摘要

本專案旨在探索並實踐強化學習（Reinforcement Learning, RL）於遊戲 AI 開發的應用。我們以打造一個類瑪莉歐（Mario-like）的平台遊戲為目標，訓練 AI 代理（Agent）學習自主探索地圖、躲避敵人並抵達終點。專案初期，團隊深入研究了 Q-Learning 等基礎演算法；隨後，我們將理論與實作結合，探討並應用了深度 Q 網路（Deep Q-Network, DQN）及其改良演算法（Double DQN, Dueling DQN）。過程中，我們使用 Unity 引擎及 ML-Agents 工具包搭建訓練環境，並挑戰結合 GPT-2 進行程序化地圖生成。本報告將詳述專案從理論學習到技術實作的全過程，展示團隊在強化學習領域的學習成果、所遇挑戰與解決方案，並對未來的研究方向提出展望。

專案簡介

研究動機與目標

隨著人工智慧技術的飛速發展，強化學習已在遊戲、機器人控制、資源調度等領域展現出巨大潛力。遊戲不僅是娛樂媒介，更是測試與驗證智慧演算法的絕佳沙盒。本團隊對強化學習如何賦予遊戲角色「智慧」充滿好奇，期望透過親手實作，將抽象的演算法理論轉化為可見的、動態的成果。

專案總目標：

透過強化學習自主探索地圖、躲避敵人並順利到達終點，完成一個類瑪莉歐遊戲，以此作為強化學習理論與實作的應用展示。

研究方法與理論基礎

本專案的核心技術基於強化學習，並逐步從傳統表格方法演進至深度學習模型。

強化學習 (Reinforcement Learning) 概論

強化學習是機器學習的一個分支，其核心思想是讓一個 代理 (Agent) 在一個 環境 (Environment) 中進行學習。Agent 透過 試誤學習 (Trial and Error)，根據執行的 動作 (Action) 所收到的 獎勵 (Reward) 或懲罰，來學習如何選擇能最大化長期累積獎勵的 策略 (Policy)。

Q-Learning 演算法

Q-Learning 是一種經典的、無模型（Model-Free）的強化學習演算法。它旨在學習一個名為 Q 函數（Q-function）的動作價值函數，用於評估在特定 狀態 (State) s 下，執行某個 動作 a 的優劣。其學習目標是找到最佳的 Q 函數 Q∗(s,a)。

Q-Learning 的核心在於 Q 值的迭代更新公式：

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

其中：

- Q(s,a)：在狀態 s 執行動作 a 的 Q 值。
- α (Alpha)：學習率（Learning Rate），控制每次更新的步長。
- r (Reward)：執行動作後獲得的立即獎勵。
- γ (Gamma)：折扣因子（Discount Factor），權衡立即獎勵與未來獎勵的重要性。
- s′：執行動作後進入的下一個狀態。
- $\max_{a'} Q(s',a')$：在下一個狀態 s′ 所有可能動作中，能獲得的最大 Q 值。

為了平衡 探索（Exploration，嘗試未知動作以發現更優策略） 與 利用（Exploitation，選擇當前已知最優動作以獲取最大獎勵），我們採用了 ε-greedy

策略。該策略以 ε 的機率隨機選擇動作（探索），以 1-ε 的機率選擇當前 Q 值最高的動作（利用）。

深度 Q 網路 (Deep Q-Network, DQN)

傳統 Q-Learning 使用表格（Q-Table）儲存 Q 值，當狀態空間或動作空間變得龐大或連續時（如遊戲畫面作為輸入），會引發「維度災難」，導致記憶體與計算資源不堪重負。

DQN 透過引入深度神經網路（特別是卷積神經網路 CNN）來解決此問題。它不再使用表格，而是訓練一個神經網路 $Q(s,a;\theta)$ 來近似 Q 函數，其中 θ 是網路的權重參數。這使得 Agent 能夠從高維度的原始輸入（如像素）直接學習策略，並具備泛化能力，能對未見過的狀態做出合理判斷。

DQN 核心改進技術

為提升訓練的穩定性與效率，DQN 引入了兩項關鍵技術：

- 經驗回放 (Experience Replay): 將 Agent 與環境互動的每一次經驗（包含狀態、動作、獎勵、下一狀態）(s,a,r,s') 儲存在一個固定大小的記憶體中。訓練時，從中隨機抽取小批量（mini-batch）的經驗進行學習。這打破了數據之間的時序相關性，使訓練過程更穩定，同時也提高了數據的利用效率。
- 目標網路 (Target Network): 在計算目標 Q 值（即 $r+\gamma \max_{a'} Q(s',a')$）時，若使用與預測網路相同的網路參數，會導致目標值不斷變動，如同「追逐一個移動的目標」，使訓練難以收斂。DQN 引入一個獨立的「目標網路」，其參數 θ^- 定期（而非每一步）從主網路 θ 複製而來，從而在一段時間內保持目標值的穩定，大幅提升了學習的穩定性。

DQN 改良版演算法

基於 DQN，本專案進一步研究了兩種主流的改良演算法：

- Double DQN: 為了解決標準 DQN 中 max 操作符可能導致的 Q 值高估問題。Double DQN 將動作的 選擇 與 評估 解耦：使用主網路來選擇在下一個狀態中價值最高的動作，再用目標網路來評估這個被選中動作的 Q 值。這有效緩解了過度樂觀的估計，使學習更穩定。

- Dueling DQN: 該架構將 Q 值的估計分解為兩部分：狀態價值函數 V(s) 和優勢函數 A(s,a)。V(s) 表示處於某狀態本身的價值，而 A(s,a) 表示在該狀態下，選擇某個動作相對於平均動作的優勢。最終的 Q 值由兩者組合而成。這種結構能更高效地學習狀態的價值，提升學習效率與策略評估的準確性。

專案實作流程與成果

本專案分為兩個主要階段，從理論的深入理解過渡到應用的具體實踐。

第一階段：理論學習與基礎奠定

此階段的目標是為團隊建立扎實的強化學習理論基礎。我們透過密集的聚會討論與資源研讀，系統性地學習了以下內容：

- 學習資源： 綜合利用了 HackMD 筆記、莫煩 Python 教學、經典論文（Watkins & Dayan, 1992; Mnih et al., 2015）以及線上部落格與教學影片。
- 學習成果：
    - 深刻理解了強化學習的試誤學習與獎勵回饋機制。
    - 完全掌握了 Q-Learning 的核心思想、Q 值更新公式與 ε-greedy 策略。
    - 建立了從理論到進階演算法（DQN）的知識體系，為後續實作鋪平了道路。

第二階段：Unity 環境建構與演算法整合

在具備理論基礎後，團隊轉向實作，目標是將所學應用於 Unity 遊戲專案中。

- 環境架設與驗證： 我們學習並成功在 Unity 中架設了 ML-Agents 強化學習環境。初期，我們採用 ML-Agents 預設的 PPO (Proximal Policy Optimization) 演算法進行基礎訓練，以驗證我們設計的環境、觀測空間、動作空間及獎勵函數是否合理有效。透過此過程，Agent 成功學習了前進、跳躍等基本行為。
- 自製演算法應用計畫： 我們的核心目標之一是將自製的 DQN 演算法

應用於專案中。計畫將 Unity 環境透過介面轉換為類 Gym 的形式，以便與我們用 Python 實現的 DQN 模型對接，並與 PPO 的表現進行橫向比較。

- 遊戲設計與 AI 訓練： 我們從零開始製作了一個基礎的類瑪莉歐平台遊戲。為 Agent 精心設計了：
    - 觀測 (Observation): Agent 周圍的環境網格，標示出平台、敵人、終點等。
    - 動作空間 (Action Space): 離散的動作集合，如「向左移動」、「向右移動」、「跳躍」。
    - 獎勵函數 (Reward Function):
        - 抵達終點：+1.0 (巨大正向獎勵)
        - 碰到敵人/掉落：-1.0 (巨大負向獎勵)
        - 每一步消耗：-0.001 (微小負向獎勵，鼓勵盡快完成)
        - 朝向終點移動：+0.01 (微小正向獎勵，引導方向)

挑戰與創新：GPT-2 地圖生成

為提升訓練環境的多樣性與 Agent 的泛化能力，我們進行了一項創新性嘗試：利用 GPT-2 模型 生成遊戲地圖。我們將地圖結構表示為文本序列，並訓練 GPT-2 學習其模式。儘管生成的地圖在初期並未完全符合遊戲機制的預期，但這次探索為 AI 生成內容（AI-Generated Content, AIGC）在遊戲開發中的應用提供了寶貴的經驗。

遭遇挑戰與解決方案

在專案進行中，我們遇到了來自理論和實作層面的多重挑戰，透過團隊協作，逐一克服。

挑戰類別	具體問題	解決方案
理論理解	Q-Learning 公式過於抽象，難以建立直觀感受。	透過交叉參考論文、觀看多個教學影片、小組白板推演與討論，將抽象符號與具體遊戲場景對應，化抽象為具體。
	DQN 及其變體的網路結構與運作方式複雜。	利用網路資源與 ChatGPT 進行針對性提問，釐清了經驗回放、目標網路等關鍵機制背後的原理，並透過小組分享加深理解。
技術實作	Unity 與 ML-Agents 版本不相容，導致安裝失敗。	仔細查閱官方文件與社群論壇，找到穩定匹配的版本組合，成功搭建開發環境。
	遊戲開發中出現大量 BUG，且運行不流暢。	採用模組化開發，逐一進行功能測試與除錯。對遊戲物理、渲染管線進行優化，提升了整體的流暢度與穩定性。
	GPT-2 生成的地圖不符合遊戲可玩性要求。	分析了失敗原因，意識到需要更精細的數據表示法與更嚴格的後處理規則。此問題作為未來可深入研究的方向。
	自製演算法與 ML-Agents 的整合存在技術壁壘。	研究 ML-Agents 的 Python API，設計了通訊介面，計畫將 Unity 環境的狀態傳遞給外部 Python 腳本，並接收動作指令。

總結與未來展望

主要學習與成就

透過本次專題，團隊取得了顯著的成長與豐碩的成果：

1. 系統性理論掌握： 從 RL 基礎到 DQN 系列演算法，團隊建立了全面且扎實的理論知識體系。
2. 核心技術深化理解： 深刻理解了經驗回放、目標網路、Double DQN、Dueling DQN 等技術如何解決具體問題，提升 AI 訓練效能。
3. 跨領域實作能力： 成功整合了遊戲開發（Unity）與強化學習（ML-Agents、Python），具備了從零到一設計、開發並訓練遊戲 AI 的能力。
4. 高效問題解決能力： 面對未知技術難題，團隊展現了積極的自學精神與協作能力，能有效利用網路、論文、AI 工具及團隊智慧解決問題。

未來展望

本專案成功地驗證了強化學習在遊戲 AI 上的應用潛力，並為未來的研究奠定了堅實基礎。我們規畫了以下幾個可行的發展方向：

- 演算法優化與比較： 完成自製 DQN 的整合，並與 PPO、Rainbow 等更先進的演算法在相同環境下進行性能的量化比較。
- 完善地圖生成器： 改進 GPT-2 或嘗試其他生成模型（如 GAN），設計更有效的地圖表示方式與驗證規則，實現高品質的程序化關卡生成。
- 提升 Agent 智慧層級： 引入更複雜的觀測（如記憶機制）和動作（如與環境互動），讓 Agent 能夠應對需要長期規劃的複雜謎題。
- 專案成果展示： 將訓練完成的 AI 模型打包成可執行的遊戲 Demo，並錄製展示影片，作為團隊技術能力的具體證明。