國立臺灣科技大學資訊工程系

一百零七學年度第一學期專題研究

總報告

根據深度強化學習技術遊玩遊戲

研究組員

陳安泰 B10432003

王彥翔 B10432010

指導教授：陳錫明

中 華 民 國 108 年 1 月 14 日

國立臺灣科技大學資訊工程系

一百零七學年度第一學期專題研究

總報告

根據深度強化學習技術遊玩遊戲

研究組員

陳安泰 B10432003

王彥翔 B10432010

指導教授：陳錫明

中 華 民 國 108 年 1 月 14 日

目 錄

圖 目 錄

表 目 錄

摘 要

受到論文《Playing Atari with Deep Reinforcement Learning》的啟發，本篇專題希望嘗試透過 Deep Q-Learning 的技術訓練程式遊玩遊戲，並以最後能通關為目標。本專題報告論述的內容涵蓋了機器學習在遊戲領域相關應用的介紹，以及相關的背景知識、技術與流程，並闡述本專題為了訓練能夠遊玩此類遊戲的程式而所使用的強化學習方面的設計與方法。我們使用了功能強大的 python，透過將當前遊戲畫面所轉成的向量矩陣，經過算法找到有價值的動作，並不斷嘗試各種動作所能帶來的獎勵收益。由於以往的遊戲通常都有明確的獎勵用來表示當前遊戲有所進展，如 Atari 遊戲中的計分表，這使得訓練模型時可以直接以該分數作為獎勵值計算，然而在目標遊戲《Getting Over It with Bennett Foddy》中，並沒有可以拿來當獎勵的分數數值，因此必須要另外設計。我們使用遊戲人物在遊戲進程中的位置來計算獎勵，讓訓練的模型學習能根據當前畫面，從給定的動作組合中，找到最適合的選擇。而所給定的動作組合，是研究者在觀察網路上的遊戲通關影片之後整理歸納而得的。最終訓練好的模型，儘管還不能夠通關整個遊戲，但仍能分別以 95%和 85%的成功率通過第一項與第二項障礙，證實如此的訓練及獎勵方式能有效地使得模型確實地在遊戲做出前進的行為，但仍有改進空間。另外，透過更改遊戲中的控制設定，我們觀察到在操作方式複雜且可能性極大的遊戲中，些微的差異可以對結果產生很大不同。最後，研究者發現了本專題所使用設計的限制，以及 Deep Q-Learning 本身的弱點，並反思日後應該如何精進以在從事其他項目時能有更好的結果。

關鍵詞： Deep Q-Learning、Getting Over It、Python

一、 緒論

1.1 研究動機

最近有一款熱門遊戲名為《Getting Over It with Bennett Foddy》，中文譯名為「和班尼特福迪一起攻克難關」。此遊戲以單純的遊戲目標和極高的控制難度在網路上紅極一時，研究者也嘗試玩過，但是難度實在太高、太過困難，始終無法成功通關。而近幾年來，人工智慧相關產業、技術等蓬勃發展，也應用在各式各樣的領域上，不論是工業上、商業上，甚至遊戲領域也都多有涉及，因此我們便順應這股熱潮，嘗試結合機器學習以及該遊戲，從而決定試著訓練出能玩此遊戲的程式。

1.2 專題目標

本專題的目標為透過機器學習的技術訓練程式遊玩遊戲，最後通關，並且玩得比一般人要好，讓一般人可以像看影片一樣觀賞並能學習其中的訣竅等。

(1) 成功運作並確實在遊戲做出前進的行為

我們希望訓練好的模型能夠確實的在操作遊戲人物往遊戲的目標方向走而不是隨意亂動。因為該遊戲的特點，具體目標訂定為能夠通過第一個障礙物——樹木，能夠通過意味著他有成功學到東西。

(2) 嘗試讓它能前進更遠、更進步

因為該遊戲流程並不短，再加上強化學習很花時間，所以我們希望在達成前項目標後能夠盡可能再前進得更遠一點，除了距離外，也希望它在操作時能更流暢，能比人玩得更好。

1.3 研究方法

本專題採用由 Deepmind 公司於 2013 年所發表的《Playing Atari with Deep Reinforcement Learning》(Mnih et al., 2013)[1]中所提出的 Deep Q-Learning 架構來應用於訓練模型遊玩《Getting Over It with Bennett Foddy》(2017) 這部遊戲。

在《Playing Atari with Deep Reinforcement Learning》中，研究者將過去的 Q-Learning (Watkins, 1989) [2]與神經網路相結合，並證明僅使用原始像素即可征服

Atari 2600 電腦遊戲的困難控制策略，可說是將強化學習與神經網路合併的第一步，並在之後發展出了 DDQN、DDPG 等更強化的強化學習架構。

為了進行實驗，我們先使用 python 很快地寫出了能夠對遊戲進行操作的 agent，寫出 Deep Q-Learning 的架構後，嘗試使用各種參數和獎勵函數來訓練，最後比較結果。

二、 研究背景

2.1 著名的強化學習架構

強化學習(Reinforcement learning)是機器學習中的一個領域，是一個透過一連串「觀察環境、選擇行為、執行行為、獲得獎勵」來學習得到最佳行為策略的方法。它和監督式學習之間的區別在於，它並不需要出現正確的輸入/輸出對，也不需要精確校正每次優化的行為。其靈感來源於心理學中的行為主義理論，即有機體如何在環境給予的獎勵或懲罰的刺激下，逐步形成對刺激的預期，產生能獲得最大利益的習慣性行為。[3]

2.1.1 Q-learning

Q-learning (Watkins, 1989)是在有限馬可夫決策過程(Finite Markov Decision Process)的框架下尋找最大的獎勵期望值 Q 的演算法。在無限的探索之下，Q-learning 可以形成最佳的行為選擇策略使得它所探索的馬可夫決策過程能有最大的回饋效益。Q-learning 的問題之一便是收斂效率慢。[2]

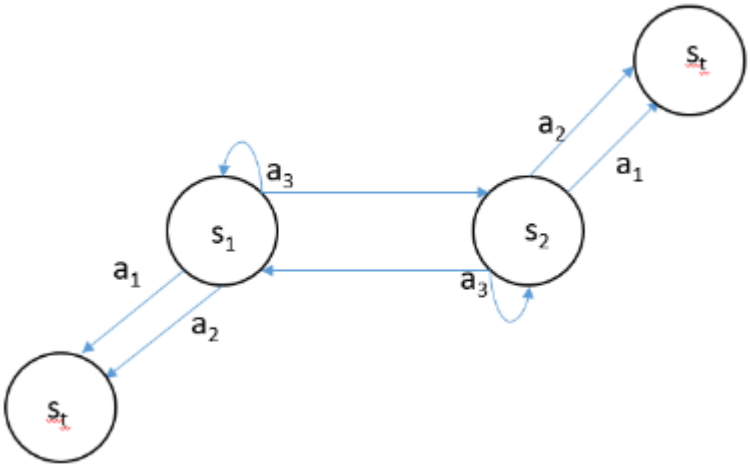

圖 1　　馬可夫決策過程範例圖

Initialized

Q-Table		Actions					
		South (0)	North (1)	East (2)	West (3)	Pickup (4)	Dropoff (5)
States	0	0	0	0	0	0	0
	.	.	.	.	.	.	.
	.	.	.	.	.	.	.
	.	.	.	.	.	.	.
	327	0	0	0	0	0	0
	.	.	.	.	.	.	.
	.	.	.	.	.	.	.
	.	.	.	.	.	.	.
	499	0	0	0	0	0	0

Training

Q-Table		Actions					
		South (0)	North (1)	East (2)	West (3)	Pickup (4)	Dropoff (5)
States	0	0	0	0	0	0	0
	.	.	.	.	.	.	.
	.	.	.	.	.	.	.
	.	.	.	.	.	.	.
	328	-2.30108105	-1.97092096	-2.30357004	-2.20591839	-10.3607344	-8.5583017
	.	.	.	.	.	.	.
	.	.	.	.	.	.	.
	.	.	.	.	.	.	.
	499	9.96984239	4.02706992	12.96022777	29	3.32877873	3.38230603

圖 2　　Q 數值 Table，其數值初始化為 0，然後在訓練過程中更新每一格的數值

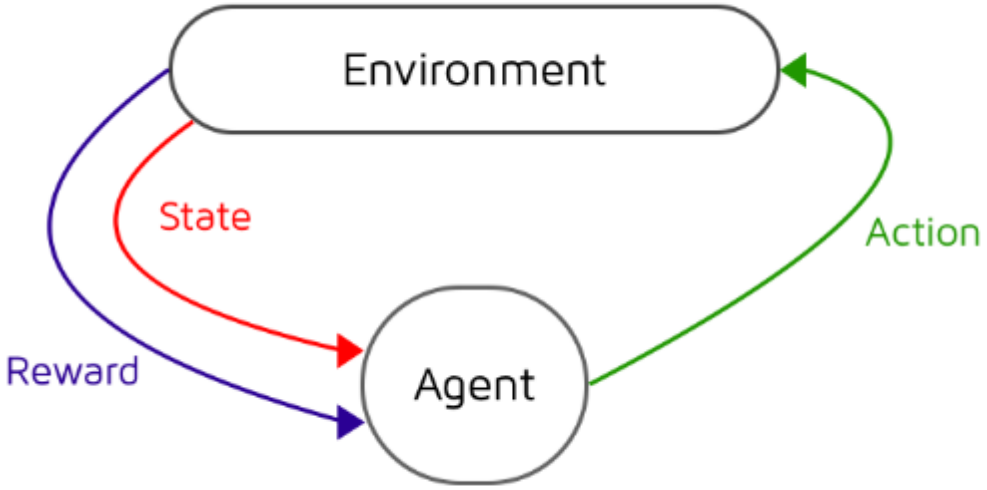

圖 3　　Q-Learning 流程簡單示意圖

2.1.2 Policy Gradient

Policy Gradient (Sutton, Richard S., et al. 2000) 是強化學習的另一大家族，它與 Value-base 方法(Q-learning)在概念上相似，但不同於 Q-learning 尋找 action 的最大獎勵期望值，它是直接尋找最佳的行為選擇策略，跳過了計算 value 的階段。輸出的這個動作可以是一個連續的值，之前 Q-learning 的方法輸出的都是不連續的值，然後再選擇值最大的動作。而 Policy Gradient 可以在一個連續分佈上選取動作。然而它的問題學習效率很低。[4]

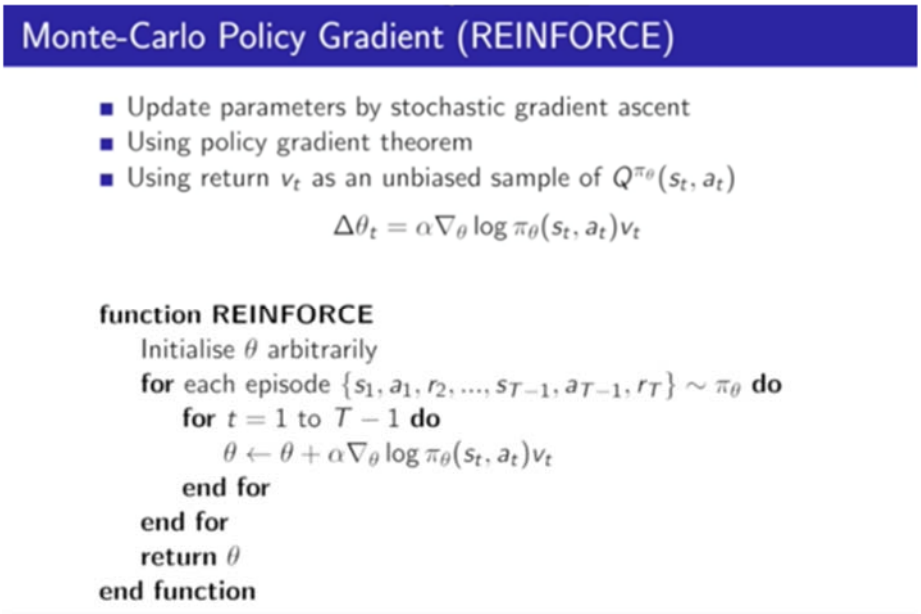

圖 4　　Policy Gradient 範例演算法(Reinforce 方法)

2.1.3 Actor-Critic

Actor-Critic 是一個結合 Q-learning 和 Policy Gradient 的演算法，它試圖利用 Q-learning 來彌補 Policy Gradient 的學習效率問題，但依舊受限於 Q-learning 收斂問題。它的改良算法 A2C 和 A3C 的目的便是要解決 Q-learning 依賴獎勵的更新算法。[5]

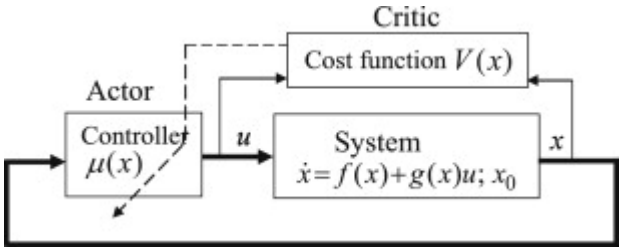

圖 5　　Actor-Critic 示意圖

2.2 強化學習的著名使用

2.2.1 AlphaGo

AlphaGo 是於 2014 年開始由英國倫敦 Google DeepMind 開發的人工智慧圍棋軟體。

圖 6　　AlphaGo 圖標

AlphaGo 使用蒙地卡羅樹搜尋(Monte Carlo tree search)，藉助估值網路(value network)與策略網路(policy network)這兩種深度神經網路，通過估值網路來評估大量選點，並通過策略網路選擇落點。AlphaGo 最初通過模仿人類玩家，嘗試符合職業棋士的過往棋局，其資料庫中約含 3000 萬步棋著。後來它達到了一定的熟練程度，它開始和自己對弈大量棋局，使用強化學習進一步改善它。在這種設計下，電腦可以結合樹狀圖的長遠推斷，又可像人類的大腦一樣自發學習進行直覺訓練，以提高下棋實力。

AlphaGo 的研究計劃於 2014 年啟動，和之前的圍棋程式相比表現出顯著提升。在和 Crazy Stone 和 Zen 等其他圍棋程式的 500 局比賽中，單機版 AlphaGo 僅輸一局。而在其後的對局中，分散式版 AlphaGo 在 500 局比賽中全部獲勝。而之後從業餘的水平到世界第一，AlphaGo 的棋力取得這樣的進步，僅僅花了二年左右。AlphaGo 在沒有人類對手後，AlphaGo 之父傑米斯·哈薩比斯宣布 AlphaGo 退役。最終版本 Alpha Zero 可自我學習 21 天達到

勝過中國棋神柯潔的 Alpha Go Master 的水平。[6]

時間	戰績	成就
2015 年 10 月	擊敗樊麾	第一個無需讓子即可在 19 路棋盤上擊敗圍棋職業棋士的電腦圍棋程式
2016 年 3 月	五番棋比賽中 4:1 擊敗李世乭	第一個不藉助讓子而擊敗圍棋職業九段棋士的電腦圍棋程式
2016/12/29 ~2017/1/4	以「Master」之名在網路快棋對戰中挑戰中韓日台的高手並 60 戰全勝	
2017 年 5 月烏鎮圍棋峰會	和世界第一棋士柯潔比試取得 3 比零全勝	成為世界第一，被中國圍棋協會授予職業圍棋九段稱號，不過聶衛平九段稱它的水平「至少 20 段」
	八段棋士和 AlphaGo 合作人機配對賽中獲勝	
	對決五位頂尖九段棋士團體賽獲勝	

表 1　　AlphaGo 戰績表

AlphaGo 被譽為人工智慧研究的一項標誌性進展，在此之前，圍棋一直是機器學習領域的難題，甚至被認為是當代技術力所不及的範疇。樊麾戰的棋局裁判托比·曼寧和國際圍棋聯盟的秘書長李夏辰兩人都認為將來圍棋棋士可以藉助電腦來提升棋藝，並從錯誤中學習。

2.2.2 DeepMind 能源管理

數據中心需要大量冷卻設施來讓伺服器運轉，因而十分耗電，如今世界上所有數據中心一起使用的電力可佔全球電力的 2%，如果不加以控制，這種能源需求可能會像互聯網一樣迅速增長。因此，高效運行數據中心是一件重要的事情。在 Google 的人工智能研究組織 DeepMind 發了一篇關於 DQN 的文章引起了轟動後，在 Google 的效率工程師的建議下，DeepMind 與數據中心情報(DCIQ)團隊合作訓練並找出更高效、通用的工作模式，透過強化學習方式對各個數據中心進行最佳化，如今已可以降低 15%的電力開銷，其中冷卻方面更是可減少 40%的電力消耗。(Evans, R., Gao, J. 2016)[7]

圖 7　　Google 數據中心大量冷卻設備

此外 DeepMind 也在跟英國負責輸配電工作的國家電網公司(National Grid)洽談合作，希望在不增加其它基礎建設的狀況下，透過類神經網路跟機器學習技術來協助降低英國的用電量。DeepMind 的共同創辦人兼 CEO 認為只要透過 AI 進行最佳化，就可以每年幫英國省下 10%的電，英國在 2014 年大約生產 330TWh 的電，成本是數百億英鎊，所以如果每年真的可以節省 10%的話，不管是碳排放量或者發電成本，都是一筆非常可觀的數字。

英國的發電廠跟配電網是由不同的公司負責，而平衡電力需求與供給的任務還是落在負責配電工作的國家電網身上，電力的需求其實是可以預測的，像是透過標準人類行為模式來判斷，或者是天氣變化，都能讓我們先預測電力需求的消長。但是能源供給就相對比較不穩定且難以預測，尤其是再生能源。2016 年聖誕節英國單日的再生能源供電比例就突破 40%，成為再生能源一個重要的里程碑，但是供電穩定性不足也是老問題，如何維持供需不失調就是現在的大難關。DeepMind 認為只要經由機器學習科技，就能夠協助電力系統減少外在環境衝擊造成的影響，例如透過機器學習預測供電與用電的尖峰，可以最佳化再生能源的使用，也就是可以透過他們的類神經網路科技，就所有可能影響供電效率的上百萬項可能因素中進行分析，找出之前英國國家電網沒發現的原因跟解法。

2.2.3 Google Brain & DeepMind 機械手臂

2016 年，Google Brain 展示了他們的協作機械臂研究項目—合作式增強學習讓機器人掌握通用技能：一台機械臂學會的東西，可以在所有機械臂之間共享，這樣所有的機械臂都能以更快的速度學習，成長。實驗結果是，這些機械臂可以進行開門，拿起罐子等簡單的操作。

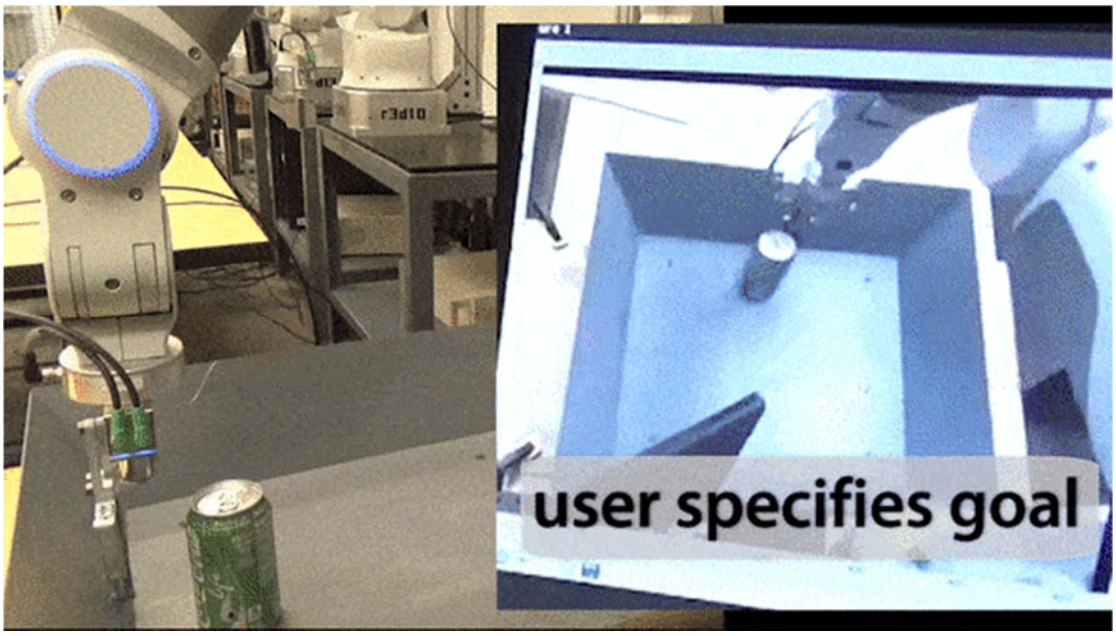

圖 8　　機械臂操作圖，圖像中點擊目標後機械臂移動物體

當時，Google Brain 的研究人員探索了通過多機器人合作完成通用目技能學習的三個可能方法：

(1)　直接從經驗中學習行動技巧

(2)　學習物體內部物理模型

(3)　在人類協助下學習技能

在這三個例子中，多個機器人共享彼此的經驗，搭建了一個通用的技能模型。雖然學習的技能相對簡單，但是 Google Brain 的研究人員表示，通過合作來更快速高效的學習這些技能，機器人未來可能會掌握更加豐富的行為指令集，最終會讓它們在人類的日常生活中起到大的作用。

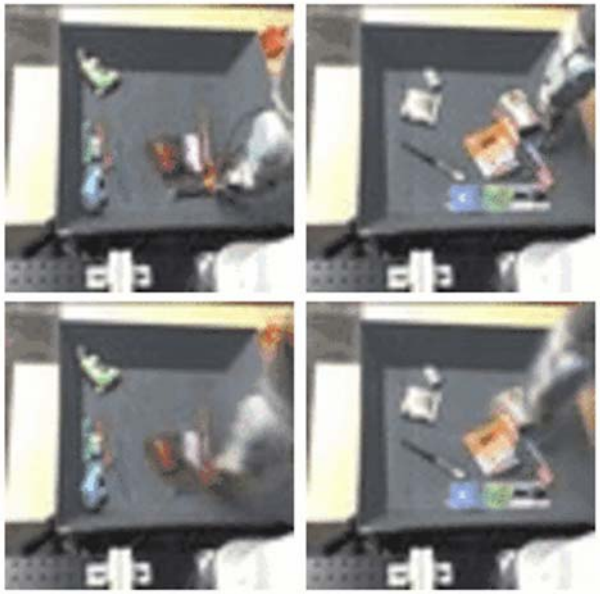

圖 9　機械臂預測路徑圖。上排為實際和物件互動圖，下排為預測路徑圖，模糊部分為預測路徑

其中，第一項，直接從經驗中學習行動技巧，也就是讓機器人用強化學習從原始經驗中學習。機器人通過實時反饋發現變化，進而增強和探索，得到更大回饋的變化。

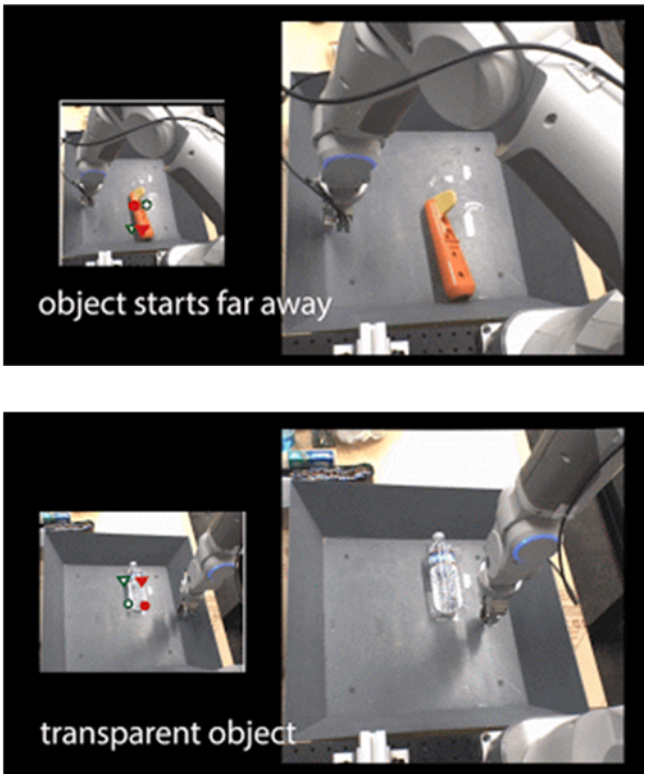

圖 10　機械臂執行圖。根據看到的任務要求的特定像素，來移動到特定的位置

Google Brain 由於擁有多個機器人，因此可以在真實世界中進行實驗。日前，沒有實體機器人的 DeepMind 也在虛擬環境中，對機器人協作開展了最新的研究。DeepMind 的方法是讓機械臂在模擬器中成功地找到一塊積木，並將這塊積木拿起來，最後將這塊積木疊在另一個積木上。實驗中，機器人會彼此共享信息，並使用最終得到的數據改善核心算法，從而學會更好地搭積木。

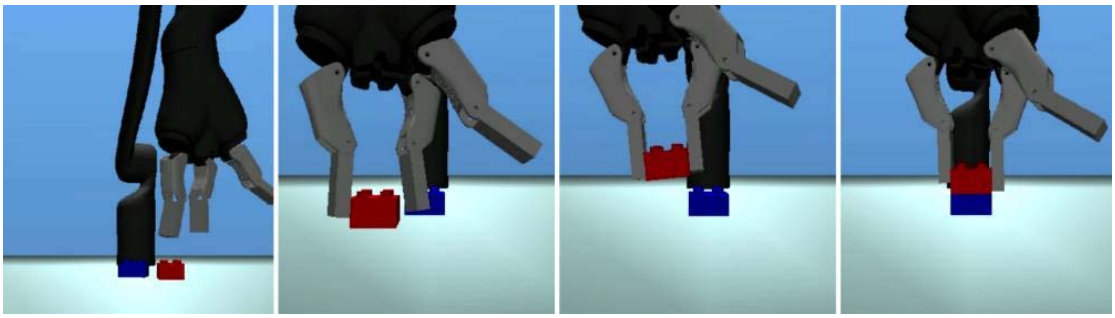

圖 11　機械臂在虛擬環境中疊積木

在他們日前發表的論文中(Popov, I. et al 2017)[8]，將訓練 16 台機器人所用的時間成功縮短到僅需要 10 小時。論文中 DeepMind 研究人員使用的是 Determinisic Policy Gradient (DPG)算法，並且從兩方面做了改善：

(1) 學習時間加長，讓算法使用 Experience replay 將數據更新的反饋給予延遲，發現這樣能讓機器人學得更好。研究人員將改進的算法稱為 DPG-R。

(2) 借助 Asynchronous Actor Critic (A3C)算法的分佈式思想，巧妙對 DPG-R 算法做了改造，使其能夠多台不同的計算機和虛擬機器人之間共享。

DeepMind 使用的機械臂是 Kinova Robotics 開發的 Jaco，實驗中使用的當然是 Jaco 的虛擬版。據介紹，Jaco 有 9 個角度可以自由活動(手臂上 6 個，手掌上 3 個)。看起來很靈活，當然這也是 Jaco 的優勢，但要操縱 Jaco 機械臂完成實際任務需要的計算量也相應的十分龐大。從這一點上，DeepMind 的實驗充分展現了使用端到端的方法訓練機械臂的好處。

圖 12　機械臂 Jaco

深度學習和強化學習方法近來被用於解決各種連續控制領域的問題。這些技術最顯著的一個應用便是機器人的靈活操縱任務，讓機器人完成靈活的操難以用傳統的控制理論或手工設計方法解決。這種任務的一個例子是抓取一個物體，並將其精確地堆疊在另一個物體上。這是一個困難而且與現實世界中很多實際應用都相關的問題，解決這個問題也是機器人領域一個重要的長期目標。在這裡，我們通過在虛擬環境中對這個問題進行考察，並提出了解決這個問題的模型和技術，朝解決實際機器人靈活操作邁出了一步。

論文從兩方面擴展了深度確定策略梯度(Deep Deterministic Policy Gradient，DDPG)算法，提出了一種基於 Q-Learning 的無模型方法，使其在數據利用率和可擴展性方面得到大幅提升。研究結果表明，通過大量使用 off-policy 數據和 replay，可以找到穩定抓取物體並進行堆疊的控制策略。此外，研究結果顯示，通過收集真實機器人的交互數據，可能很快就能成功訓練堆疊策略。

2.3 Deep Q-Learning

Deep Q-Learning 是基於 Q-Learning，並結合深度類神經網路的一個機器學習演算法。在 Q-Learning 中，系統會與環境$\mathcal{E}$進行一連串的互動。每過一段時間，系統從可操作的動作集合A中選擇一個動作a_t。此操作使得環境狀態及分數改變，程式會觀察環境當下狀態得到x_t，並從分數的改變得到獎勵值r_t。由於有時當下的觀察可能無法包含從開始以來的所有環境資訊，因此定義序列$s_t = x_1, a_1, x_2, a_2, \ldots, x_t, a_t$來表達環境在時間 t 當下的狀態，這裡假設時間有最大值 T。

系統的學習目標即是做出使未來獎勵最大化的動作。假設未來的獎勵每往後一步就有以γ為倍數的衰減$(0 < \gamma \leq 1)$，則在時間為 t 的時候，未來將獲得的總獎勵即為$R_t = \sum_{t'=t}^{T} \gamma^{t'-t} r_{t'}$。我們定義一個理想動作價值函數$Q^*(s, a)$，其為在環境狀態為 s 下，動作 a 將預期得到的最大總回饋獎勵，即：

$$Q^*(s, a) = max_\pi \mathbb{E}[R \mid s_t = s, a_t = a, \pi]$$

這時，若下一個時間點的狀態序列為s'，則預期的最大獎勵即是從所有可能動作中選擇一個動作a'使得有最大值$r + \gamma\, Q^*(s', a')$，因此

$$Q^*(s, a) = \mathbb{E}_{s' \sim \varepsilon}[r + \gamma max_{a'} Q^*(s', a') \mid s, a]$$

為求得Q^*，我們可先粗略猜測一個Q_i，並透過上述的方程式迭代更新：

$$Q_{i+1}(s, a) = max_\pi \mathbb{E}\,[r + \gamma max_{a'} Q_i(s', a') \mid s, a], \text{when } i \to \infty, Q_i \to Q^*$$

這方法雖然基本，卻是很不實用的，因為這需要將所有狀態都個別紀錄，沒有做歸納，因此通常使用的方法是做函數近似：$Q(s, a; \theta) \approx Q^*(s, a)$。

Deep Q-Learning 的概念即是採用類神經網路/深度學習的技巧來近似理想動作價值函數，我們並稱這樣的網路叫 Deep Q Network。

為給予 Deep Q Network 足夠的輸入資料，我們使用 Experience Replay (經驗重現)的策略。它會在每一個時間點將程式與環境互動所得到的反饋集合成一個個經驗$e_t = (s_t, a_t, r_t, s_{t+1})$，並儲存在記憶 D 裡面：$D = e_1, \ldots, e_N$，並在與環境互動的同時隨機抽樣$e \sim D$來為 Q 函數更新，並且使用「ε-greedy」的策略，由隨機選擇執行的動作漸漸變化成依據價值函數選擇動作。

由於要讓神經網路接受任意長度的輸入數據的難度太高了，因此還需要設計一個函數φ，讓狀態序列s簡化成一個固定長度的向量$\varphi(s)$，作為價值函數的輸入張量。

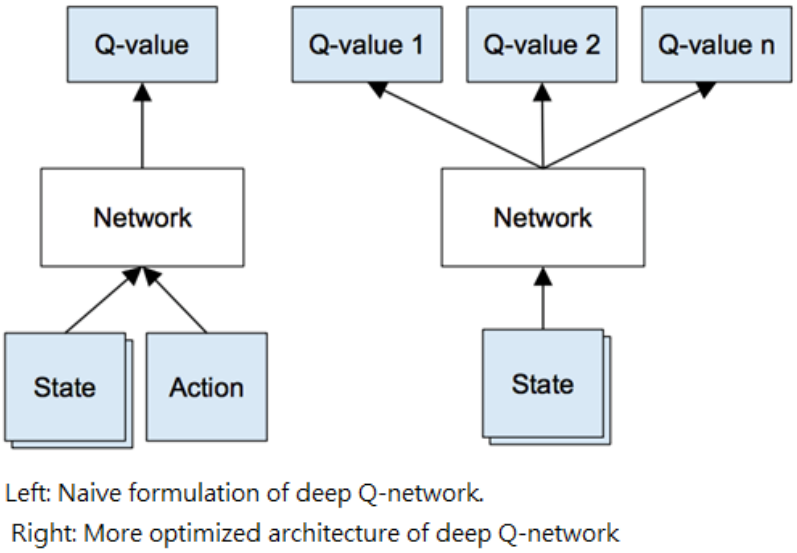

圖 13　Deep Q Network 示意圖

2.4 Getting Over It with Bennett Foddy

《Getting Over It with Bennett Foddy》是一款單人平台類動作遊戲，是作者 Bennett Foddy 於 2017 年 10 月 6 日發佈的實驗性遊戲。此遊戲以單純的目標、困難且特殊的操作方式著名，並於推出時在網路上風靡一時。其巧妙的設計也使它得到當年的多項遊戲設計獎項提名[9]，此遊戲正是本專題選定要讓程式遊玩的遊戲。

圖 14　Getting Over It

三、 系統設計

3.1 系統環境與工具

3.1.1 使用工具

程式語言：Python

使用的 Python 模組：Pyautogui、PIL、Keras、Numpy、Pywin32

3.1.2 開發環境

作業系統：Windows 10

顯示卡：NVIDIA GeForce GTX 950M

Getting Over It with Bennett Foddy 遊戲版本：1.5861

3.2 程式架構

程式有主要四個部份，和遊戲互動的 Agent、Deep Q-Network、Memory 以及獎勵函數。

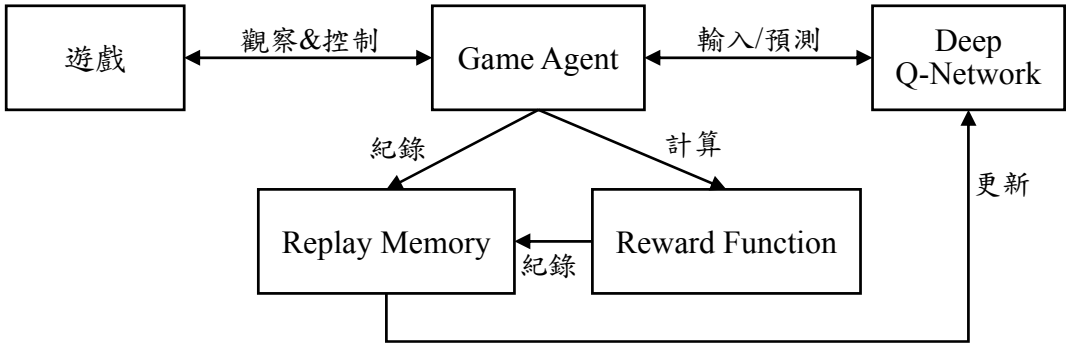

圖 15 程式架構圖

3.3 執行流程

Deep Q-Learning 演算法 (來源：參考文獻[1])

初始化 replay memory D ，大小為 N

初始化 Q 網路

for episode = 1, M do

初始化狀態序列$s_1 = \{x_1\}$ 並處理得$\varphi_1 = \varphi(s_1)$

for t = 1, T do

ε 的機率下選擇一個隨機動作 a_t

否則選擇 $a_t = argmax_a Q(\varphi(s_t), a; \theta)$

在遊戲中執行動作 a_t 並觀察獎勵r_t跟畫面x_{t+1}

設$s_{t+1} = (s_t, a_t, x_{t+1})$ 並處理得到 $\varphi_{t+1} = \varphi(s_{t+1})$

將 $(\varphi_t, a_t, r_t, \varphi_{t+1})$ 存至 D

從 D 隨機抽一段 minibatch$(\varphi_j, a_j, r_j, \varphi_{j+1})$

$$y_j = \begin{cases} r_j & , for\ terminal\ \varphi_{j+1} \\ r_j + \gamma max_{a'} Q(\varphi_{j+1}, a'; \theta) & , for\ non\text{-}terminal\ \varphi_{j+1} \end{cases}$$

使用y_j對 Q 作梯度下降更新

end for

end for

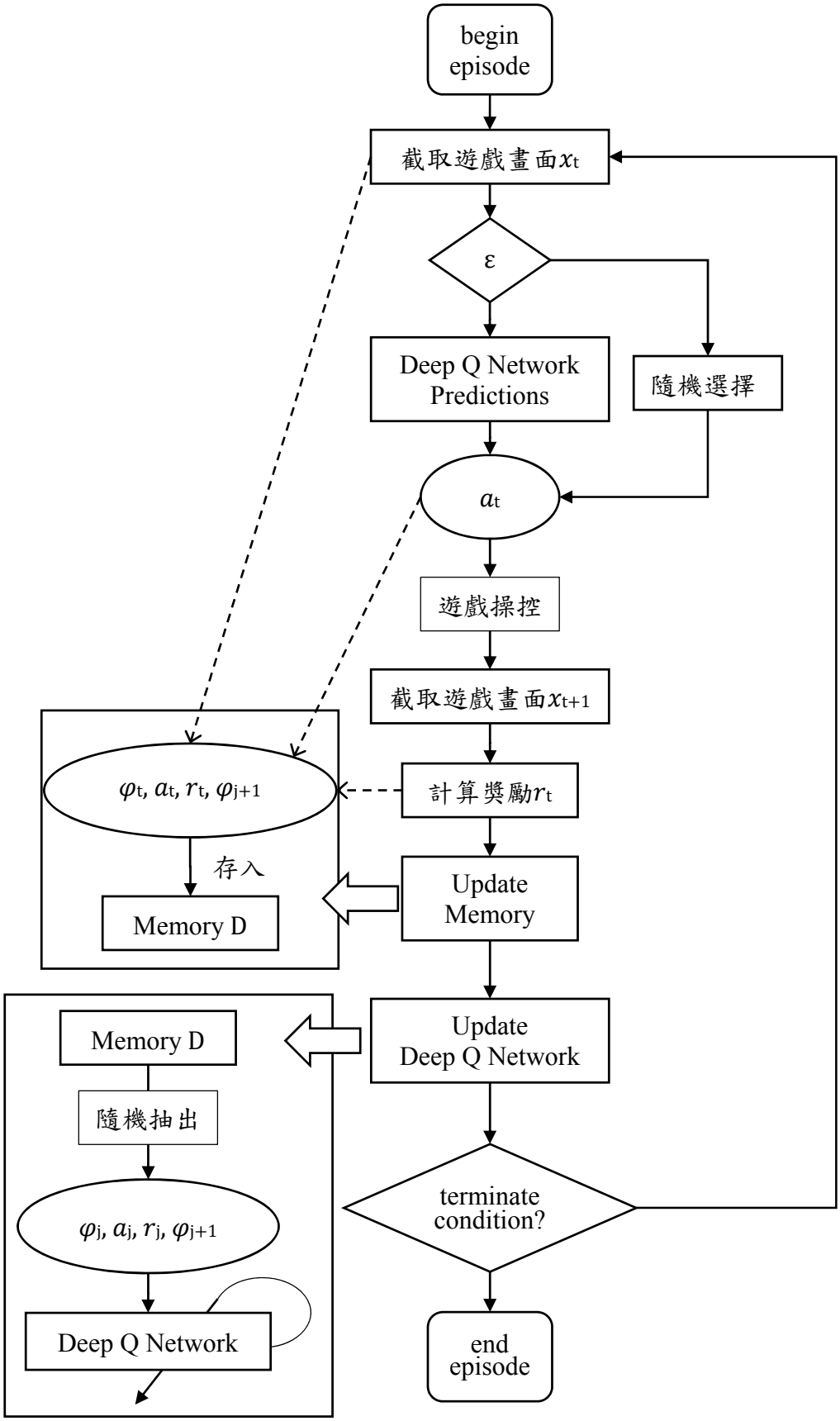

圖 16　　每回合執行流程示意圖

3.4 獎勵函數

獎勵函數是整個強化學習中最重要的部分，好的獎勵函數能夠讓模型明確知道怎麼做比較好，所以它會影響模型的學習效率與上限。由於此遊戲與常見遊戲不同，沒有相關計分的畫面，所以必須自行設計獎勵的計算方式。

我們一開始先設計出了圖片差值算法，此算法本意為鼓勵模型看到新的畫面，由於目標遊戲是平台通關類遊戲，走在通向終點的路上會比卡住和往回走看到更多的新畫面，因此在大量的嘗試過後，有益於前往目標的動作會得到更多的期望值。

圖片差值獎勵算法的問題是它需要非常大量的探索，使得學習效率非常慢。因此我們又設計了進度表(progress list)獎勵算法，此算法利用遊戲截圖間接地給訓練模型提供了標準答案，儘管這樣需要人類給答案的設計有點違反強化學習的本意，不過在前置實驗中，我們發現進度表算法的效率與圖片差值算法相比有非常顯著的提高。

(1) 圖片差值算法：為用x_{t+1}計算得到r_t，此算法從x_1到x_t的紀錄中尋找一個x_i使得x_{t+1}與x_i的所有像素的差的絕對值的總和為最小值：

$$m = min_{i'}\left(|x_{i'} - x_{t+1}|\right)$$

$$i = argmin_{i'}\left(|x_{i'} - x_{t+1}|\right)$$

在找到x_i有與x_{t+1}最小的圖片差值m之後，檢查m是否大於一個門檻，如果是的話，那麼就認為它到了一個「沒看過」的畫面並給它獎勵，如果不是，那麼它就是在曾經來過的位置，並給它零或負的懲罰，具體如以下算式：

$$r_t = \begin{cases} -\frac{d\times(t-i)}{t}, & m < threshold \\ c, & m \geq threshold \end{cases}$$ (基礎獎勵$c > 0$、基礎懲罰$d \geq 0$)

若$i = t$，表示畫面幾乎沒有移動，則懲罰為 0，而i越小懲罰越趨近d。

(2) 進度表(progress list)算法：此算法需要將角色前進至目標的過程畫面每過一段距離截一張圖，並按照時間先後順序給予編號，集合成 progress list = $m_0, m_1, \ldots, m_N$，然後計算：

$$n(x) = argmin_i(|x - m_i|)$$

n傳回x在 progress list 中所能找到最相似畫面的編號，再來就可以利用這編號$n(x)$得到分數$p(x)$。另外我們希望編號增加而得到的分數是指數遞增而不是線性遞增的，因此使用遞增率$\gamma^{-1/N}$，這將使得$n(x)=$N 時所得到的分數為c/γ。

分數$p(x)=n(x)c\gamma^{\frac{-n(x)}{N}}$ (基礎獎勵$c>0$、獎勵遞增率$\gamma^{-1/N}$)

最後，將x_{t+1}和x_t所得到的分數相減，得到獎勵$r_t=p(x_{t+1})-p(x_t)$

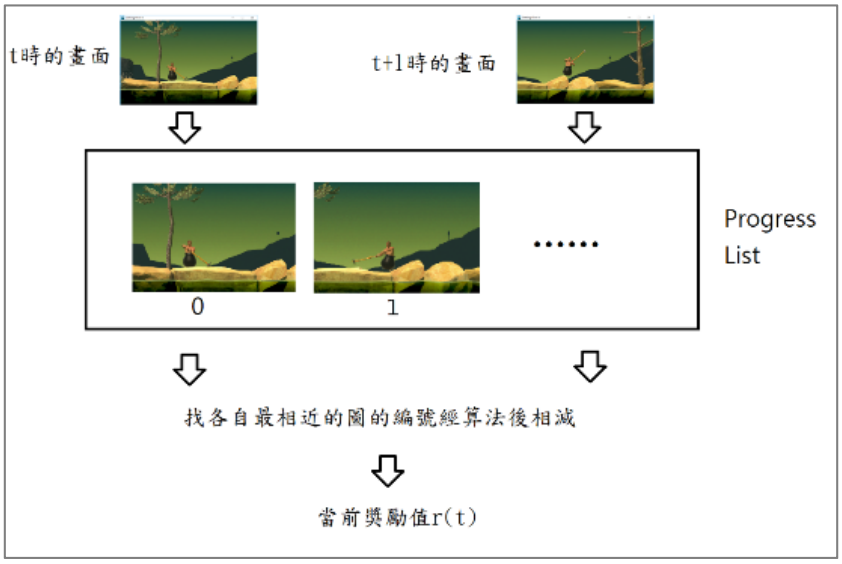

圖 17　進度表獎勵算法示意圖

```python
def calReward(self, pre_scrshot, cur_scrshot) :
    pre_scrshot = np.squeeze(pre_scrshot) # before action
    cur_scrshot = np.squeeze(cur_scrshot) # after action

    min_pre_diff = 2147483648
    pre_map = -1

    min_cur_diff = 2147483648
    cur_map = -1

    for this_mapnum, this_mapshot in enumerate(self.mapList) :
        d = np.sum(np.absolute(this_mapshot - pre_scrshot))
        if d <= min_pre_diff :
            min_pre_diff = d
            if this_mapnum > pre_map : pre_map = this_mapnum

        d = np.sum(np.absolute(this_mapshot - cur_scrshot))
        if d <= min_cur_diff :
            min_cur_diff = d
            if this_mapnum > cur_map : cur_map = this_mapnum

    if pre_map == cur_map :
        return 0
    else :
        # r = base * m * (rg ^ m)
        pre_score = self.base_score * pre_map * (self.rev_gamma ** pre_map)
        cur_score = self.base_score * cur_map * (self.rev_gamma ** pre_map)
        return cur_score - pre_score
# end def calReward
```

圖 18　進度表獎勵算法程式碼

3.5 ϵ-greedy

ϵ-greedy 是一個最常用的權衡「探索-利用問題」的策略，ϵ 代表執行執行「探索」的機率。比如設置 ϵ＝0.1，那麼就表示有 10%的機率會進行「探索」操作，而 90%會進行「利用」操作，也就是利用當前最好的選擇。在 Deep Q-Learning 的訓練中，ϵ 值會隨每回合下降一點。它會在每回合開始前先計算好當前 epoch 的 ϵ 值，之後再根據 ϵ 值選擇「探索」(隨機動作)還是「利用」(最大預期獎勵的動作)。

```python
this_epoch_eps = max(set.eps_min, set.epsilon * (set.eps_decay ** e))
```

```python
# make action
predQ = np.squeeze(self.Q.predict(self.add_noise(cur_shot)))
avgQ_list[e] += np.max(predQ) / set.steps_epoch
if np.random.random() < this_epoch_eps :
    cur_action = np.random.randint(set.actions_num)
else :
    cur_action = np.argmax(predQ)

self.game.do_control(cur_action)
```

圖 19　ϵ-greedy 程式碼

3.6 Deep Q Network

我們所使用的 Deep Q Network 的架構如下圖所示。因為要簡化設計和難度，神經網路的輸入張量並不是不定完整的狀態序列，而是將序列經過一個函數φ以化簡為固定大小的輸入。本專題使用$\varphi(s_t) = x_t$。

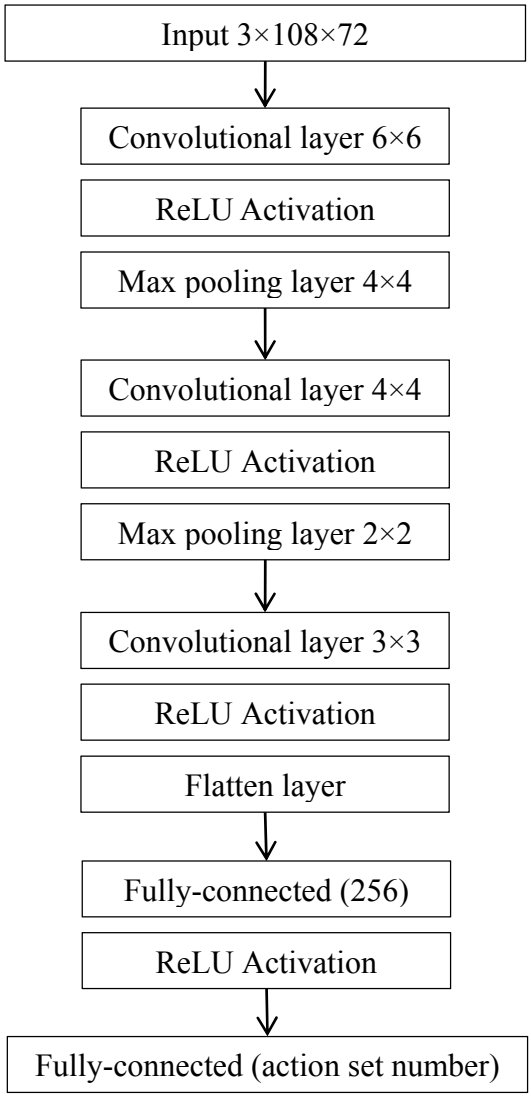

圖 20　使用的 Deep Q Network 設計

3.7 動作組合

由於該遊戲並非是透過有限的按鍵來控制操作，而是用滑鼠滑動的方式進行操控，理論上會有近乎無限的動作組合。然而越多的動作組合會使模型訓練起來更費時，所以研究者在觀察真人遊玩的操作後，挑選出最常使用的幾個動作納入動作組合之中，主要分為兩類，直線型操作以及畫圓弧操作，並再依據各種角度細分，且每種操作又各有快、慢兩種，共有 36 種動作可供程式選擇執行。

```
def do_control(self, id) :

    intv_time = 0.001
    # is straight
    if id < set.mouse_straight_angles * 2 :

        if id < set.mouse_straight_angles :
            # slow
            delta, distance = 3, 2400
        else :
            # fast
            delta, distance = 20, 4000

        angle = 2 * math.pi * id / set.mouse_straight_angles
        d_x = math.ceil(math.cos(angle) * delta)
        d_y = math.ceil(math.sin(angle) * delta)

        for i in range(distance // delta) :
            self.directInput.directMouse(d_x, d_y)
            sleep(intv_time)
        if id >= set.mouse_straight_angles :
            sleep(0.02)
    # is round
    else :
        id -= set.mouse_straight_angles * 2

        if id < set.mouse_round_angles * 2 :
            is_clockwise = 1
        else :
            is_clockwise = -1
            id -= set.mouse_round_angles * 2

        if id < set.mouse_round_angles :
            # slow
            radius, delta, proportion = 560, 4, 0.8
        else :
            # fast
            radius, delta, proportion = 720, 20, 0.7

        angle_num = 36.0
        angle_bias = 4.0
        angle_offset = (id / set.mouse_round_angles) + angle_bias / angle_num
        edge_leng = math.floor(2 * radius * math.sin(math.pi / angle_num))
        # we cut a circle's edge into circular arcs.
        # each arcs is similar to the base of an isosceles triangle
        # an isosceles triangle with legs = r and apex = a has base = 2r * sin(a/2)

        for i in range(int(angle_num * proportion)) :
            angle = 2 * math.pi * (i * is_clockwise / angle_num + angle_offset)
            d_x = math.ceil(math.cos(angle) * delta)
            d_y = math.ceil(math.sin(angle) * delta)
            for j in range(edge_leng // delta) :
                self.directInput.directMouse(d_x, d_y)
                sleep(intv_time)
        sleep(0.01)
    sleep(set.do_control_pause)
# end def do_control()
```

圖 21　動作組合與操作動作

3.8 其他訓練參數

除了盡可能優化算法外，研究者也嘗試了許多種訓練參數數值，並觀察能否有更好的表現，如調整 ϵ 大小及衰減速率，調整 learning rate，到調整每回合有幾步或是總共多少回合等，然而由於時間不足，總訓練步數(回合數x每回合步數)基本上和論文所做的相比少了許多，若是能夠訓練足夠久，成果應可以再更進步，下面的圖為最終使用的參數值、輸入以及相關設定等實際程式碼

```
# SCREENSHOTS SETTING
shot_w = 108
shot_h = 72
shot_c = 3
shot_shape = (1, shot_h, shot_w, shot_c)
shot_resize = (shot_w, shot_h)
shot_intv_time = 0.01
shot_wait_max = 100
noise_range = 0.008

# Q NET SETTING
model_input_shape = (shot_h, shot_w, shot_c)
learning_rate = 0.0005
learning_rate_decay = 1e-4

# REWARD SETTING
use_mapreward = False
mapname_list = sorted(os.listdir("map/"), key = sorting_filename_as_int)
no_move_thrshld = shot_h * shot_w * shot_c * 0.05 * ((shot_c - 1) * 0.01 + 1)
gamma = 0.5
base = 1.0


# ACTION SETTIN
mouse_straight_angles = 12
mouse_round_angles = 6
actions_num = (mouse_straight_angles * 2 ) + (mouse_round_angles * 2)
# ROUND ACTION ONLY HAS CLOCKWISE BECAUSE COUNTER-CLOCKWISE IS USELESS
# {slow straight(12), fast straight(12), cwise round slow / fast(12), ccwise round slow / fast(12)}
do_control_pause = 0.03

# STEP QUEUE SETTING
stepQueue_length_max = 10000 # set 0 to be no limit

# TRAINING SETTING
use_target_Q = False
epsilon = 1.0
eps_min = 0.1
eps_decay = 0.98

check_stuck = True
stuck_thrshld = 100

epoches = 300
steps_epoch = 150
train_thrshld = 100
steps_train = 4
train_size = 48

test_intv = 5
draw_fig_intv = 20

eps_test = 0.1
steps_test = 100
```

圖 22　訓練參數

四、 實驗成果

4.1 實驗設計

為了測試訓練好的模型是否有真正的學習到如何玩遊戲，以及能否玩得好，實驗設定兩個目標點，測試 100 次遊玩中，在 100 步內能夠達成這兩個目標的機率各有多少。此外我們也測試在不同遊戲環境設定下，對於模型的訓練而言是否會有不同的影響。

訓練時的設計是：獎勵函數的部份，我們使用進度表獎勵算法，設基礎獎勵為 1.0，$\gamma = 0.5$。在 Deep Q-Network 的部份使用 RMSprop 優化算法，每 4 步更新一次，mini-batch 大小為 48。訓練回合數為 300，每回合最大步數為 150 步，ε 從 1.0 以每回合等比例降低 0.98 至最小值 0.1。而在測試模型的目標達成率時，我們設 ε 為 0.1，每回合最大步數為 100 步，並每 25 步統計一次達成率，測試 100 回合。

我們最後使用在前置實驗中所找到的最佳動作組合，並在進行適當微調之後，在觸控板模式開和關的兩種環境下以上述的參數訓練模型，最後測試並比較兩個模型的目標達成率。

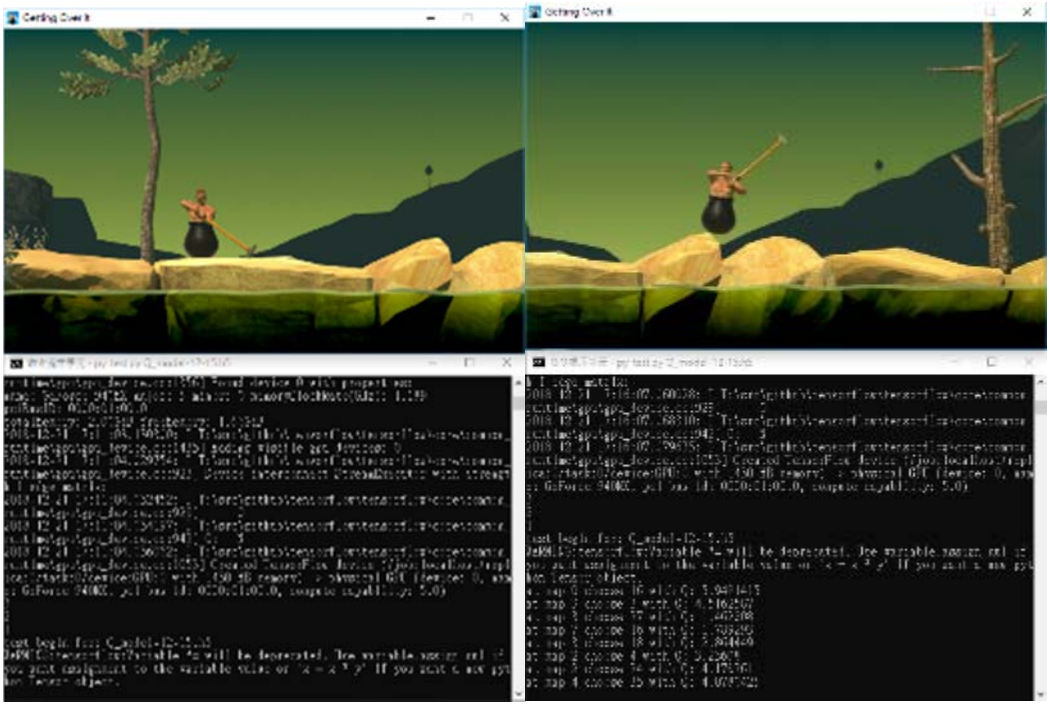

圖 23　遊戲與程式執行畫面。下方的命令提示字元中會顯示當前步數以及選擇的動作編號，還有預期的獎勵數值。

4.2 實驗結果

表 2　　在兩種環境下訓練的模型目標達成率

遊戲設定	目標畫面	25 步	50 步	75 步	100 步
觸控板模式開	樹後(#11)	57%	85%	95%	95%
	山下(#35)	21%	54%	74%	85%
觸控板模式關	樹後(#11)	18%	36%	53%	62%
	山下(#35)	3%	16%	24%	37%

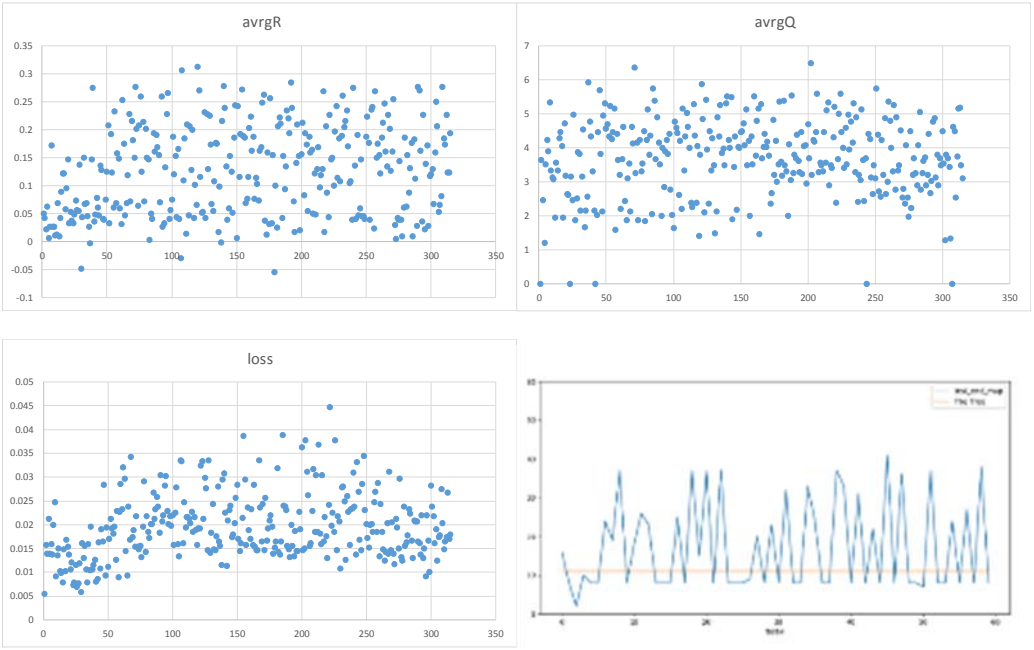

圖 24　　觸控板模式關的訓練數值圖。左上：每回合的平均獲得獎勵，右上：每回合的平均最大 Q 值，左下：訓練的 loss 變化，右下：每回合結束時所在 progress list 的畫面編號。

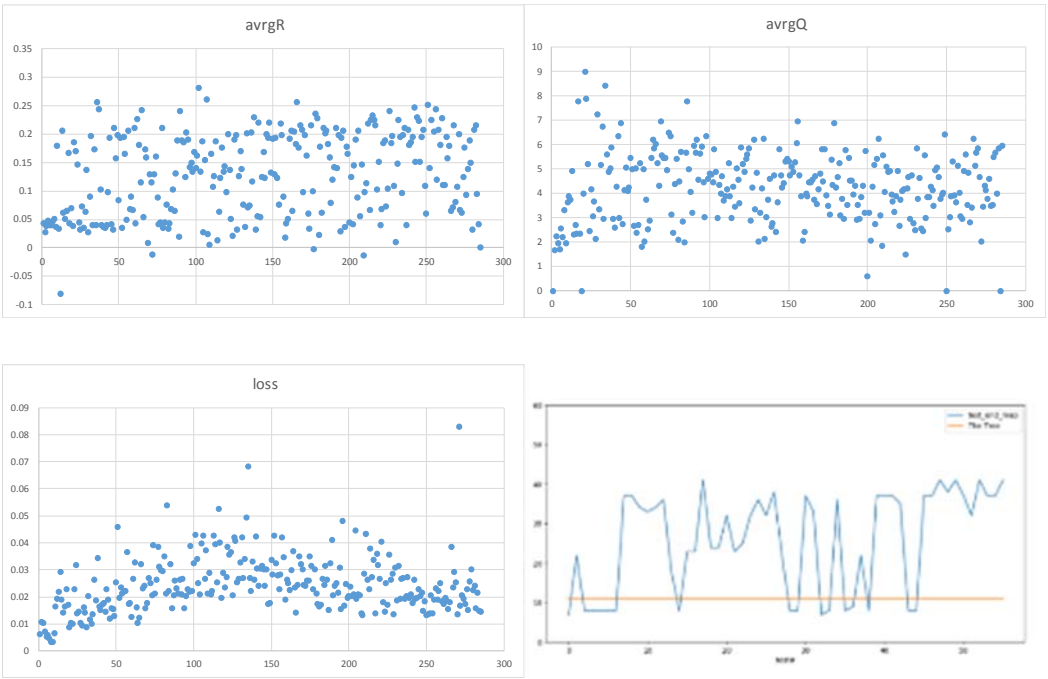

圖 25　觸控板模式開的訓練數值圖。左上：每回合的平均獲得獎勵，右上：每回合的平均最大 Q 值，左下：訓練的 loss 變化，右下：每回合結束時所在 progress list 的畫面編號。

4.3 數據分析

從表 1 可以發現在觸控板模式為開啟的設定下，所訓練的模型通過目標的機率較高，而且能用更少的步數抵達的機率也提高不少，顯示研究者選擇出的動作組合，以及程式操控的方式較適合該設定。此外從表格中可看到當前有達到本實驗的預期目標：通過第一個障礙－－樹的機率高達 95%，可證明它確實有學習到如何遊玩該遊戲。不過，從數據中也發現到在 75 步時機率逼近 95%後就不再上升的情形，顯示它並沒有學到在少數某些情況下如何通過障礙。而實驗的另外一個目標：到達山下，於 100 步到達的機率在最好的設定下也有 85%達成率。

從圖 13 和圖 14 的比較中，我們也可再次發現觸控板模式開的狀態下，平均獎勵、平均最大 Q 值、loss 等都較為密集、收斂，可看出在該模式下訓練的模型的確比較好。另外從圖中也可以看到平均獎勵、平均最大 Q 值等不像他人成果有很漂亮的曲線，顯示這個模型應還有更進步的空間。

五、 結論

5.1 本專題成果與貢獻

成果方面，本專題使用 python 及深度強化學習方法實作能玩遊戲的程式，最初預期目標為能成功運作並遊玩此遊戲，期望它能學會如何通過第一個障礙物。而最終，訓練好的模型能夠順利遊玩遊戲，通過障礙超過預期目標的機率也很高，並且能夠前往遊戲中更後方的部分，有成功達成基礎目標。然而，即便能夠前往更後面的部分，目前通過率仍不能達到 100%，此外對於整體遊戲流程而言，它能抵達的部分仍有些少，對於這些不足之處，我們認為有以下幾點需要改進：

1. 時間問題

   經過數次訓練後，我們發現達成基礎目標後每多一些進展所需的訓練時間便大幅增加，若有充足時間讓它能如論文中一樣訓練夠久，應該能有更多的進展。

2. 專業能力

   研究者的專業能力仍明顯不足，大部分只能按照論文及其他來源提供的方式照做，使用的模型及演算法仍有不小進步空間。

3. 動作組合

   本專題目標遊戲的動作操作並非像論文中選擇的遊戲那般都是固定的按鍵操作，雖然我們盡量選用了較有效且常用的操作模式，但是也可能限制了模型發揮的可能性，還有許多可再進一步探討的空間。

4. 獎勵計算

   作為強化學習中最重要的獎勵計算方法，好的方法能夠讓模型訓練的更有效率，雖然這次設計的方法對這款遊戲在這次的實驗中是有效的，但是應該可以有更普遍性的計算方法。

貢獻方面，由於遊戲中通常都有明確的分數表示當前遊戲進展，使訓練模型時可以直接以該分數作為獎勵值計算，但是本專題的目標遊戲並沒有這個機制，因此研究者必須重新設計一個獎勵計算方法，目前所選用的方法雖然簡單暴力，但卻是我們嘗試的幾種中最有效的一種，這個獎勵計算方法可供這類沒有明確分數的平台通關類遊戲進行參考。

5.2 感想與未來工作

這次的專題讓我們學到許多事情。首先選題目非常重要，因為機器學習這部份研究者在之前的課堂或是生活經驗中幾乎沒有接觸過，只因為是近年熱門起來的領域，在有些好奇的情況下就投身其中，導致這次所選的題目對於我們是不小的挑戰，因此研究者一開始先選修相關課程並研究相關資料、補充知識，有了一定了解後在開始實作。這一路走來並不是一帆風順，初期一直難有好的進展，之後有了進度後又發現只是單純運氣好的曇花一現而非真正的進度，不過在研究者多方嘗試各種方法以及調整相關參數後，終於找到方法讓它順利的學習起來，從這些挫折與困難中，我們對強化學習有了更深的認知。

雖然本次專題到此就結束了，但我們希望能夠讓它更近一步，所以在此羅列了未來待進行的工作與目標：

1. 通關遊戲

   正如研究動機所言，本專題希望做出通關目標遊戲的程式，雖然目前受限時間不能完成，但是希望之後能讓它成功通關。

2. 玩得比人類更好

   這個遊戲的特點就是困難，所以希望能改進當前算法讓它能表現得比人類更好，如此一來能反過來讓人學習如何做得更好。

參考文獻

[1] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., & Riedmiller, M. (2013). Playing atari with deep reinforcement learning. arXiv preprint arXiv:1312.5602.

[2] Watkins, C.J.C.H. (1989) Learning from Delayed Rewards

[3] 維基百科. 強化學習. 2019 年 01 月 08 日取自 https://en.wikipedia.org/wiki/Reinforcement_learning

[4] Miller, W. T., Werbos, P. J., & Sutton, R. S. (Eds.). (1995). Neural networks for control. MIT press.

[5] Konda, V. R., & Tsitsiklis, J. N. (2000). Actor-critic algorithms. In Advances in neural information processing systems (pp. 1008-1014).

[6] 維基百科. AlphaGo. 2019 年 01 月 10 日取自 https://zh.wikipedia.org/wiki/AlphaGo

[7] Evans, R., Gao, J. (2016) DeepMind AI Reduces Google Data Centre Cooling Bill by 40%. 2019 年 01 月 10 日取自 https://deepmind.com/blog/deepmind-ai-reduces-google-data-centre-cooling-bill-40/

[8] Popov, I., Heess, N., Lillicrap, T. P., Hafner, R., Barth-Maron, G., Vecerik, M., ... & Riedmiller, M. (2017). Data-efficient Deep Reinforcement Learning for Dexterous Manipulation.

[9] 維基百科. Getting Over It with Bennett Foddy. 2019 年 01 月 08 日取自 https://en.wikipedia.org/wiki/Getting_Over_It_with_Bennett_Foddy