AN ILLUSTRATED, HANDS-ON GUIDE

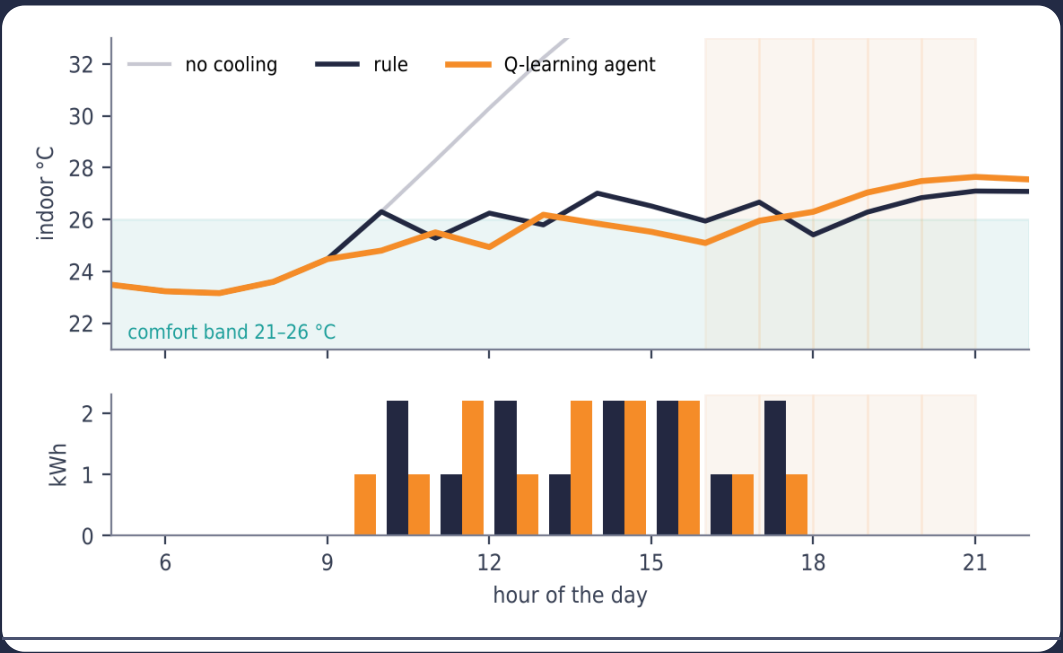
Reinforcement Learning

A friendly guide to AI that learns by trying, for buildings that act

Plain English, pictures first, a room you teach to cool itself,
five labs in plain Python, real numbers from the code, a glossary and a cheat sheet.

Seven short chapters · Five labs · Three challenges · Cheat sheet

Prompted and directed by Mahmoud A.
Written and illustrated by Claude (Anthropic)



Prepared for the 2026 CCS: Urban AI + Agent Workshop · Harbin Institute of Technology · November 2026

How to Use This Guide

Read it in order: every chapter ends with something to try.

This guide teaches reinforcement learning (RL) with one running example: an office room on a hot summer day. The room can cool itself a little, a lot, or not at all. Electricity costs three times more in the evening. People complain when the room is too warm. You will teach a controller to make the trade-off by itself.

YOU WILL BE ABLE TO

- Explain RL with the words agent, environment, state, action and reward.
- Run a bandit and see why an agent must sometimes explore.
- Write a small simulator of a room: a world model.
- Train a Q-learning agent in about 20 lines of numpy.
- Compare the agent with a rule an engineer would write.
- Say when RL is a good idea in a real building, and when it is not.

WHAT YOU NEED

- Python 3 with `numpy`. No internet, no GPU.
- The file `rl_lab.py` (or `rl_lab.ipynb`). Run it with `python rl_lab.py`.
- Paper and a pen for the exercises.

CONTENTS

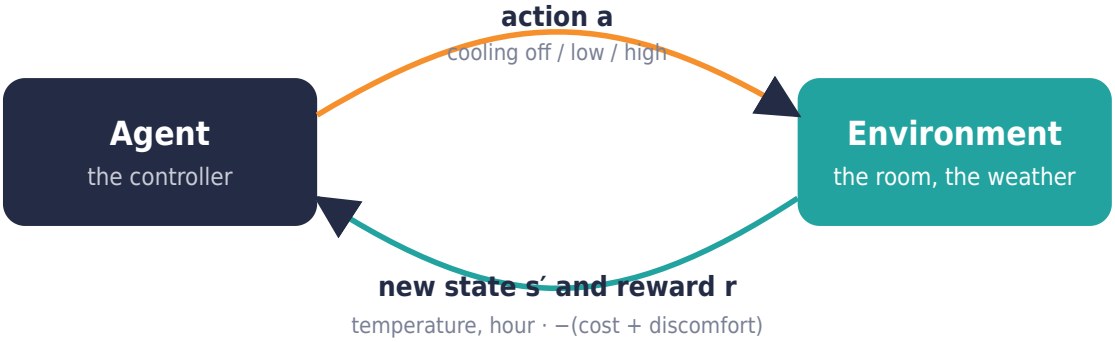
1. What is reinforcement learning?
2. The bandit: explore or exploit · Lab 1
3. States, actions, rewards · Lab 2
4. The rule to beat · Lab 3
5. Q-learning · Lab 4
6. Reading the agent's day · Lab 5
7. From one room to the city
8. Pitfalls, glossary, cheat sheet, reading

About the data. The room is a simulation with simple, visible equations in `rl_lab.py`. Every number in this guide comes from running that file. A real building is slower, noisier and less forgiving; Chapter 6 says what that changes.

1 What Is Reinforcement Learning?

Learning what to do from the results of doing it.

A child learns to ride a bicycle without a list of rules. The child tries, wobbles, falls, adjusts and tries again. Nobody gives the right answer for each moment; the child only feels whether things went well. **Reinforcement learning** is the same idea for a computer: an **agent** takes **actions** in an **environment**, receives a **reward**, and slowly learns which actions lead to more reward over time.



Word	Meaning	In our room
Agent	the one that decides	the cooling controller
Environment	everything the agent acts on	the room, the weather, the people
State s	what the agent sees now	indoor temperature and hour of the day
Action a	what the agent can do	cooling off, low or high
Reward r	one number: how good that was	−(electricity cost + discomfort)
Policy π	the agent’s rule: state → action	what we want to learn

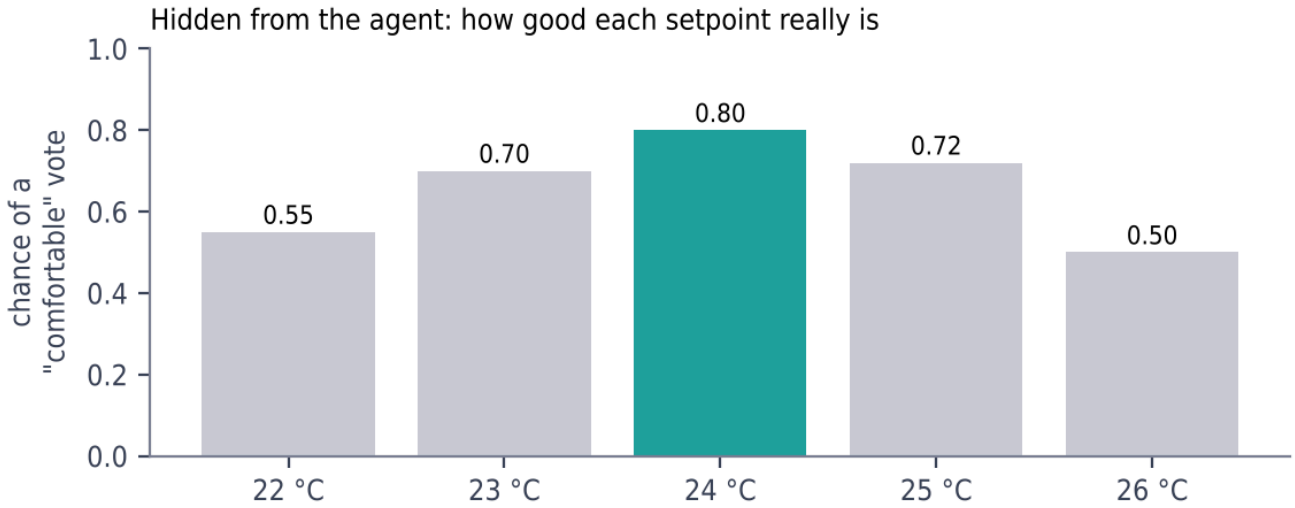
How RL differs from other machine learning. In supervised learning, every example comes with the right answer. In RL, nobody tells the agent the right action. The agent only gets a reward, often later, and must work out which earlier actions earned it.

> **IN ONE LINE**
>
> RL learns a policy, a rule from situations to actions, by trial, error and reward.

2 The Bandit: Explore or Exploit

Lab 1 · the simplest RL problem there is.

Start without time and without temperature. A building manager can choose one of five setpoints each day: 22, 23, 24, 25 or 26 °C. After each day, one occupant answers “comfortable” (reward 1) or not (reward 0). Each setpoint has a hidden chance of a comfortable vote. This is a **multi-armed bandit**, named after a row of slot machines.



The dilemma: **exploit** the setpoint that looks best so far, or **explore** another setpoint that might be better. The **ε-greedy** strategy explores at random with probability ε, and exploits otherwise.

```
def bandit(eps, steps=500, seed=0):
    r = np.random.default_rng(seed)
    n = np.zeros(5); q = np.zeros(5)                # tries and average reward per setpoint
    for t in range(steps):
        a = r.integers(5) if r.random() < eps else int(np.argmax(q))   # explore or exploit
        reward = float(r.random() < TRUE_OK[a])      # 1 = "comfortable" vote
        n[a] += 1; q[a] += (reward - q[a]) / n[a]    # update the running average
    return q, n
```

```
eps=0.0: mean comfortable votes per step = 0.550 | seed 0 picks setpoint 22 | tries [500   0   0
0   0]
eps=0.1: mean comfortable votes per step = 0.730 | seed 0 picks setpoint 24 | tries [ 15   6  43
417  19]
eps=0.3: mean comfortable votes per step = 0.743 | seed 0 picks setpoint 24 | tries [ 32  53 307
87  21]
```

With ε = 0 the agent never explores: it tries 22 °C first, keeps that choice forever, and averages 0.55. With a little exploration (ε = 0.1 or 0.3) the agent finds the setpoints around 24 °C and averages about 0.73–0.74.

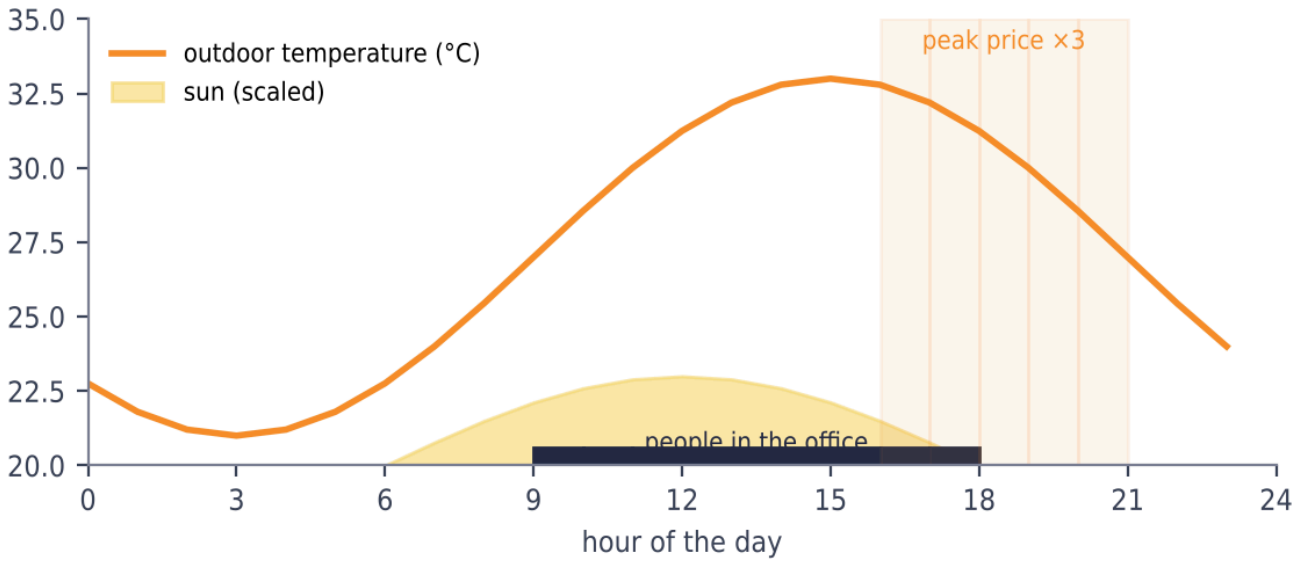
TRY THIS · EXERCISE 1

1. Run ε = 0.9. Why is the average worse, even though the agent knows the best setpoint?
2. Make 25 °C the best setpoint. How many days does the agent need to notice?
3. In a real building, what does one “exploration day” cost, and who pays for it?

3 States, Actions, Rewards

Lab 2 · a room that changes hour by hour.

A bandit has no memory: today's choice does not change tomorrow. A room does. If the room is not cooled at 13:00, the room is hotter at 14:00. Problems where actions change the next state are called **Markov decision processes** (MDPs).



The simulator is our **world model**: a few lines that say what happens next. Heat comes through the walls, from the sun and from people. Cooling removes 0, 1.5 or 3 degrees per hour. The reward adds two penalties: the cost of electricity, and the squared distance outside 21–26 °C while people are in the room.

```
def step(T_in, h, a):
    T_next = (T_in + 0.15 * (T_OUT[h] - T_in)   # heat through the walls
              + 1.2 * SUN[h] + 0.6 * OCC[h]       # sun and people
              - ACTIONS[a])                       # cooling: 0, 1.5 or 3 degrees
    cost = POWER[a] * PRICE[h]                    # kWh x price (x3 at 16:00-21:00)
    discomfort = OCC[h] * max(0, T_next - 26)**2 + OCC[h] * max(0, 21 - T_next)**2
    return T_next, cost, discomfort
```

```
hour 08: outdoor 25.4 C, sun 0.50, people 0, price 1
hour 12: outdoor 31.2 C, sun 1.00, people 1, price 1
hour 15: outdoor 33.0 C, sun 0.71, people 1, price 1
hour 18: outdoor 31.2 C, sun 0.00, people 0, price 3
no cooling all day: cost 0.0 | discomfort 580.0 | hottest indoor 37.3
```

WATCH OUT

The reward decides what the agent learns. The weight `W` says how many units of cost one unit of discomfort is worth. That is a value judgement, not a physics fact. Who should set the weight: the engineer, the building owner, or the occupants?

4

The Rule to Beat

Lab 3 · what most buildings do today.

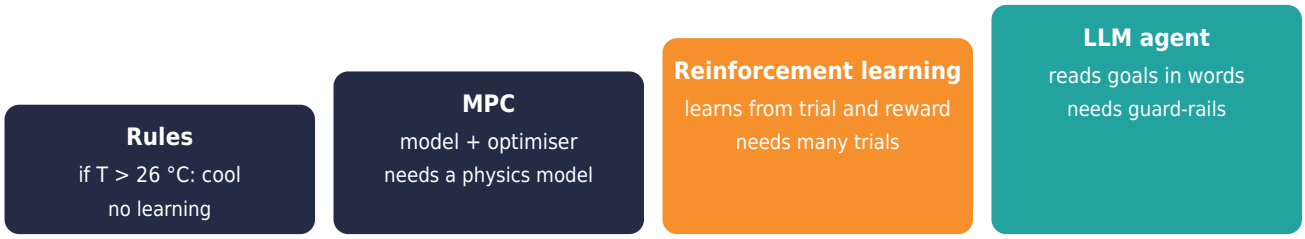
Most building management systems run fixed rules written by an engineer. Before training any agent, measure the rule. If the agent cannot beat the rule, the agent is not worth the risk.

```
def rule(T, h):
    if T > 26 and OCC[h]: return 2       # high cooling
    if T > 25 and OCC[h]: return 1       # low cooling
    return 0                             # off
```

```
rule-based thermostat: cost 20.4 | discomfort 1.9 | total 22.3
```

The rule is sensible and keeps discomfort low. But the rule only reacts: the rule waits until the room is already warm, and the rule does not know that electricity is three times more expensive after 16:00.

Four ways a building can decide



Rules react. Model predictive control (MPC) plans with a physics model. RL learns a plan from trials. LLM agents take goals in plain language. Real projects often mix them.

TRY THIS · EXERCISE 2

Improve the rule by hand: start cooling earlier, or cool harder before 16:00. Can you beat a total of 22.3? Write down how long it took you. The agent in the next lab will find a plan by itself.

5 Q-Learning

Lab 4 · a table of scores, improved one surprise at a time.

Q-learning keeps a table Q(s, a): the total future reward the agent expects if it takes action a in state s and acts well afterwards. To act, the agent looks up its row and picks the action with the largest score.

After each step, the agent compares what it expected with what happened, and moves the score a little toward the new information:

	off	low	high
24 °C, 10:00	-3.1	**-2.4**	-2.9
26 °C, 14:00	-9.8	-5.2	**-4.1**
27 °C, 17:00	-6.0	**-4.4**	-6.7

illustrative values · pick the largest in each row

$$Q(s, a) \leftarrow Q(s, a) + \alpha \, [\, r + \gamma \cdot \max_{a'} Q(s', a') - Q(s, a) \,]$$

α: learning rate (0.1) · γ: how much the future counts (0.95) · r + γ max Q(s′, a′): the new, better guess · the bracket: the surprise

```
Q = np.zeros((17, 24, 3))                # temperature bin x hour x action
for ep in range(6000):                   # 6000 simulated days
    T = r.uniform(24, 28)
    for h in range(24):
        s = tbin(T)
        a = r.integers(3) if r.random() < eps else int(np.argmax(Q[s, h]))
        T2, c, d = step(T, h, a)
        target = -(c + d) + gamma * Q[tbin(T2), h + 1].max()      # 0 after the last hour
        Q[s, h, a] += alpha * (target - Q[s, h, a])               # learn from the surprise
        T = T2
```

```
Q-learning agent:      cost 17.8 | discomfort 0.1 | total 17.9
Q-learning total over 5 training seeds: mean 17.9 | min 17.9 | max 17.9
```

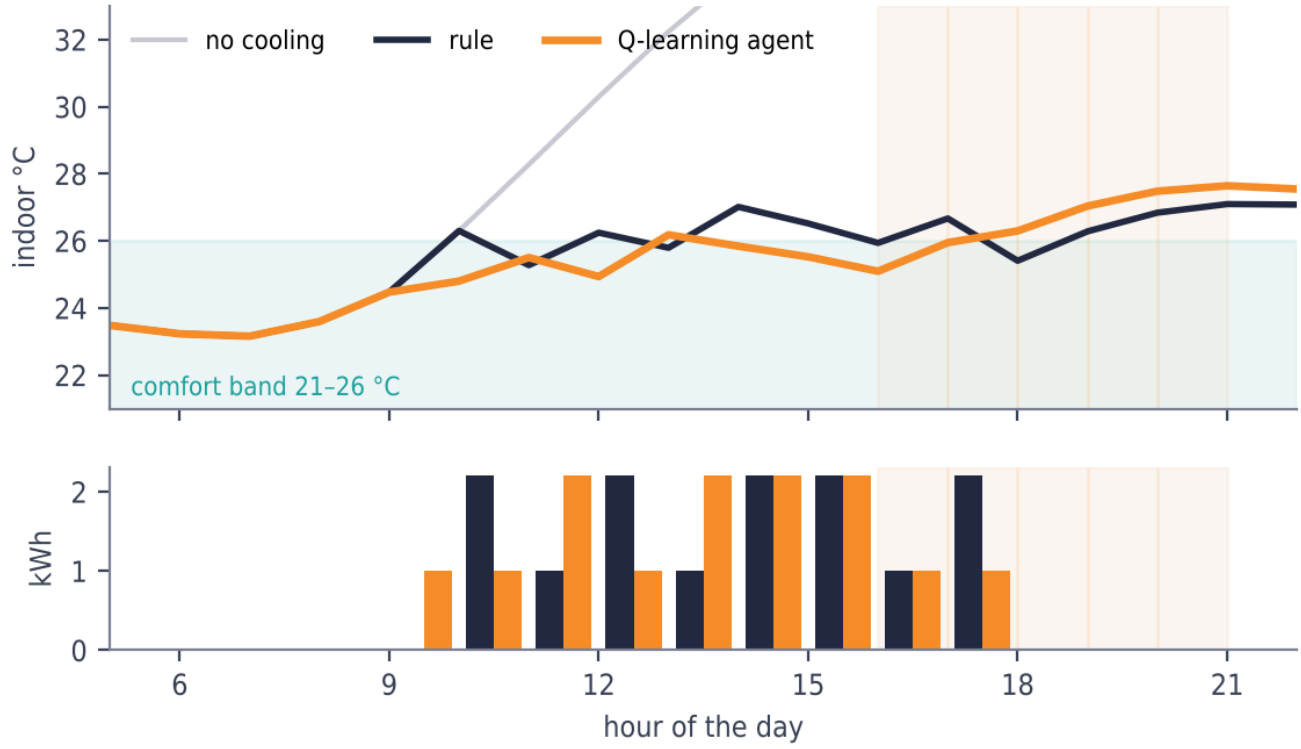


total = energy cost + discomfort for one day (lower is better). No cooling scores 580.

After 6,000 simulated days, the agent spends less on electricity than the rule (17.8 versus 20.4) and keeps the room almost always in the comfort band (discomfort 0.1 versus 1.9). The result is the same for five different random seeds.

L5 Reading the Agent's Day

What did the agent actually learn?




```
hour  price  rule-action rule-T   agent-action agent-T
06    1      0           23.2    0            23.2
07    1      0           23.6    0            23.6
08    1      0           24.5    0            24.5
09    1      0           26.3    1            24.8
10    1      2           25.3    1            25.5
11    1      1           26.2    2            24.9
12    1      2           25.8    1            26.2
13    1      1           27.0    2            25.8
14    1      2           26.5    2            25.5
15    1      2           25.9    2            25.1
16    3      1           26.7    1            26.0
17    3      2           25.4    1            26.3
18    3      0           26.3    0            27.0
19    3      0           26.8    0            27.5
20    3      0           27.1    0            27.6
energy 16:00-21:00 (during the peak):  rule 3.2 kWh, agent 2.0 kWh
```

Two habits the agent learned without being told. First, the agent starts cooling at 09:00, as soon as people arrive and before the room is warm; the rule waits until 10:00. Second, during the expensive evening hours the agent uses 2.0 kWh while the rule uses 3.2 kWh: the agent cools the room hard in the afternoon and coasts through the peak.

DISCUSS

1. Look at 18:00–20:00. The agent lets the room reach 27.6 °C. Why is that acceptable in this reward? Would it be acceptable in a hospital?
2. The agent learned from 6,000 days in a simulator. How many days would that be in a real building?
3. This is the evening peak of Part 3 of the course: if every building learns the same habit, what happens to the grid at 15:00?

6 From One Room to the City

More agents, slower actions, people in the loop.

Our agent controls one room with three actions. Real buildings and cities are harder in four ways.

Harder because	What researchers do
Trials are expensive. A bad action makes people uncomfortable or wastes energy.	Train in a simulator first (EnergyPlus, CityLearn, BOPTEST), then transfer carefully. Offline RL learns from logs of past operation.
States are large. Hundreds of sensors, not two numbers.	Replace the table with a neural network: deep Q-networks (Mnih et al., 2015), policy-gradient methods.
Many agents share one grid. Every building reacts to the same price.	Multi-agent RL and coordination; CityLearn was built to test this (Vázquez-Canteli et al., 2019).
Safety. Some actions must never happen.	Keep a rule-based fallback and limits around the agent, as DeepMind did for data-centre cooling (2018).

CHALLENGE A · A FAIRER REWARD

Change `W` to 0.2 and to 5. Retrain. Plot the agent's day for each value. Which value would you defend to the occupants, and which to the owner?

CHALLENGE B · TWO ROOMS, ONE METER

Simulate two rooms with one shared electricity price that rises when both rooms cool at the same hour. Train one agent per room. Do the agents learn to take turns?

CHALLENGE C · A WEATHER FORECAST

Add tomorrow's maximum temperature to the state. Does the agent pre-cool more on hot days?

Pitfalls and Glossary

Read this before you use RL in a real building.

Pitfalls checklist

- **Reward hacking.** The agent finds a loophole in the reward, such as never cooling when nobody is present, even if people arrive later to a hot room. Test the policy, not only the score.
- **Simulator gap.** A policy that is perfect in the simulator can fail in the real room. Calibrate the simulator with real data.
- **Too few trials.** Q-learning needed 6,000 days here. Real buildings give one day per day.
- **Exploration is not free.** Random actions in a real building annoy real people.
- **Changing environment.** Weather, tenants and prices change. A policy needs re-checking.
- **Always have a baseline.** Report the rule's score next to the agent's score.
- **Keep a human override.** Occupants and operators must be able to stop the agent.

Glossary

Agent
The decision maker.

Environment
Everything the agent acts on.

State
What the agent sees at one moment.

Action
A choice the agent can make.

Reward
One number that scores the last step.

Return
The sum of future rewards, discounted by γ.

Policy π
The rule from states to actions.

Q-value
Expected return of an action in a state.

ε-greedy
Explore at random with probability ε.

MDP
A problem where actions change the next state.

World model
A model that predicts the next state; here, the room simulator.

Offline RL
Learning a policy from logged data, without new trials.

One-Page Cheat Sheet

Everything in this guide on one page.

THE LOOP

state s → action a → reward r, next state s′ → repeat.

GOAL

maximise $\mathrm{E}[\, r_0 + \gamma\, r_1 + \gamma^2\, r_2 + \ldots \,]$

BANDIT UPDATE

$q(a) \leftarrow q(a) + (\, r - q(a)\,) \,/\, n(a)$

E-GREEDY

random action with probability ε, else argmax q.

Q-LEARNING

$Q(s,a) \leftarrow Q(s,a) + \alpha\, [\, r + \gamma \max Q(s',\cdot) - Q(s,a)\,]$

ACT

$a = \operatorname{argmax}_a Q(s,a)$

TYPICAL SETTINGS

α 0.05–0.2 · γ 0.9–0.99 · ε from 0.2 down to 0.02.

REWARD DESIGN

−(cost + W · discomfort). W is a value choice; write it down.

LAB RESULTS (SIMULATED)

Rule 22.3 · Q-learning 17.9 · peak energy 3.2 → 2.0 kWh.

BEFORE DEPLOYMENT

simulator → offline test → shadow mode → limited control with fallback.

BIG STATE SPACES

DQN, PPO, SAC in Stable-Baselines3 or RLlib.

BUILDING TESTBEDS

CityLearn · BOPTEST · Sinergym (EnergyPlus).

Further Reading

The papers behind each idea.

1. Sutton, R. S., & Barto, A. G. (2018). *Reinforcement Learning: An Introduction* (2nd ed.). MIT Press.
2. Watkins, C. J. C. H., & Dayan, P. (1992). Q-learning. *Machine Learning*, 8, 279–292.
3. Mnih, V., et al. (2015). Human-level control through deep reinforcement learning. *Nature*, 518, 529–533.
4. Silver, D., et al. (2016). Mastering the game of Go with deep neural networks and tree search. *Nature*, 529, 484–489.
5. Ha, D., & Schmidhuber, J. (2018). World models. arXiv:1803.10122.
6. Vázquez-Canteli, J. R., & Nagy, Z. (2019). Reinforcement learning for demand response: a review of algorithms and modeling techniques. *Applied Energy*, 235, 1072–1089.
7. Vázquez-Canteli, J. R., Kämpf, J., Henze, G., & Nagy, Z. (2019). CityLearn v1.0: an OpenAI Gym environment for demand response with deep reinforcement learning. *BuildSys*.
8. Wang, Z., & Hong, T. (2020). Reinforcement learning for building controls: the opportunities and challenges. *Applied Energy*, 269, 115036.
9. Blum, D., et al. (2021). Building optimization testing framework (BOPTEST) for simulation-based benchmarking of control strategies in buildings. *Journal of Building Performance Simulation*, 14(5), 586–610.
10. Evans, R., & Gao, J. (2016). DeepMind AI reduces Google data centre cooling bill by 40%. DeepMind blog.
11. Drgoňa, J., et al. (2020). All you need to know about model predictive control for buildings. *Annual Reviews in Control*, 50, 190–232.

Files · `rl_lab.py` (all labs) · `rl_lab.ipynb` (the same code as a notebook)

Part of the Fundamentals 101 series · mahmoud.work