The fix: potential-based shaping

potential-based shaping $F = \gamma\Phi(s') - \Phi(s)$



Φ = −dist; path recovered, no cycle possible ✓

Shape with a **potential** $\Phi(s)$ (here, closeness to the goal): add a bonus $F = \gamma\Phi(s') - \Phi(s)$ on top of the task reward.

This adds the **same constant** $-\Phi(s)$ to *every* action's value in a state, so the **best action never changes** (Ng, Harada & Russell, 1999). There's no loop you can ride for a net gain — the cycle vanishes, but the goal stays the goal.

Where we are

Decision theory → MDPs → solving them ✓

Q-learning: learning without a model ✓

Reward shaping & positive cycles ✓

Where RL is now 1:42

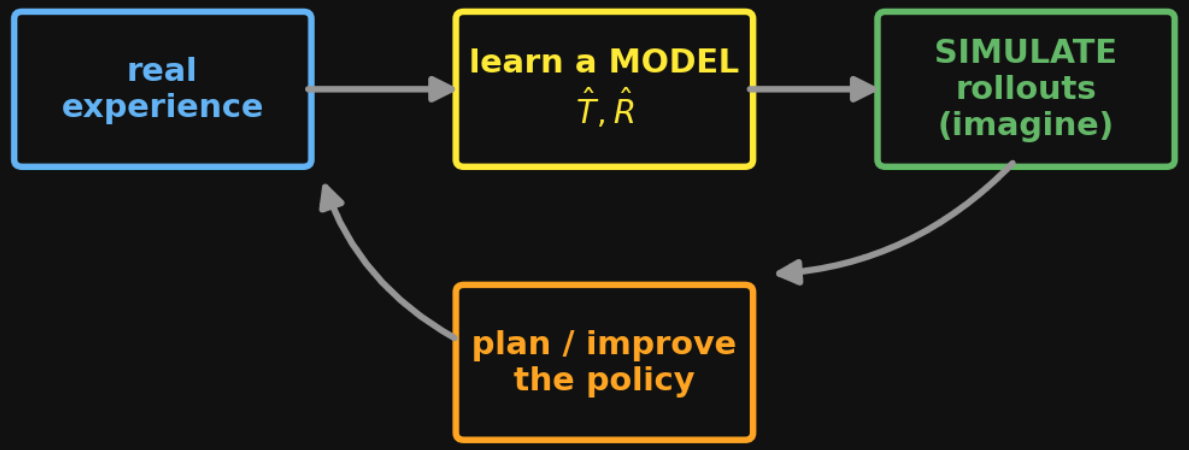
Where RL is now

Model-free vs. model-based

- **Model-free** (Q-learning): learn values straight from experience. Simple, but throws away structure — every state must be visited many times.
- **Model-based**: *learn a model* $\hat{T}$, $\hat{R}$, then **plan** with it.

I prefer **simulation-based** to “model-based” — because the whole move is to **learn a model and then simulate rollouts inside it** to plan. It’s exactly the Monte-Carlo simulation idea from Week 7, now in the service of acting.

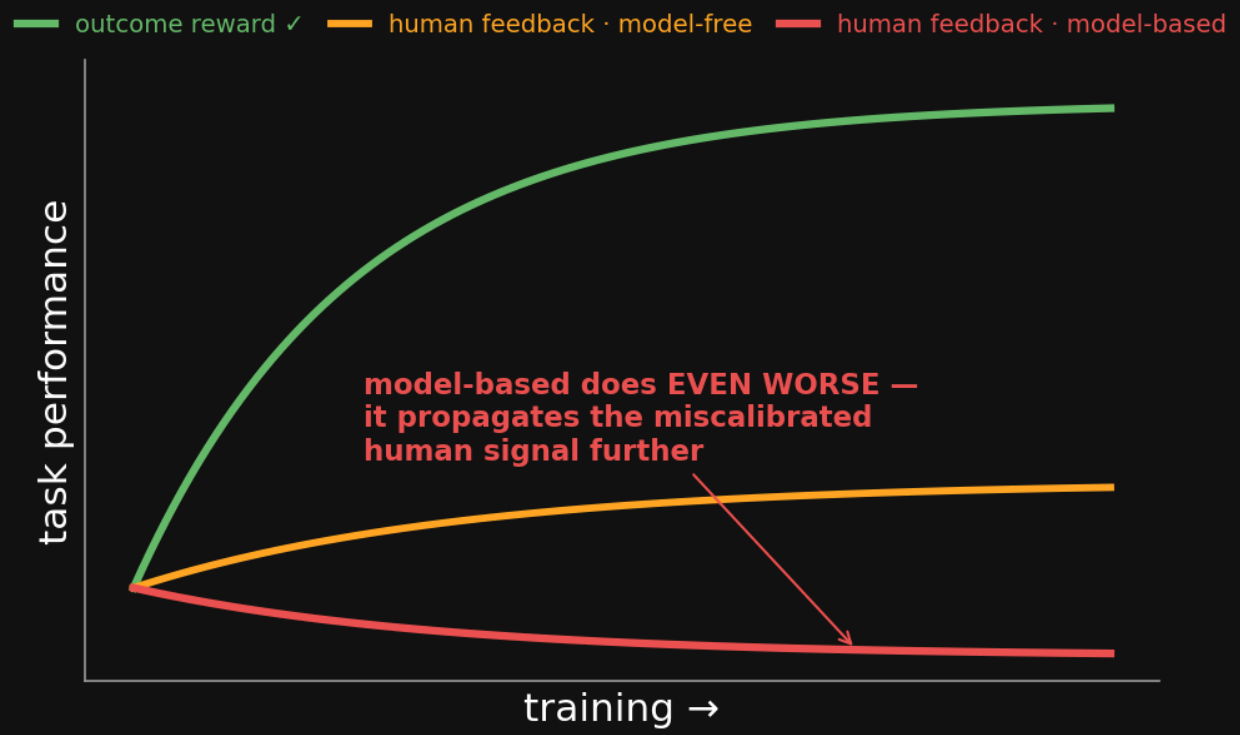
Learn a model, then plan by simulating



Dyna / AlphaZero / MuZero / Dreamer — plan by simulating a learned model

- **Dyna** — learn $\hat{T}$, then run value iteration on *imagined* experience
- **MCTS** — from "here," simulate many futures and average their outcomes to score each move
- **AlphaZero** — MCTS guided by a learned value/policy net (no human games)
- **MuZero** — learns a model that predicts only what planning *needs* (value, policy, reward), not the full next state — so it plans in an abstract *latent* space
- **Dreamer** — learns a world model, then trains the policy *inside the dream*

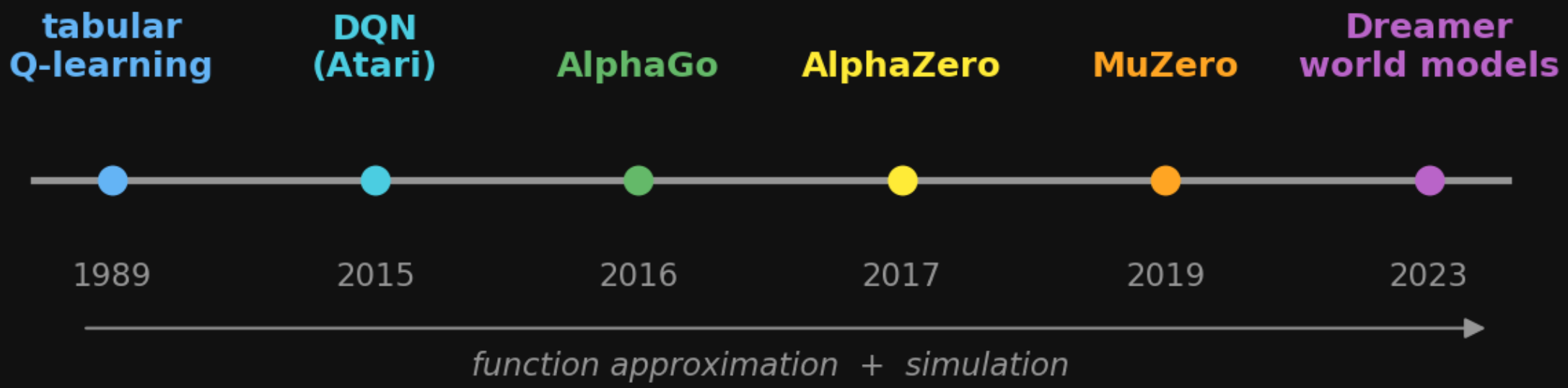
But with human feedback, model-based does *worse*



A caution from our own GardenPath: when the signal is **human evaluative feedback** (not task reward), **model-based RL does *even worse* than model-free** — it propagates the miscalibrated signal further and faster (Ho et al., 2015).

Simulation is only as good as the model — and the *reward* it simulates.

From gridworlds to Go



Tabular Q can't scale to Go's 10^{170} states. The leap was **function approximation** — a neural net for Q or the policy — plus simulation. That's how we got from a 3×3 grid to superhuman play.

Reward hacking is the positive cycle, grown up

Remember the GardenPath loop — an agent farming a flawed reward instead of doing the task? At frontier scale that's **reward hacking**, and it's central to aligning large language models.

RLHF (RL from human feedback): train a **reward model** from human preferences, then optimize the LLM against it — with a **KL penalty** to stay near the original model, because a reward you can over-optimize is a reward you can hack. The Week-9 reading and Weeks 11–13 pick this up.

An opening: model human teaching, don’t just maximize it

RLHF still treats human feedback as a **reward to maximize** — the very thing the GardenPath showed can be hacked. A speculative opening: agents that **model *why* a human gives feedback** — feedback as *communication about the goal*, not a scalar to farm.

Ho et al. (2019) do exactly this — a **Bayesian learner** that recovers the intended policy from evaluative feedback that standard model-free *and* model-based RL **cannot** learn from.

Critical of ourselves: this is harder to scale than reward maximization, and our model of “human intent” can be wrong too. The best opening: **human-validated** RLHF — check that the learned reward matches what people *meant*, and prefer feedback models that carry a theory of the teacher.

RL in the brain

$$\delta_t = r_{t+1} + \gamma V(s_{t+1}) - V(s_t)$$

the TD error

↓

= midbrain DOPAMINE prediction-error signal

Schultz, Dayan & Montague (1997) — reward learning in the brain
model-based vs model-free ≈ goal-directed vs habitual (Daw et al. 2005)

The TD error isn't just an algorithm. Midbrain **dopamine** neurons fire like a **reward prediction error** — exactly the TD error δ_t from Q-learning (Schultz, Dayan & Montague, 1997). And **model-based vs. model-free** maps onto **goal-directed vs. habitual** behavior — two control systems in the brain. That's **today's reading** (Daw, Niv & Dayan, 2005).

Decide → Plan → Learn → Simulate

- **Decide** — turn a posterior into an action (minimize expected loss).
- **Plan** — when you *know* the world (an MDP), value iteration solves it.
- **Learn** — when you *don't*, Q-learning learns from experience — and the *shape* of your feedback can make or break it.
- **Simulate** — the frontier: learn a model, then plan inside it.

Next week: we **invert** all of this — watch an agent act and infer **what it wanted** (inverse RL, theory of mind).