計算型智慧 作業一

110403518 資工 4B 林晉宇

一、程式介面說明

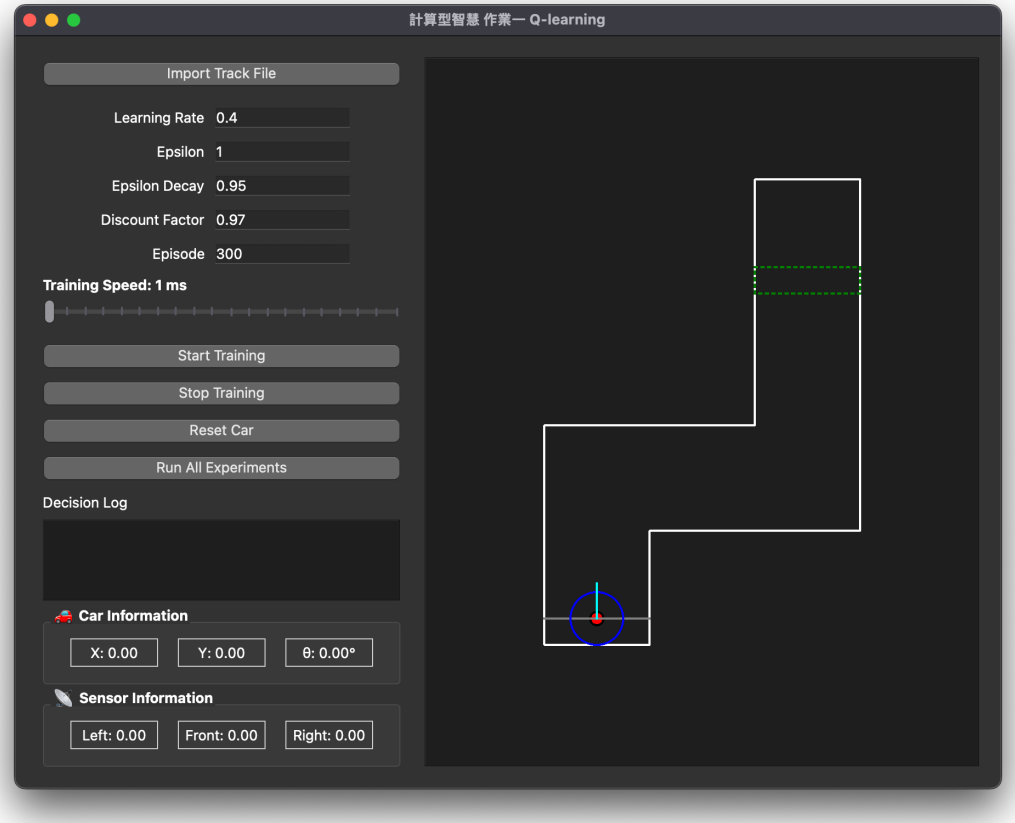
本次作業以 Python 的 PyQT5 開發互動式的 GUI 介面（圖一），整體介面分為左右兩區，操作方式為先匯入軌道座標檔，接著調整參數以及訓練速度，在按下「Start Training」。

左側功能區：

1. **匯入軌道**：點選「Import Track File」可讀入軌道.txt 檔案，並顯於右方顯示軌道
2. **超參設定**：包含學習率（Learning Rate）、ε（探索率）、ε 衰減、折扣因子（Discount Factor）與訓練次數（Episode）。
3. **訓練速度**：最低可設為 1ms 以達到快速訓練效果（加速動畫）。
4. **控制按鈕**：提供開始、暫停訓練（Stop Training）與重置車輛（Reset Car）。
5. **Decision Log**：記錄訓練過程中的決策狀態。
6. **車輛資訊**：即時顯示車輛的座標 (x, y) 及角度 θ。
7. **感測器資訊**：顯示三個方向（左前右）對邊界的偵測距離。

右側畫布區：

1. 顯示匯入的軌道圖形，包含白色邊界線、紅色起點、綠色終點區域。
2. 車輛以圓形表示，朝向以藍色線段表示。
3. 訓練過程中，車輛行經路徑會留下白色軌跡。

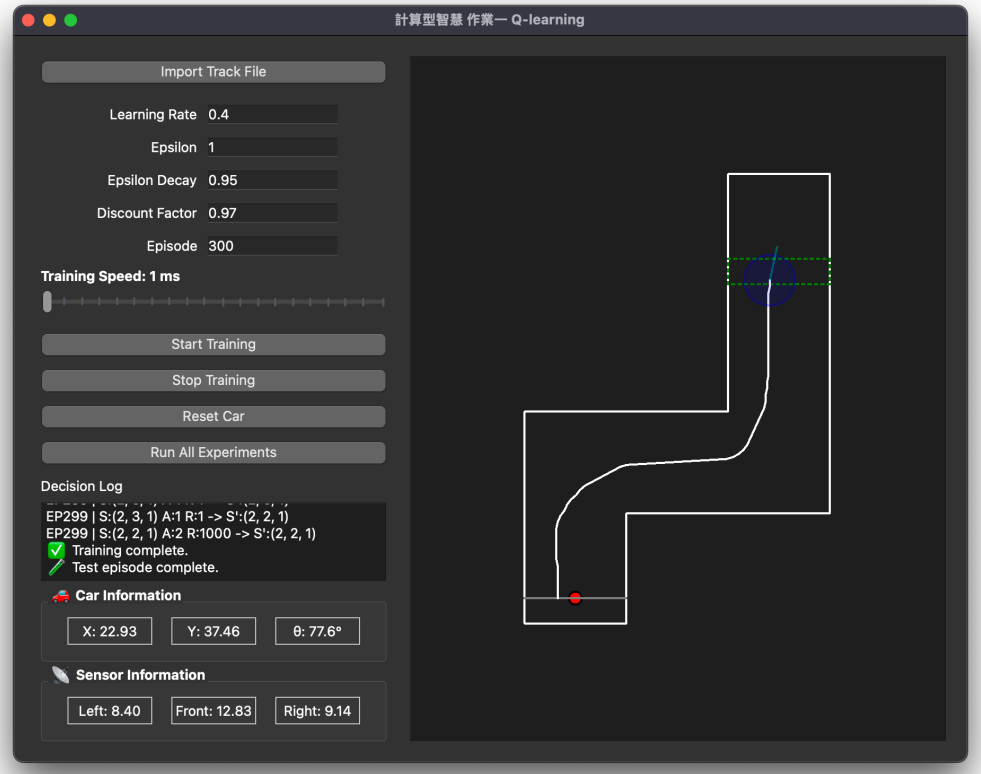


圖一、GUI 介面展示

二、實驗結果

經過大量實驗後，最終發現趨於穩定的設定如下，基於這個設定，測試十次的失敗次數大概是一到兩次，下圖為移動軌跡截圖（圖二）：

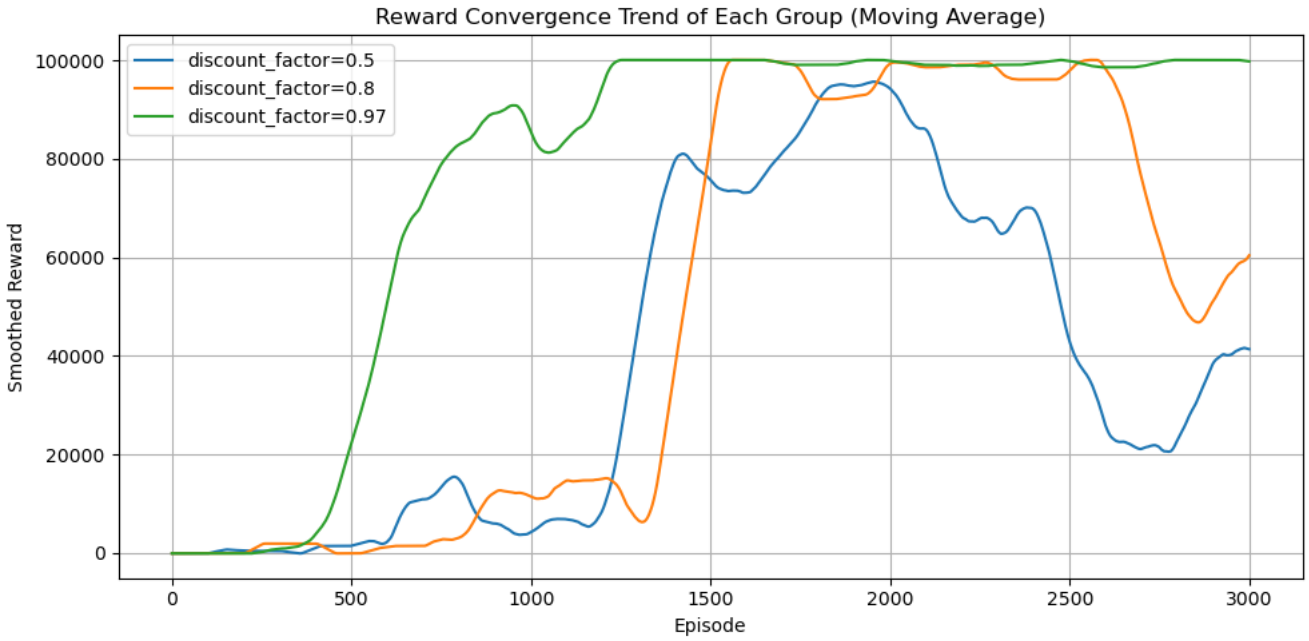
1. **學習率（Learning Rate α)**: 0.4
2. **探索率（Epsilon ε)**: 1
3. **探索率下降（Epsilon Decay）**: 0.95 (per episode)
4. **折扣因子（Discount Factor）**: 0.97
5. **訓練次數（Episode）**: 300
6. **狀態（State）**: 每個 sensor 值離散為 6 個區間（0～5），以每 5 單位為一格，最大限制為 5。所以總共會有 216 種 state。
7. **動作（Action）**: [-40, 0, 40]，共 3 種。



圖二、移動軌跡截圖（測試階段）

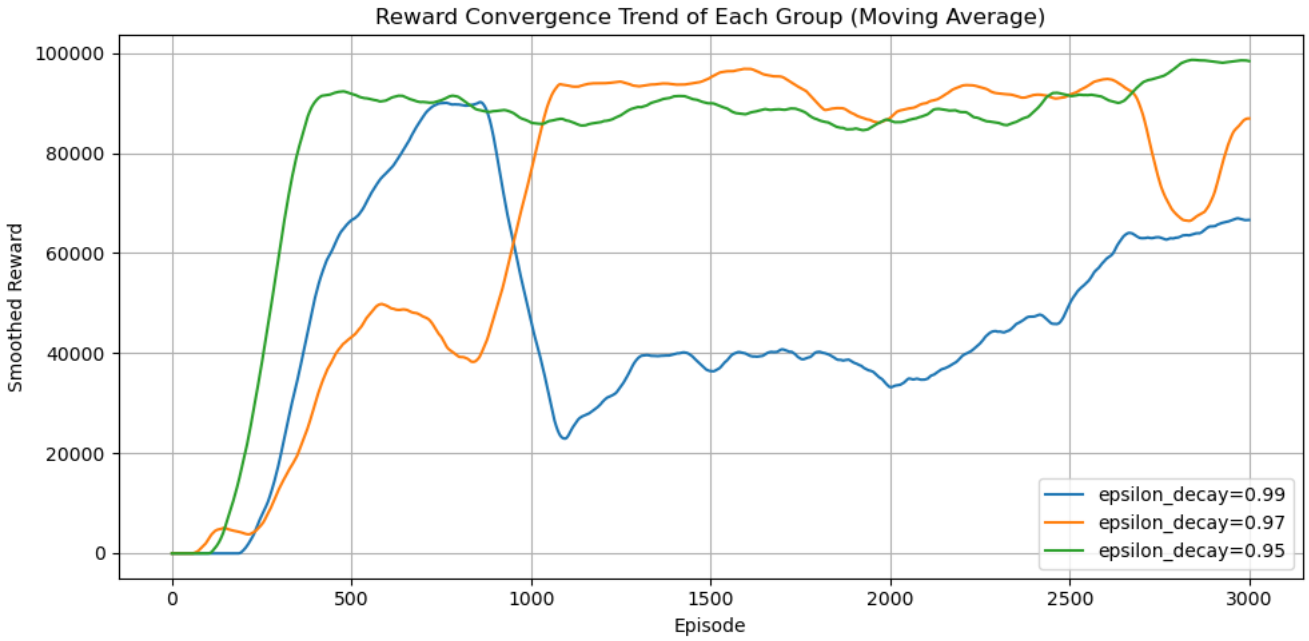
此外，我也針對個別參數調整做實驗，同時附上實驗比較圖（為了讓實驗圖易讀，有放大抵達終點獎勵至 100000，並提高訓練次數），在「分析與探討」階段會針對這些實驗圖做分析及想法闡述：

1. **折扣因子（Discount Factor）**：將折扣因子分別設為 0.5、0.8、0.97，其餘參數依照如上設定。如圖所示（圖三），可以看到越大的折扣因子（0.97），會比較快收斂，而比較小的則尚未收斂（0.5）。



圖三、Discount Factor 比較圖

2. **探索率下降（Epsilon Decay）**：將其設定為 0.99、0.97、0.95，其餘參數依照如上設定。如圖所示（圖四），可以看到下降率比較小的（0.95），反而比較快收斂，而比較高的（0.99）在前期不容易收斂。



圖四、Epsilon Decay 比較圖

三、判斷規則

需要判斷的部分主要分為以下幾點：

1. **Episode 結束方式**：當車子撞到邊界或是抵達終點才會結束。
2. **策略（Policy）設計**：採用 ε-greedy 的方式，在訓練初期以較高機率隨機選擇動作進行探索（ε=1），隨著訓練次數逐步降低探索率（ε decay），轉向利用學得的最佳策略（Q table）。
3. **三個感測器距離與動作（state & action）**：三個感測器的數值會以每 5 個單位切割，並分成至多 6 等分，總共會有 216 種狀態。動作的話以 40 度切割，總共有三種動作。
4. $$state = \left(b_{\text{left}}, b_{\text{front}}, b_{\text{right}}\right),\ b = min\left(\left\lfloor \frac{s}{5} \right\rfloor, 5\right),\ b \in \{0,1,2,3,4,5\}$$
$$action = \{-40°, 0°, +40°\}$$
5. **獎勵（Reward）設計**：如果車子抵達終點：+1000；碰到邊界：-100；其餘每走一步：+1。

四、分析與探討

針對這次作業，我主要有以下幾點觀察與思考：

1. **狀態（State）與動作（Action）的設計對學習效率影響**：實驗初期，我將狀態和動作切分的非常細緻，結果發現車子（agent）很難再合理的時間內收斂。後來將狀態和動作的數量減少，狀態剩下 125 種，動作以 40 度切割，也就是只有前、左、右三種方向選擇，反而可以學的更好，更快收斂。
2. **探索率下降（ε decay）對於收斂速度的影響**：在初期實驗中，我傾向將探索率的下降速度（epsilon decay）設為較大（例如 0.995），以鼓勵車輛能在訓練早期盡可能探索不同路徑。然而，隨著我將狀態與動作的數量簡化後發現：車輛其實在早期便有相當高的機率能夠探索到終點的路徑。而由於探索率下降得太慢，導致 agent 在後續仍然偏好隨機探索，而非鞏固既有學習成果。因此，即使車輛曾經走過通往終點的路徑，也可能未將該策略保留下來，而是繼續嘗試其他可能性。

   從圖四可觀察到，當 ε decay 為 0.99 時，雖然 reward 曾在初期上升，但整體收斂速度相對較慢；而當 decay 改為 0.95 時，車輛在探索到終點路徑後，能更快地收斂至穩定策略，約於第 500 回合便已趨於穩定。所以我認為在狀態與動作空間相對簡單的情況下，過慢的探索率下降反而會拖慢學習效率。
3. **折扣因子（Discount Factor）設計意義**：此參數可以控制車子對未來獎勵的重視程度，值越小，代表越重視近期的動作回報，越大，代表越重視遠期的回報。在這次作業中，我認為將這個值設定大一點比較合理，當車子在隨機探索時走到終點，較大的的折扣因子，可以放大抵達終點的獎勵。

   如圖三所示， 設定 0.97 時，曲線明顯比 0.8 或 0.5 更早收斂，並維持穩定高報酬；反之，較低的組別即使最終也能學會策略，但收斂速度較慢，且在學習過程中容易出現波動與不穩定的行為。
4. **隨機性仍佔主導因素**：即便在同樣的參數組合下，重複訓練多次，策略學的好壞懸殊。我認為 Q-learning 學的好壞，很大一部分原因還是跟一開始 ε 探索的方向是否正確，接著再經過大量的 trial and error 才能獲得好的結果。