113學年度下學期強化學習DQN期末專題

201 05吳苡柔

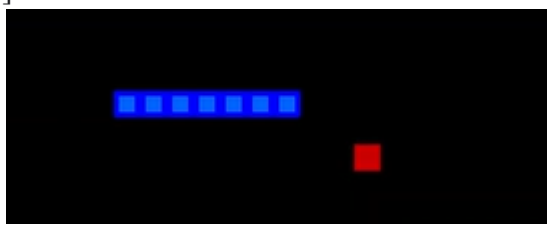
壹、問題敘述：

- **選用環境：**
  將之前用 Pygame 自製的「貪吃蛇」遊戲，修改成適合智能體訓練的環境。 仿照 gymnasium 環境的格式，加入 reset() 初始化、play_step(action) 控制動作和 rewards 回傳獎勵、is_collision() 判斷死亡條件等等。

- **遊戲目標：**
  操控貪吃蛇，在不撞到牆壁和自身的狀況下，吃紅點點以獲高分。

- **狀態空間：**
  由 11 個布林值所組成，用 one-hot encoding 的方式表示目前狀態，其狀態回傳的資訊包括：往各方向移動是否會撞到牆壁或自己、當前移動方向、蛇頭和食物的相對位置，如下：

```
[
   danger_straight, danger_right, danger_left,   # 危險判斷
   move_left, move_right, move_up, move_down,    # 當前方向
   food_left, food_right, food_up, food_down     # 食物方向
]
```



回傳 [0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1]

- **動作空間：**
  用 one-hot encoding 表示三種離散動作，以控制貪吃蛇前進方向，分別為：

動作	說明
[1, 0, 0]	什麼都不做
[0, 1, 0]	向右轉(順時針)
[0, 0, 1]	向左轉(逆時針)

- **獎勵機制：**
  設計在 play_step() 裡：

情況	獎勵值	說明
吃到食物	+10	鼓勵靠近與吃食物
撞牆或撞到自己	-10	處罰死亡

其他	0	正常移動沒有特別獎勵

- **死亡條件：**
  由 is_collision() 函式判斷，以下狀況會導致死亡：
  - 蛇撞到邊界
  - 蛇撞到自己的身體
  - 遊戲時間超過 100 × 蛇長 (避免智能體一直在原地轉圈，不吃紅點)

貳、DQN原理介紹：

- **探索與利用**
  小時候不知道以什麼方法在領紅包時能拿最多錢，經過幾年探索，以立正、躺著、翹二郎腿、盤腿、倒立、蹲著、跪著、劈腿、下腰、後空翻.....不同姿勢領紅包後，發現以土下座的姿勢最容易拿到 10000 元，因此長大後可能會較常利用土下座姿勢領紅包，這便是探索與利用的生活化例子。在深度 Q 網路（DQN）中，**探索（Exploration）與利用（Exploitation）**是非常重要的概念，用來平衡在學習過程中探索新策略與利用已知策略之間的關係，特別是在強化學習中面對未知環境或複雜任務時尤其關鍵。常用的演算法為 ε−greedy，它根據機率使智能體選擇目前已學到的最優動作（即利用當前知識）或進行探索（隨機採取動作）。這種策略透過平衡探索和利用，旨在在學習過程中逐步減少探索率，從而轉向使用最優策略。即使是學習的經驗也可以被重新利用，因此也能透過經驗回放（Experience Replay）以隨機抽樣過往學習資料來提供探索，以在不同的狀態下重新探索可能的動作。簡單來說，探索與利用就是一個避免原地踏步的機制。

- **DQN**
  DQN 全稱是 Deep Q Network，是一種基於 Q−learning 再變化的深度學習＋強化學習的結合，由 DeepMind 研發，並且可以解決傳統 Q 學習在處理高維、複雜環境時的困難。本質上是運行 Q−Table，再反覆挑出最大 Q 值，但與傳統 Q−Table 不同的是，DQN 通過使用深度神經網絡來近似 Q 函數，可以解決 Q−Table 在狀態空間非常大或無法離散化時遇到的問題。

- **Replay Buffer**
  Replay Buffer 的主要作用是儲存先前的經驗數據，以便後續的訓練可以隨機抽樣這些數據進行學習，提升樣本利用率，而不是僅僅依賴當前的經驗。儲存的資料包含狀態（state）、動作（action）、獎勵（reward）、下一個狀態（next state）等。另外，Replay Buffer 的其他功能包含打亂樣本關聯性。當連續的訓練資料有高度相關時，使用回放資料能讓訓練時不會陷入局部最優或導致過度擬合，就像某學生連續十天請假在家複習，前五天都讀化學，後五天都讀數學，考試時很可能會忘了大部分前五天讀的化學再說什麼，為了避免神經網路也出現類似情況而在 DQN 中加入 Replay Buffer 後，便能有效緩解高估。

- **fixed Q Target**
  fixed Q Target 是為了避免使用同一個 Q 網路來決策目前以及目標，避免訓練不穩定或是過度估計。在 DQN 中，通常會設定兩個神經網路：目前 Q 網路（用於選擇動作和計算目前狀態下的 Q 值）與目標 Q 網路（用於計算目標 Q 值，會週期性更新網路參數），以使得目標 Q 值相對穩定。

- **Bootstrapping**
  Bootstrapping，中文翻譯為自舉，在強化學習中的意思是用估計去更新同類的估計。具體來說，Bootstrapping 是結合已有的獎勵與未來的獎勵，以更新現在的獎勵，並且能夠在沒

有先驗證假設分佈的情況下，透過反覆抽樣和重複計算來對統計量進行估計，是一種強大的統計工具。舉生活化的例子，假設你現在準備期末考了，因為時間很趕，沒辦法做太多事，很猶豫到底要先讀物理第五章、第六章還是做期末專題。一開始發現自己第五章小考成績較不理想，決定先複習第五章，但後來學姊告訴你每個選項對物理分數的重要性，分別是第五章會有40分，第六章會有60分，做專題0分。同時你又發現第五章的內容可能會延伸到第六章，所以你調整複習計畫，把時間多分給第五章，未來可能會取得較好的時間。像這樣重新評估每件事的分數權重，這過程就像Bootstrapping，結合過去(學姊經驗)、當下(小考分數)、未來(期末考分數)，反覆更新當前的策略(複習計畫)。

- **Double DQN**

  Double DQN（Double Deep Q-Network）是對經典的 DQN 演算法的改進，旨在解決傳統 DQN 中存在的過度估計問題。在傳統的 Q-learning 中，用於評估下一個狀態的最大動作值時使用的 Q 函數與選擇動作時使用的 Q 函數是同一個，可能導致對動作價值的高估。 **Double DQN** 引入了兩個獨立的 Q 函數（Q-networks），分別用於選擇動作和評估動作值，透過這種方式減少了過度估計的發生，提高學習的穩定性跟效率。

- **Dueling DQN**

  Dueling DQN 是一種透過重新設計 Q 網路的結構來提升效能的方法。Dueling DQN 將 Q 函數分解為狀態值函數（State Value Function）和優勢函數（Advantage Function）。狀態價值函數估計狀態的基本價值，而優勢函數則估計每個動作相對於其他動作的優劣程度，也就是比較成本的概念。用種田概念來類推就是，原本只會預期收穫為多少，要施肥嗎？但是 Dueling DQN 可以學習狀態以及比較利益，比如在豐雨年，不論種法，可以預期有不錯的收成，也可以說此時狀態價值函數較高；而在蝗災年，就可以預期遭殃，不論種法如何，此時狀態價值函數較低。優勢函數則是比較分析動作間的相對可能造成的結果，比如說現在去睡覺而不是寫報告對於我的健康有更大的益處。一言以蔽之，優勢函數就是相對優秀解。

- **Prioritized Experience Replay(PER) DQN**

  PER DQN 是一種針對 DQN Replay Buffer 進行改進的演算法，會有這樣的改進是因為，不同經驗對智能體學習的重要性不同，因此修改經驗池的抽樣方式，傾向從記憶中抽取「學習價值高」的經驗來學習。在傳統的 DQN 中，經驗回放 (Replay Buffer) 是隨機抽樣的，會導致一些關鍵但稀有的經驗被忽略。而 PER 引入了 優先級 (Priority) 概念，透過 TD 誤差 (Temporal Difference error) 來衡量每筆經驗的重要程度，誤差越大表示學習機會越高，因此更容易被抽取來訓練。這就像在段考前複習時，要根據你「哪些題目錯最多」來決定你要重做哪些題目，而不是隨機挑題。這樣可以讓學習更有效率，聚焦於「弱點補強」。當然為了避免過度偏差，PER 也加入了 重要性取樣修正因子 (Importance Sampling Weights)，避免模型偏向高優先級經驗而忽略整體分布，維持學習的公平性與穩定性。

- **Rainbow-lite DQN**

  Rainbow DQN 是一種混和了各種 DQN 技術的演算法，就像彩虹上有很多顏色而得名。而這次採用的 **Rainbow-lite DQN** 則是 Rainbow DQN 的精簡版，功能融合了 Double DQN、Dueling DQN 和 PER DQN。Rainbow-lite 相較於 Rainbow 就像一套精簡卻實用的農耕策略，不一定要把所有農業技術 (NoisyNet、Distributional Q、全套正規化等) 都用上，但把核心技術像是「挑地 (Dueling)」、「選種 (Double)」和「看天氣重練 (PER)」用好，也能有很高收成。

參、研究方法：

1. 搭建 DQN、Double DQN、Dueling DQN、PER DQN、Rainbow-lite DQN 的網路架構

DQN：

DQN 最主要的特徵是用 Q 網路取代 Q-Table，並配合經驗回放與 Q-learning 的目標更新策略，訓練 agent 在「貪吃蛇」環境中學會選擇最適當的行動。

網路架構

本專題 DQN 所使用的 Q 網路，是由兩層全連接層組成，架構如下(圖一)：

- **輸入層：**輸入維度為 11，對應於貪食蛇遊戲中設計的狀態向量（包含移動方向、障礙物方向、食物相對位置等）。
- **第一層線性層 fc1：**輸出維度為 128，激活函數採用 ReLU。
- **第二層線性層 fc2：**輸出維度為 64，激活函數採用 ReLU。
- **輸出層 out：**輸出維度 3，分別是直行、左轉、右轉三種動作的 Q 值。

另外加入了經驗回放（Replay Buffer）及使用 Fixed Q Target 以緩減模型發生高估的狀況。

```
class DQN(torch.nn.Module):
    def __init__(self, input_dim=11, output_dim=3):
        super(DQN, self).__init__()
        self.fc1 = torch.nn.Linear(input_dim, 128)
        self.fc2 = torch.nn.Linear(128, 64)
        self.out = torch.nn.Linear(64, output_dim)

    def forward(self, x):
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        return self.out(x)
```

^(圖一) DQN 架構

參數選擇

- 折扣因子 **γ** : 0.9
- 學習率 **lr** : 0.001
- 最小探索率: 0.05
- 目標網路更新頻率: 200
- 批次大小 **batch_size** : 64
- 最大記憶容量 **MAX_size** : 10000
- 訓練回合數 **N_EPISODES** : 1000

Double DQN：

Double DQN 改良了 DQN 中可能出現的「高估 Q 值」問題，透過兩個神經網路模型（預測網路與目標網路）分別負責動作選擇與 Q 值評估，使訓練更穩定且表現更佳。

網路架構

本專題 Double DQN 所使用的 Q 網路，是由兩層全連接層組成，同 DQN 。

與 DQN 差異

Double DQN 和 DQN 最大差異在 target Q 的計算方式。計算方式如下(圖二):

- 使用 預測網路 model 選擇下一步的最佳動作 (argmax(Q(s', a)))
- 使用 目標網路 target_model 計算該動作的 Q 值
- 計算 target Q 時不再直接取最大值，而是獎勵最優動作的加權。

```
q_pred = self.model(s).gather(1, a.unsqueeze(1)).squeeze()

with torch.no_grad():
    best_actions = self.model(s_).argmax(dim=1, keepdim=True)
    q_next = self.target_model(s_).gather(1, best_actions).squeeze()
    q_target = r + self.gamma * q_next * (1 - done)
```

^ (圖二) Double DQN 的 target Q 計算

參數選擇

- 折扣因子 γ : 0.9
- 學習率 lr : 0.001
- 最小探索率: 0.05
- 目標網路更新頻率: 200
- 批次大小 batch_size : 64
- 最大記憶容量 MAX_size : 10000
- 訓練回合數 N_EPISODES : 1000

Dueling DQN：

Dueling DQN 最主要的特徵是將 Q 網路分成狀態價值 (Value) 和優勢函數 (Advantage) 進行處理，最後再合併成 Q 值。使智能體的在某些動作不影響結果的情況下更快學習。

網路架構

本專題 Dueling DQN 所使用的 Q 網路由三部分組成，先透過兩層全連接層抓特徵，再分為兩條路徑計算狀態價值和優勢函數。架構如下 (圖三):

- **輸入層：**輸入維度為 11，對應於貪食蛇遊戲中設計的狀態向量 (包含移動方向、障礙物方向、食物相對位置等) 。
- **第一層線性層 fc1：**輸出維度為 128，激活函數採用 ReLU。
- **第二層線性層 fc2：**輸出維度為 64，激活函數採用 ReLU。

- **狀態價值：**
  - **value_fc：**輸出 32 維向量
  - **value：**輸出 1 個數值，表示該狀態的整體價值 V(s)
- **優勢函數：**
  - **adv_fc：**輸出 32 維向量
  - **adv：**輸出 3 維向量，分別表示三個動作的優勢值 A(s, a)
- **Q 值合併公式：**
  - $Q(s,a) = V(s) + (A(s,a) - |A|1a'\sum A(s,a'))$
- **輸出層 out：**輸出維度 3，分別是直行、左轉、右轉三種動作的 Q 值。

```
class DQN(torch.nn.Module):
    def __init__(self, input_dim=11, output_dim=3):
        super(DQN, self).__init__()
        self.fc1 = torch.nn.Linear(input_dim, 128)
        self.fc2 = torch.nn.Linear(128, 64)

        self.value_fc = torch.nn.Linear(64, 32)
        self.value = torch.nn.Linear(32, 1)

        self.adv_fc = torch.nn.Linear(64, 32)
        self.adv = torch.nn.Linear(32, output_dim)

    def forward(self, x):
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))

        value = self.value(F.relu(self.value_fc(x)))
        adv = self.adv(F.relu(self.adv_fc(x)))

        return value + (adv - adv.mean(dim=-1, keepdim=True))
```

^(圖三) Dueling DQN 架構

參數選擇

- 折扣因子 **γ** : 0.9
- 學習率 **lr** : 0.001
- 最小探索率: 0.05
- 目標網路更新頻率: 200
- 批次大小 **batch_size** : 64
- 最大記憶容量 **MAX_size** : 10000
- 訓練回合數 **N_EPISODES** : 1000

PER DQN：

PER DQN 最主要的特徵是傾向從 Replay Buffer 中選取 TD 誤差大的經驗，而非全部隨機採樣。網路架構如下：

網路架構

本專題 PER DQN 所使用的 Q 網路，是由兩層全連接層組成，同 DQN 。

與 DQN 差異

PER DQN 和 DQN 最大差異在 Replay Buffer 的採樣方式。計算方式如下:

- 優先經驗回放：根據 TD 誤差調整每筆經驗被抽取的機率。(圖五)
- 重要性採樣：在損失計算中加入權重，修正非均勻抽樣所導致的偏差。(圖四)

```
td_error = q_pred - q_target
prios = (td_error.abs() + 1e-5).detach().cpu().numpy()
self.rb.update_priorities(indices, prios)
loss = (td_error.pow(2) * weights.to(device)).mean()
```

^ (圖四) PER 更新

```
def append(self, s, a, r, s_, done):
    max_prio = self.priorities.max() if self.size > 0 else 1.0

    self.s[self.ptr] = s
    self.a[self.ptr] = a
    self.r[self.ptr] = r
    self.s_[self.ptr] = s_
    self.done[self.ptr] = done
    self.priorities[self.ptr] = max_prio

    self.ptr = (self.ptr + 1) % self.max_size
    self.size = min(self.size + 1, self.max_size)

def sample(self, batch_size, beta=0.4):
    if self.size == self.max_size:
        prios = self.priorities
    else:
        prios = self.priorities[:self.ptr]

    probs = prios ** self.alpha
    probs /= probs.sum()

    indices = np.random.choice(len(probs), batch_size, p=probs)
    weights = (len(probs) * probs[indices]) ** (-beta)
    weights /= weights.max()

    return (
        torch.FloatTensor(self.s[indices]),
        torch.LongTensor(self.a[indices]),
        torch.FloatTensor(self.r[indices]),
        torch.FloatTensor(self.s_[indices]),
        torch.FloatTensor(self.done[indices]),
        torch.FloatTensor(weights),
        indices
    )

def update_priorities(self, indices, prios):
    for idx, prio in zip(indices, prios):
        self.priorities[idx] = prio
```

^ (圖五) Replay Buffer 差異

參數選擇

- 折扣因子 **γ** : 0.9
- 學習率 **lr** : 0.001
- 最小探索率: 0.05
- 目標網路更新頻率: 200
- 批次大小 **batch_size** : 64
- 最大記憶容量 **MAX_size** : 10000
- 訓練回合數 **N_EPISODES** : 1000
- **TD** 優先權 **α** : 0.6

- **IS 修正指數 β：**一開始為 0.4，隨時間線性增加至 1.0

Rainbow-lite DQN：

Rainbow-lite DQN最主要的特徵是其結合了前幾項 DQN 的優點。像 Duling DQN 將 Q 網路分成價值函數與優勢函數。像 Double DQN 用預測網路選擇動作、目標網路評估 Q 值。Replay Buffer 的抽樣方式改用重要性採樣。

參數選擇

- **折扣因子 γ** : 0.9
- **學習率 lr** : 0.001
- **最小探索率**: 0.05
- **目標網路更新頻率**: 200
- **批次大小 batch_size** : 64
- **最大記憶容量 MAX_size** : 10000
- **訓練回合數 N_EPISODES** : 1000
- **TD 優先權 α** : 0.6
- **IS 修正指數 β：**一開始為 0.4，隨時間線性增加至 1.0

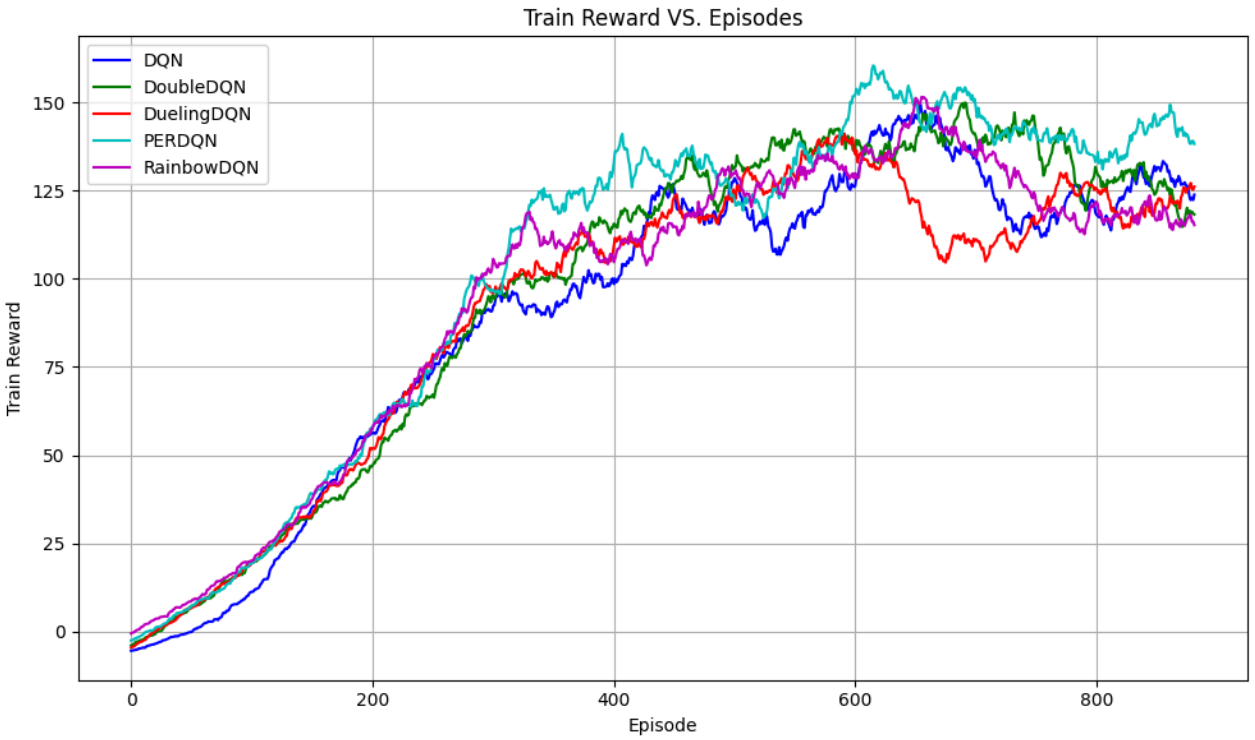
2. DQN、Double DQN、Dueling DQN、PER DQN、Rainbow-lite DQN訓練的比較

訓練模型時，我們將折扣因子 γ 固定為 0.9 、 探索最低點 eps_low 固定為 0.05、學習率 lr 固定為 0.001、訓練次數 N_EPISODES 固定為 1000，並使用五種在 研究方法1 介紹的不同 DQN 演算法，執行 貪吃蛇 環境並訓練，最後畫出不同 DQN 的訓練分數和訓練回合關係圖並分析結果。

肆、研究結果與分析：

DQN、Double DQN、Dueling DQN、PER DQN、Rainbow-lite DQN訓練的比較

(劃出不同DQN的訓練分數和訓練回合的關係圖並分析結果)



- DQN:
  - 線性成長相較其他演算法，成長較為緩慢。
  - 最高分達到 150 左右。
  - 最終收斂大約在 125 左右。
- Double DQN:
  - 初期成長穩定勝過其他演算法（300~600回合）。
  - 後期表現差（600~800回合），明顯下降。
  - 最高分達到 150 左右
  - 最終收斂大約在 125 左右。
- Dueling DQN:
  - 初期成長穩定，但到中後期（600回合後）大幅下降，穩定性略低。
  - 最高分達到 135 左右。
  - 最終收斂大約回升到 125 左右。
  - 為本次專題最差模型。
- PER DQN:
  - 在中期之後（300 回合後），明顯超越其他模型。為本次專題最優模型。
  - 在中後期（600~800）達到高峰，且波動不大，穩定性高。
  - 最高分達到 160 左右。
  - 最終收斂大約在 140 左右
- Rainbow-lite DQN:
  - 初期成長成長速度僅次於 PER DQN。
  - 最高分達到 150 左右。
  - 最終收斂大約在 125 左右
- 整體：
  - 初期皆穩定成長。
  - 到 600 回合左右，所有模型都呈現下降趨勢，表示可能 600 回合後，就達到過度擬合。

伍、討論與心得：

1. 透過這次報告及了解相關研究，更具體了解 DQN 及其相關演算法運作原理與組成。
2. 目前環境為了避免智能體一直繞圈圈，不去吃紅點，因此設定成遊戲時間超過身長的

100倍就會死亡。這在前期確實能達成效果，但到了後期可能造成智能體難以訓練，例如:為了避開身體需要花較長時間才能吃到紅點。因此環境該如何設計會讓智能體更好訓練是值得思考的問題。

3. 這次專題把環境包成跟 gymnasium 一樣的環境，但還有些功能沒實現（像是 render()），未來可以繼續研究該如何寫環境。
4. Rainbow-lite DQN 的表現比預期差，或許可以試試 Rainbow DQN（加入 NosiyNet 等等）。或重新調整參數。

陸、參考資料

- Mini-Batches in RL. (2018, December 20). Stackoverflow. https://stackoverflow.com/questions/53864434/mini-batches-in-rl
- Amit Yadav. (2024, October 9). N-Step Bootstrapping in Reinforcement Learning. Medium. https://medium.com/@amit25173/n-step-bootstrapping-in-reinforcement-learning-e4f70f264933
- 劉建平. (2018, October 16). 強化學習(十一) *Prioritized Replay DQN*. 博客園. https://www.cnblogs.com/pinard/p/9797695.html
- 微笑sun. (2018, December 24). 強化學習（四）—— *DQN*系列（*DQN, Nature DQN, DDQN, Dueling DQN*等）. 博客園. https://www.cnblogs.com/jiangxinyang/p/10112381.html
- Openchat. (2024, January 25). 強化學習中的*RainbowDQN*. 稀土掘金. https://juejin.cn/post/7327723045287559205
- Z-yq. (2024, May 29).【*RL*】自訂義強化學習環境（基礎）. CSDN. https://blog.csdn.net/weixin_41434829/article/details/139204435
- DQN 成果影片