

Machines That Act

Reinforcement learning — learning from reward, not answers

Day 09 · Week 2

Unlocks 🎮 Game Master

1 Mission briefing

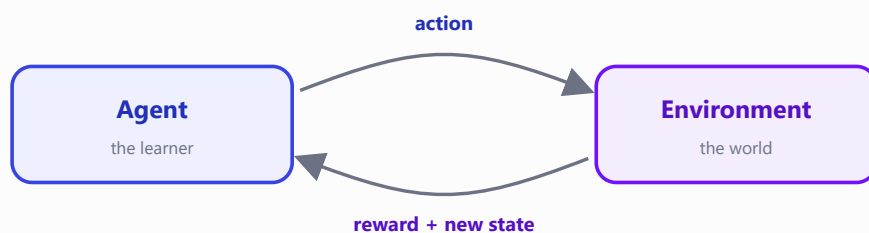
Earlier days explored both labeled examples and unlabeled patterns. Today's agent learns from its own actions, state transitions and rewards. It tries, fails, and updates its own instincts. You'll teach an agent to cross a frozen lake with a 64-number Q-table, then scale up to a neural network that balances a pole it has never seen.

Previously: you unlocked 🗣️ Language Lab — tokens, attention, and next-token prediction.

- ☐ Trace the **RL loop**: agent, action, environment, reward
- ☐ Build a **Q-table** — a 64-number instinct
- ☐ Balance exploring vs exploiting with **ϵ -greedy**
- ☐ Apply the one-line **Q-learning law**
- ☐ See when tables explode → deep Q-networks (**DQN**)
- ☐ Compare before / after CartPole videos

2 Field guide the concepts

The RL loop. An **agent** in an **environment** picks an **action**, receives a **reward** and a new **state**, and repeats. There is no answer key — only the score it is trying to grow.



The loop: the agent picks an action; the environment answers with a reward and a new state. Repeat thousands of times.

The Q-table is a tiny instinct. For FrozenLake there are 16 tiles $\times$ 4 moves = 64 numbers, each saying "how good is this move from here?" **ϵ -greedy** mostly takes the best-known move (exploit) but a fraction ϵ of the time tries something random (explore). And the whole thing learns from one line — the **Q-learning law**: $\text{new} = \text{old} + \alpha(\text{reward} + \gamma \cdot \text{best-future} - \text{old})$, nudging each estimate a step toward reality.

When tables explode → DQN. CartPole's state is continuous — infinite rows. So replace the table with a neural network that *predicts* Q-values. Same idea, far bigger brain.

 **Matrix Watch** — your Q-table is a 16 $\times$ 4 matrix, and the DQN that replaces it is just layers of matrix multiplies predicting the same Q-values. Same operation, bigger — and high-performance computing makes it fast.

A precise reading

RL learns from experience: states, actions, rewards and transitions. It does not require a correct-action label for every step. For a truly terminal transition, set the next-state value to zero in the Q target. A time-limit truncation is a separate case and can still bootstrap. The FrozenLake exercise uses deterministic, non-slippery transitions.

3 Lab missions check each off in the notebook

1 How far does pure luck get?

Run a purely random agent on FrozenLake many times.

Expect: it reach the gift only rarely — luck is a bad strategy.



2 Explore vs exploit (choose_action)

Pick a random move with probability ϵ , else `argmax(q[s])`.

Expect: the agent wander early and act decisively later.



3 Start bold, end wise (get_epsilon)

Decay ϵ from near 1 toward near 0 over training.

Expect: exploration high at first, then fading as it learns.



4 Train the agent (the crown jewel) 🏆

Apply the one-line Q-update across thousands of episodes.

Expect: the Q-table sharpen until it solves the lake.



5 How good did it get?

Evaluate the greedy policy's success rate.

Expect: near-100% on the non-slippery lake.



6 Read out the policy

Print the best action per state as arrows.

Expect: a little arrow map that walks straight to the goal.



7 The CartPole baseline

Compare an untrained agent to your trained DQN.

Expect: a few wobbly steps vs a pole balanced toward 500.



4 Cheat sheet Q-learning & Gym

```
gym.make("FrozenLake-v1",  
is_slippery=False)
```

Create the frozen-lake world.

```
s, info = env.reset(seed=0)
```

Start a fresh episode.

```
s, r, term, trunc, info = env.step(a)
```

Take action a ; get the 5-tuple back.

```
np.argmax(q[s])
```

Greedy move: the best-known action from state s .

```
q[s,a] +=  $\alpha$ *(r +  $\gamma$ *(0 if term else  
q[s2].max()) - q[s,a])
```

The Q-learning law, in one line.

```
DQN("MlpPolicy", env).learn(60_000)
```

A neural network replaces the table.

```
model.save("game_master_dqn.zip")
```

Freeze the trained agent to a file.

```
imageio.mimsave("run.gif", frames,  
fps=25)
```

Turn rendered frames into a replay GIF.

5 Unlock your Studio module

Game Master

Save your Q-table and your trained DQN. The unlock cell must write exactly:

```
ai_studio/models/game_master_qtable.npy
```

```
ai_studio/models/game_master_dqn.zip
```

Watch both agents replay live — the icy walk and the balancing pole:

```
$ python ai_studio/app.py
```

6 Mini-glossary

agent

The learner that chooses actions.

state

A snapshot of the situation the agent is in.

reward

The score signal the agent tries to maximize.

Q-value

The estimated long-term value of an action in a state.

DQN

A deep Q-network — a neural net replacing the Q-table.

environment

The world that answers with rewards and states.

action

A move the agent can make.

policy

The agent's strategy for choosing actions.

ϵ -greedy

Mostly exploit the best move, sometimes explore a random one.

7 Show & tell

1. Walk us through your Q-learning update line — what does each term actually do?
2. What ϵ did you use, and what happened when you set $\epsilon = 0$ (pure greedy)?
3. Show random vs trained CartPole. How many steps did each survive, and why the gap?