第一題作業

有三個版本，以 pA_decay版最好

一、終止條件設計

pA與 pA_ Decay 的終止條件設計：

- **策略穩定 (Policy Stability)**：
  - 條件：`abs(Profit_new - Profit_old) < 1e-4`
  - 意義：Agent 找到的最佳解（利潤）已經不再劇烈變化。
- **數值穩定 (Value Stability)**：
  - 條件：`Avg_Delta_Q < 1e-3` (平均 Q 值變動量)
  - 意義：Q-table 內部的數值已經不再變動。**這是「固定參數」最難通過的一關。**
- **持久性 (Persistence)**：
  - 上述兩個條件必須**連續維持 50 個回合 (Episodes)** 才算數。
  - 這是為了防止運氣好剛好一回合沒動。

pA_patience的終止條件設計：

1. 先跑一段暖機期（min_episodes_run = 3000）

→ 為了避免一開始亂探索就提前停止，系統至少會跑 3000 回合 再開始看「有沒有進步」。

2. 耐心值（patience_limit = 3000）

→系統允許 Agent 最多 3000 次沒有進步。

3. 如何判斷有沒有進步

每一回合會計算「目前 Q-table 建議的策略」的利潤：

- 如果新策略利潤 比歷史最佳還好 →

➜ 代表找到更好的解

➜ 更新 global_best_profit

➜ 耐心計數器歸 0

- 如果沒有變好 →

  ➜ 耐心計數器 +1

4. 什麼時候停止訓練？

這兩個條件都成立才會停：

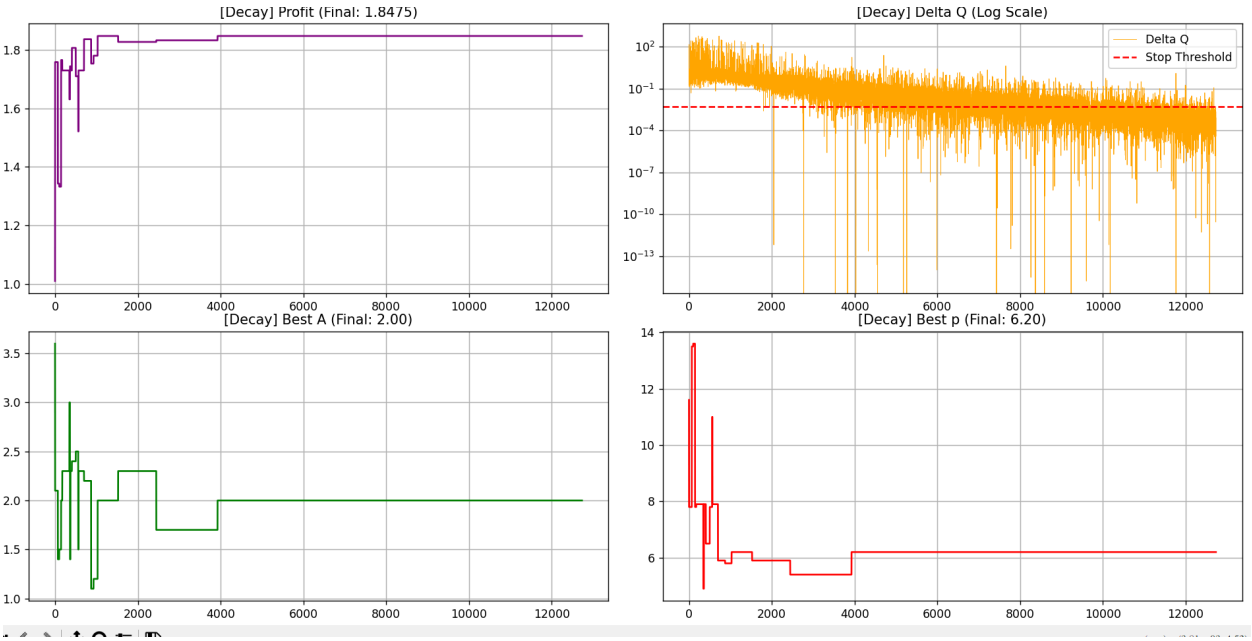
1. 已經過了暖機期（> 3000 回合）
2. 耐心值耗盡（3000 回合都沒有進步）

二、三個版本的比較表

比較項目	pA.py (基準)	pA_patience.py (折衷)	pA_Decay.py (最佳解)
參數設定	Fixed $\alpha = 0.1, \epsilon = 0.1$	Fixed $\alpha = 0.1, \epsilon = 0.1$	Decay $\alpha \to 0.001, \epsilon \to 0.01$
終止條件	雙重：Profit穩定 + ΔQ穩定	**Patience**：Profit 長期無進步	雙重：Profit穩定 + ΔQ穩定
ΔQ 行為	在高位震盪 (Noise Floor)	在高位震盪 (無視之)	**平滑下降並趨近於 0**
收斂性質	難以達成嚴格收斂	**假性收斂** (停止但不靜止)	**真收斂** (數值與策略皆靜止)
主要風險	永不停止或需門檻極寬	Patience 設太短會錯失最佳解	衰減太快可能導致學習不完全
結論定位	理論上嚴謹但實務不可行	固定參數下的最佳權宜之計	**最穩定、科學的解決方案**

三、收斂結果

最終結果: A=2.000, p=6.000, π=1.8400 (A 與P 正負0.3， π 正負0.00XX)



四、程式碼(decay 最佳版)

```
import numpy as np
import random
from tqdm import tqdm

# ==========================================
# 1. 核心參數 (Experimental Group: Decay)
# ==========================================
n_episodes = 15000
n_steps = 100
gamma = 0.9

# --- 實驗變數: 雙重衰減 ---
# Epsilon: 1.0 → 0.01
epsilon_start = 1.0
epsilon_end = 0.01
epsilon_decay = 0.999  # 控制衰減速度
```

```
# Alpha: 0.1 → 0.001 (起始值刻意設為 0.1 與 Fixed 組一致，但允許變小)
alpha_start = 0.999
alpha_end = 0.001
alpha_decay = 0.9995    # Alpha 通常比 Epsilon 衰減得慢一點或同步

# 初始化
epsilon = epsilon_start
alpha = alpha_start

# --- 經濟與環境參數 (完全一致) ---
beta = 10
eps_A = 0.5
eps_p = -1.5
c = 2
penalty = -1_000_000

A_min, A_max = 0.0, 10.0
p_min, p_max = 0.1, 20.0
n_A = 101
n_p = 200

A_values = np.linspace(A_min, A_max, n_A)
p_values = np.linspace(p_min, p_max, n_p)

actions = [(-1, -1), (-1, 0), (-1, 1), (0, -1), (0, 0), (0, 1), (1, -1), (1, 0), (1, 1)]
num_actions = len(actions)

Q_table = np.zeros((n_A, n_p, num_actions))

# --- 終止條件 (完全一致) ---
eps_pi_threshold = 1e-4
stop_delta_q_threshold = 5e-3 # 同樣的門檻
stable_episodes_req = 50
min_episodes_run = 3000

# 數據紀錄
```

```
history_profit = []
history_delta_q = []
history_best_A = []
history_best_p = []
history_params = [] # 額外紀錄參數變化

def get_reward(A, p):
    if A < 0 or p <= 0: return penalty
    Q = beta * (A ** eps_A) * (p ** eps_p)
    pi = (p - c) * Q - A
    return pi

# ==========================================
# 2. 訓練迴圈
# ==========================================
stable_count = 0
prev_best_profit = -np.inf

progress_bar = tqdm(range(n_episodes), desc="[Decay] Training")

for episode in progress_bar:

    A_idx = random.randint(0, n_A - 1)
    p_idx = random.randint(0, n_p - 1)

    delta_sum = 0.0
    update_count = 0

    for step in range(n_steps):
        # 策略: 變動 Epsilon
        if random.uniform(0, 1) < epsilon:
            action_index = random.randint(0, num_actions - 1)
        else:
            action_index = np.argmax(Q_table[A_idx, p_idx])

        dA, dp = actions[action_index]
```

```
        new_A_idx = max(0, min(n_A - 1, A_idx + dA))
        new_p_idx = max(0, min(n_p - 1, p_idx + dp))

        reward = get_reward(A_values[new_A_idx], p_values[new_p_idx])

        # 更新: 變動 Alpha (使用當前 alpha)
        old_q = Q_table[A_idx, p_idx, action_index]
        best_next = np.max(Q_table[new_A_idx, new_p_idx])
        td_target = reward + gamma * best_next

        Q_table[A_idx, p_idx, action_index] += alpha * (td_target - old_q)

        delta_sum += abs(Q_table[A_idx, p_idx, action_index] - old_q)
        update_count += 1
        A_idx, p_idx = new_A_idx, new_p_idx

    # --- 參數衰減執行 ---
    if epsilon > epsilon_end: epsilon *= epsilon_decay
    if alpha > alpha_end: alpha *= alpha_decay

    # --- 紀錄與收斂檢查 ---
    avg_delta_q = delta_sum / max(1, update_count)

    best_idx_flat = np.argmax(np.max(Q_table, axis=2))
    best_A_i, best_p_i = np.unravel_index(best_idx_flat, (n_A, n_p))
    pi_best = get_reward(A_values[best_A_i], p_values[best_p_i])

    history_profit.append(pi_best)
    history_delta_q.append(avg_delta_q)
    history_best_A.append(A_values[best_A_i])
    history_best_p.append(p_values[best_p_i])
    history_params.append(alpha)

    progress_bar.set_description(f"π={pi_best:.2f}, ΔQ={avg_delta_q:.1e}, α={al
pha:.3f}")
```

```
    if episode > min_episodes_run:
        profit_stable = abs(pi_best - prev_best_profit) < eps_pi_threshold
        q_stable = avg_delta_q < stop_delta_q_threshold

        if profit_stable and q_stable:
            stable_count += 1
        else:
            stable_count = 0

        if stable_count >= stable_episodes_req:
            print(f"\n*** [Decay] 收斂達成於 Episode {episode} ***")
            print(f"原因: 利潤穩定 且 ΔQ({avg_delta_q:.5f}) < {stop_delta_q_threshol
d}")
            break

    prev_best_profit = pi_best

print(f"最終結果: A={history_best_A[-1]:.3f}, p={history_best_p[-1]:.3f}, π={hist
ory_profit[-1]:.4f}")



# ==========================================
# 3. 視覺化與繪圖 (Visualization)
# ==========================================
import matplotlib.pyplot as plt

print("\n繪製 [Decay] 結果...")
plt.figure(figsize=(15, 10))

# 1. Profit
plt.subplot(2, 2, 1)
plt.plot(history_profit, color='purple')
plt.title(f'[Decay] Profit (Final: {history_profit[-1]:.4f})')
plt.grid(True)

# 2. Delta Q (Log Scale)
```

```
plt.subplot(2, 2, 2)
plt.plot(history_delta_q, color='orange', linewidth=0.5, label='Delta Q')
# 畫出 Alpha 的下降軌跡來對照 (用右軸或縮放)
plt.yscale('log')
plt.title('[Decay] Delta Q (Log Scale)')
plt.axhline(y=stop_delta_q_threshold, color='r', linestyle='--', label='Stop Thre
shold')
plt.legend()
plt.grid(True)

# 3. A Trajectory
plt.subplot(2, 2, 3)
plt.plot(history_best_A, color='green')
plt.title(f'[Decay] Best A (Final: {history_best_A[-1]:.2f})')
plt.grid(True)

# 4. p Trajectory
plt.subplot(2, 2, 4)
plt.plot(history_best_p, color='red')
plt.title(f'[Decay] Best p (Final: {history_best_p[-1]:.2f})')
plt.grid(True)

plt.tight_layout()
plt.show()
```