M1354020 資工碩一 林昀佑

一、作業目標

本次作業目標為透過 Q-Learning 強化學習方法，設計一個智能體在 8×10 的網格環境中學習最佳行動策略。作業重點包含以下項目：

1. 更改網格大小為 8（row）× 10（column）
2. 修改起點與終點位置
3. 隨機產生每格節點的 discount factor（四個方向）
4. 紀錄並可視化最佳路徑策略
5. 輸出完整 Q-table（非僅起點）
6. 比較不同 epsilon 值對執行時間的影響

二、實作內容

(1) 網格大小調整與起點/終點設定

- 原程式使用 4×6 的網格，現已擴充為 8×10。
- 起點與終點根據自定義需求進行調整（例如起點為 (0,0)，終點為 (7,9)）。

(2) 隨機 discount factor 產生

- 利用隨機生成器為每個格子建立四個方向的 discount factor，數值介於 0～1，精確至小數點後三位。
- 資料儲存於 discount_factor.csv，格式為每個格子的座標與其對應的 discount factor。

(3) 資料結構與效能優化

項目	說明	效益
使用 NumPy 陣列 儲存 Q-table	從字典改為 NumPy 陣列	提升 10–100 倍查詢速度
預先載入 discount factor	減少重複 CSV 存取	降低 I/O 延遲
加入進度條與終止控制	避免無限迴圈	增加可觀察性與穩定性
精簡 epsilon 測試點	減少測試時間	保持覆蓋率同時加快速度
支援 numba JIT	選擇性加速	進一步提升運算效率

(4) ε (epsilon) 測試與分析

- 設定 ε = [0.0, 0.25, 0.5, 0.75, 1.0] 進行比較。

- 每組 ε 測試 1000 次 episode，統計其平均訓練時間。
- 以圖表方式呈現 epsilon 變化對訓練時間的影響（見 epsilon_performance_optimized.png）。

(5) Q-table 與最佳路徑視覺化

- 使用 numpy_to_txt_grid.py 產生可讀的最佳路徑圖與完整 Q-table。
- 輸出結果包含：
    - q_table.txt：文字版完整 Q-table
    - reward_plot.png：reward 收斂曲線
    - q_table_optimized.npy 與 full_q_table_optimized.npy

三、心得反思與困難

在進行本次 Q-Learning 強化學習作業的過程中，我除了更深入理解 Q-Learning 演算法的運作邏輯外，也實際經歷了多個問題與挑戰，從中獲得不少實務經驗。

1. 網格擴充與起訖點調整問題

   剛開始嘗試將原程式中的 4×6 網格擴充至 8×10 時，因為許多地方寫死了 row 和 column 數值（例如初始化 Q-table、discount factor 輸入等），導致程式執行時出現 index error。我必須逐步檢查所有用到座標的程式段，並使用變數代替硬編碼數字，確保整個邏輯能隨網格大小自動調整。

2. discount factor 隨機產生與讀取效能問題

   在生成隨機 discount factor 時，最初版本是每次 Q 值更新時都從 CSV 中讀取對應格子的 discount factor。這種設計在大規模訓練（數千次 episode）下，造成明顯的延遲與效能瓶頸。後來我改寫成一次性載入整份 CSV 檔，並以 NumPy 陣列儲存，查詢時直接用索引取值，效能因此大幅提升。

3. 無限迴圈與訓練終止問題

   由於環境中可能存在 agent 無法到達終點的情況（例如一開始 ε 值過大），導致 agent 在某些 episode 中無限嘗試而無法終止。為了解決這個問題，我在每個 episode 加入最大步數的限制（例如 100 步），若超過則強制結束該輪，避免卡死。同時，也加入了 tqdm 進度條來觀察訓練進度與追蹤是否有異常行為。

4. epsilon 參數測試時執行時間過長

   原本設定 11 種 ε 值，每種跑 10000 次 episode，總訓練時間超過數十小時(可能是我電腦問題)，實務上難以接受。所以經過多次測試與結果比對後，我將 ε 減少為 5 個代表值（0.0, 0.25, 0.5, 0.75, 1.0），並將 episode 數降為 1000。這樣可在保有足夠比較意義的前提下，大幅減少執行時間。

總結

本次作業從程式調整到效能優化，都讓我深刻體會到：

1. 強化學習模型雖理論簡單，但實作層面充滿細節。
2. 實驗設計（如 epsilon 測試）、效能優化（如 NumPy、預載入資料）、可視化與結果輸出都是不可忽視的關鍵。
3. 除了演算法邏輯正確外，程式碼可維護性、可延展性、資源效率都是實務應用的重點。

這些挑戰與解決過程讓我獲得許多寶貴經驗，也讓我更加有信心處理未來更大型與複雜的強化學習任務。