東海大學資工系研究所

碩士論文

指導教授:陳隆彬

多重代理人之策略競爭遊戲之強化學習方法

研究生:王宇軒

中華民國 108 年 01 月 07 日

東海大學碩士學位論文考試審定書

東海大學資訊工程學系 研究所

研究生 王 宇 軒 所提之論文

多重代理人之策略競爭遊戲之強化學習方法

經本委員會審查，符合碩士學位論文標準。

學位考試委員會

召 集 人 _____ 簽章

委 員 _____

_____

_____

指 導 教 授 _____ 簽章

中華民國 108 年 1 月 10 日

論文摘要

本篇論文針對研究目標設置一個環境，觀察代理經由 PPO 及 Curiosity 在環境中的交互情況，使代理經由訓練表現戰術行為，並採取優秀的策略取得勝利，並調整不同參數或加入分散式處理技術已達到最佳效果。

目次

圖目錄

第一章:介紹

在遊戲開發中，若想實現互動行為，需要將程式架構規畫為接收層，邏輯層及輸出層等，撰寫及測試極為複雜且依然無法如真實人類般能應對各種狀況，或者要為此付出較高的開發成本，即便如此，復有針對性、迷惑性且分階層的 AI 也較難完全開發。

針對上述問題，我們希望能藉由機器學習，從此簡化遊戲開發中的 AI 問題，使開發者能透過簡單的選擇模組，使怪物、敵方等透過環境產生相對應的”智慧”，甚至能自由選擇智慧的程度。機器學習顧名思義便是讓機器真對環境自我學習並且學會如何作出自己決定的方法，從早期的 Q-Learning 到現在的 DQN、PPO 等，機器通過算法能掌握的狀態與策略行為越來越多，更能應付更高的隨機性，因此在面對我們想解決的問題時，機器學習是一個良好的解決方式。

在這篇論文，我們選擇一個遊戲，建立一個環境比較代理在單獨面對問題及多方代理處在競爭關係時面對問題會產生的變化，並意圖使代理產生我們預期的戰術行為，我們使用近端優化策略算法(PPO)、及好奇心算法(Curiosity)比較代理在不同環境的表現、如何調整獎勵值使代理能表現出偷襲、守株待兔、回避防禦等無法用獎勵、觀察值等參數直接設計之行為。

最後，在已上的基礎下，我們使用分散式的處理方法，加速代理的訓練過程。

第二章:研究背景

AI 的發展:

從 20 世紀前葉開始，來自各領域的學者專家們便不斷討論人工大腦的可能性，1950 年代，圖靈提出著名的圖靈測試:如果一台機器可以透過傳輸設備與人類對話並使人類無法分辨是否是機器，那便是理想中的人工智慧。透過這個假說人工智慧的概念被具現化，也有了發展的方向

1955 年，已逆向搜索為主要策略的”邏輯理論家”誕生，它能模擬人已符號證明定理的思維活動，並成功證明了<<數學原理>>中 52 個定理的前 38 個，有些甚至比人證明的方法更加精巧。

在那之後，人工智慧進入黃金年代，許多具有影響力的分之也開始出現雛形:如使計算機可以通過自然語言(中文、英文等)語人類交流，使複雜的現實物體簡化成積木、網體等的「微型世界」在計算機圖學上取得了重大的突破，然而問題也隨著學者們更加深入的研究而浮現。

1974 年開始，科學家們開始發現機器學習幾個難以突破的問題: (1)只能克服基礎的問題 (2)需要的資料太過大量，沒人知道要多少資料才能讓機器學習到足夠可以應付常識問題 (3)受限於當時的硬體裝置，機器處理問題的速度都太過緩慢。

以上批評最影響的便是影響現代機器學習最深的感知器與類神經網路，因為遲遲無法解決上述問題，這兩門學科消失了十年。

1997 年，深藍擊敗當時的世界棋王，人工智慧的研究又開始火熱起來，隨著理論及硬體設備的發展，感知器及類神經網路又在捲土重來，這次，它從淺層網路演化成深層網路，避免了淺層網路造成的問題，並結合各種機器學習的理論應用於各方各面，如結合強化學習的深度強化學習(EX:Q Learning　DQN)，圖像辨識的 CNN 等。

遊戲 AI 的應用：

在遊戲開發過程中，AI 可以說佔據了相當重要的地位，從場景的自動生成，氣候對玩家造成的影響、怪物因為玩家及環境而作出的反應。

然而，過去的 AI 較為偏向已經寫好的邏輯編程，因此有許多想像中的功能可以說幾乎無法被開發出來，例如開放式遊戲中希望每個怪物都能自己繁衍、產生新特性，例如希望遊戲內 NPC 的行為可以自適應每個玩家的行為。

而隨著人工智慧技術的發展，機器人開始可以自行”進化”，學習外界的環境，在沒有過多刺激的環境下透過自己的好奇心探索，這些複雜的期望似乎也漸漸看到曙光。

在這個章節中，我們介紹深度強化學習所使用的算法及技巧並介紹論文研究的目標主題與採用的算法核心思想。

2-1 研究目標與方法

2-1-1 研究遊戲

圖 1 荒野亂鬥示意圖

(來源:荒野亂鬥官網)

《荒野亂鬥》(英語：Brawl Stars)是由 Supercell 開發和發布的一款免費的手機實時戰略遊戲第三人稱射擊遊戲。

在今日以前(2019/1/22),《荒野亂鬥》下載已經超越千萬人次，在 APP STORE 上的評價達 4.5 顆星;在 Play Store 上的評價達 4.6 星，是個擁有高人氣與評價的遊戲，我們希望能透過機器學習的方式訓練能在該環境機制下流暢與玩家對戰的代理。

2-1-2 遊戲機制

本論文選用遊戲的”寶石爭奪戰”模式作為目標，此模式為三對三的對戰，環境包括了視野遮蔽區、屏障、及定時產生的寶石，玩家可以通過武器攻擊對方玩家，若成功擊殺，該被擊殺玩家身上的寶石便會掉落，玩家需要搶奪寶石並搶先累積到 10 顆寶石並守護一段實現後獲得勝利。

圖 2 荒野亂鬥遊戲示意圖

（來源:荒野亂鬥官網）

2-1-3 預期效果

從上述遊戲機制我們可以發現，環境本身並不複雜，任何玩家都能輕鬆理解，對於機器也是，然而對於機器來說難點在於如何這些環境(包含對方玩家)的所傳達之千變萬化的資訊下找到對應的策略，包括從無到有探索出能令代理獲勝的策略，並且因應環境的改變作出應對，也就是針對環境採取不同的戰術行為，預期代理能在經由訓練後能在對戰中自行採取以下戰術行為:

1.	等待行動
2.	試探性攻擊(預判)
3.	團隊行動
4.	沿牆躲避子彈
5.	獲得一定數量寶石後進行躲閃行為
6.	透過觀察兩方寶石數量，克制吃寶石
7.	固守中央區域
8.	當處於劣勢時，不莽撞
9.	偷襲、利用視野優勢

2-2 相關算法介紹

在介紹本論文使用的算法之前，我們先從較為早期的機器學習算法開始介紹，它們所使用的概念到今天還是被廣泛的使用，甚至可以說今天很多算法是基於這些算法的衍伸，下面我們便說明這些算法邏輯:

2-2-1 感知器

模擬神經系統的算法，將輸入 X1,X2,X3,X4,X5,乘上各自權重 W0,W1,W2,W3,W4 後加總，並將加總結果輸入激勵函數，取得結果後再透過結果更新權重，訓練出能正確判斷結果的感知器。

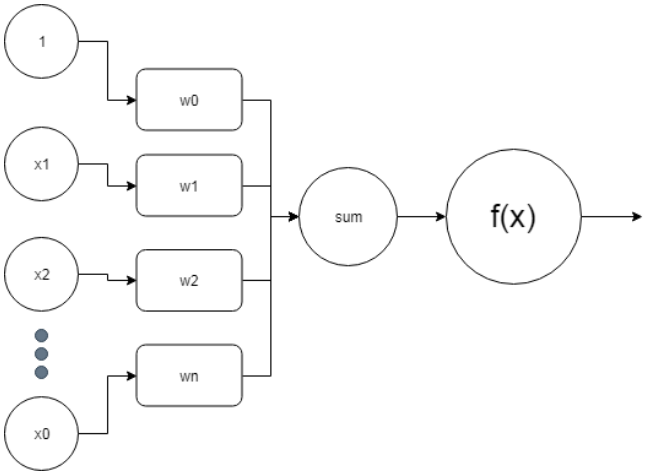

圖 3 感知器示意圖

2-2-2 類神經網路

將多個感知器結合，形成網狀，並透過反向傳播法更新每個感知器權重。

所謂反向傳播，便是這一層網路在更新權重前，會先往前追朔上一層網路的權重並更新它們，如下圖:

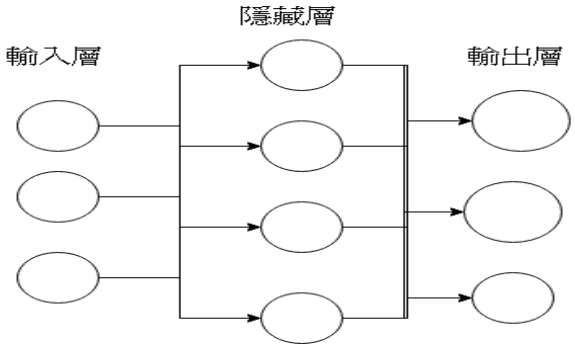

圖 4 類神經網路示意圖

2-2-3 Q-Learning

Q-Leaining 為強化學習很有名的一個算法，其主要概念為，將代理在某狀態下作出的行為評分，並儲存起來，透過下一次的獎勵更新，使儲存的分數漸漸出現差距，透過不斷更新找到最佳解。

當代理處於狀態 S 時有數個選擇，每個選擇以 Q(S,A)代表代理在該狀態選擇 A 的值，代理選則動作後，會將下一個動作最高的 Q 值(稱作 maxQ)作為預想中的獎勵，並比對該動作

表示為 Q(S,A) ← Q(S,A)+a{r+maxQ(S,S,)-Q(S,A)}:

$$Q_0(s,u) = 0 \qquad \text{for all } s \in S,\ u \in U$$

$$Q_{t+1}(s,u) = \begin{cases} Q_t(s,u) & \text{if } s \neq s_t \text{ or } u \neq u_t \\ r(s,u) + \beta \cdot \max_{u' \in U} Q_t(\delta(s,u),u') & \\ & \text{if } s = s_t \text{ and } u = u_t \end{cases}$$

圖 5 Q-learning 算法

(來源: Lauer, M., & Riedmiller, M. (2000). An algorithm for distributed reinforcement learning in cooperative multi-agent systems. In In Proceedings of the Seventeenth International Conference on Machine Learning.)

而我們常說的深度學習，便是在類神經網路的基礎下增加隱藏層，使其擁有更高辨識能力，並結合各種演算法達到效果，如 DON 便是 Q-learning 加上深度學習。

2-3-4 深度神經網路

來自類神經網路概念，第一層為輸入層，中間為隱藏層，最後一層為輸出層 ，與傳統類神經不同的是，深度神經網路的隱藏層不只一層，但也不是隱藏層層數越高越精確，事實上，如果太多隱藏層，可能會導致最後輸出的結果與答案誤差太過為小，導致無法訓練。

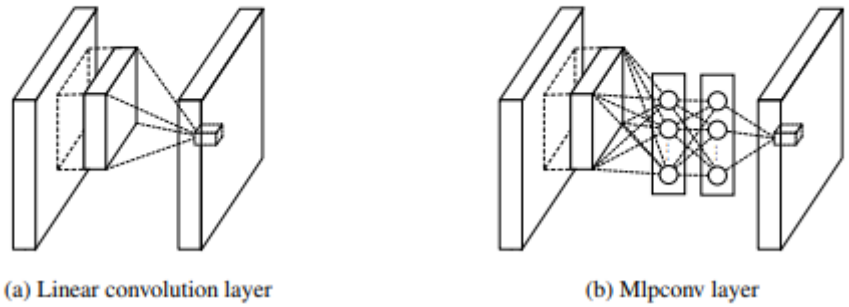

圖 6 深度神經網路示意圖

（來源: Lin, M., Chen, Q., & Yan, S. (2013). Network in network. arXiv preprint arXiv:1312.4400.）

2-3 論文使用算法

2-3-1 Policy Gradient

Policy Gradient 是強化學習的重要方法之一，也是本論文採用之 PPO 算法的基礎，因此在介紹 PPO 算法前必須先詳解 Policy Gradient 的核心與更新，其根本概念在於 AI 學習的不是獎勵而是隨機選擇策略:

所謂策略，便是當代理在某個狀態 S1 採取某個行動 A1，進而使環境發生改變給出狀態 S2，然後代理再採取動作 S2 以此類推直到 t 步驟結束，此一連串環境所提供的狀態 S，以及依據環境所選擇的行動 A 的組合。
若以π代表策略，可表示為:

$$\pi = (s1, a1, s2, a2 \dots\dots\dots\dots\dots st, at)$$

而在模型(已 p 代表)裡，每個狀態與行動都有其發生的機率，因此我們可以求出在該模型 p，某個參數(已θ表示)的狀況下，某個策略發生的機率為每個狀態發生的機率乘上在該狀態裡採取某行動的機率，以此類推，可表示為:

$$p_\theta(\pi) = p(s1)p_\theta(a1|s1)p(s2|s1, a1) \dots\dots$$

而藉由機率隨機選擇策略，評比好的策略後將其機率提升，便是此算法的本質。

為了更新機率，Policy Gradient 採取了根據期望獎勵(已 R 表示)進行梯度上升的算法，之所以採用期望獎勵而不是固定獎勵則是因為 Policy Gradient 無論是環境給的狀態 S 或是行為 A 皆有其機率，因此獲得的獎勵也是有機率的。因此，我們將策略在某參數算法中獲得的所有獎勵乘與該策略的機率，便是其期望獎勵，可用數學式表示為:

$$R_\theta = R(\pi)p_\theta(\pi)$$

而我們可以藉由$\nabla f(x) = f(x)\nabla \log f(x)$，取得獎勵期望值的梯度，導出下列公式:

$$R(\pi)\nabla \log p_\theta(at|st)$$

，接著梯度上升的算法更新參數，如下列公式:

$$\theta \leftarrow \theta + r\nabla R_\theta$$

由上述兩個公式我們可以得出，R 為總獎勵，$\nabla \log p_\theta(at|st)$ 差異度，因此當差異比較大的時候也會較大幅度的更新，也會根據 R 為正 OR 負作負更新或正更新

以上，便完成了 Policy Gradient 演算法的核心思想與更新，而在這個算法在實作中，可能會遇到一些難以解決的狀況，因此在實作上通常會會再加上一些細節:

(1) 在現實中或許多常見的遊戲中，很多狀況都是只有正向獎勵，而沒有負獎勵，例如機器人手臂無論以哪種東西拿取到物品都是正向行為，區別只在於效率，為了使代理在這種狀況依然可以達到好的訓練效果，我們可以將獲得的獎勵減掉平均獎勵，人為製造正負。

$$(\mathrm{R}(\pi)-\mathrm{b})\nabla \log p_\theta(at|st)$$

(2) 其次 Policy Gradient 算法更新使用的獎勵是從整個策略獲得的獎勵，然而該策略不一定每個動作都是好的動作，因此，我們將某個動作之後獲得的獎勵作為某個動作獎勵，此外，因為某個動作實際上並不會影響太久之後的動作，我們將該動作獲得的獎勵乘與某個參數，借此限制，可以用數學式表示為:

$$\sum_{t}^{Tn} r^{t'-t} r_{t'}^{n} \nabla \log p_\theta(at|st)$$

有的人也會將$r^{t'-t}r_{t'}^{n}$稱為優勢函數A^{θ}，藉由上述算法更新，帶給代理好改變的行為機率便會越來越高，使代理在任何狀態下都能選擇較好的行為。

2-3-2 PPO

PPO 是個基於 Policy Gradient 的算法，其基本概念便是把 Policy Gradient 從 On-Policy 改成 Off-Policy 型態，其中，所謂 On-Policy 指的是訓練算法時，使用代理實際在環境中採取的行為和反應進行更新;而 Off-Policy 則是使用另一個代理儘可能的收集不同數據，而後使用收集到的數據進行訓練，等訓練差不多之後再重新收集一次數據。

而從 On-Policy 改成 Off-Policy 型態時，演算法因為是透過另一個資料集進行學習，學到的策略更新容易與實際上的情況會有所誤差，為了避免誤差，我們會採用下列數學公式，計算差異並且限制更新的幅度:

$$\int f(x)p(x)dx = f(x)\frac{p(x)}{q(x)}$$

將此數學公式代入 Policy Gradient 演算法中，就能導出下列公式:

$$\frac{p_\theta(at|st)}{p_{\theta k}(at|st)}A^{\theta k}\nabla \log p_\theta(at|st)$$

除了上述的限制外，PPO 算法也引入 Clip 算法對新舊數據的更新進行了更進一步的限制，其限制分為兩種狀況，其一，當優勢函數$A^{\theta k}$為正時，壓縮小於限制(通常為 1-e)的$\frac{p_\theta(at|st)}{p_{\theta k}(at|st)}$直到大於等於限制，其二，當優勢函數$A^{\theta k}$為負時，壓縮大於限制(通常為 1+e)的$\frac{p_\theta(at|st)}{p_{\theta k}(at|st)}$直到小於等於限制。

介由上述的兩個步驟，便能使 Policy Gradient 轉換為 PPO 算法，大幅度提升訓練的效率。

2-3-3 Curiosity

在運用各類機器學習算法進行訓練時，時常會遇到代理取得一點小成績便固守不前，導致整體成績處於一個極低的水平，這是因為在訓練中，代理只會收到由環境給予的向量值，這個向量值是由環境的創建者設定的，如果在一個獎勵過於稀疏的環境，在遲遲無法取得獎勵的情況下，代理便很可能選擇就此止步，就算它某一次獲得的獎勵較高也會認為是偏差無視掉。

因此，就像人類除了外部刺激以外，也常需要內部的驅動一樣，Curiosity 算法的原理便是將代理獲得的獎勵從原本的獎勵拆為兩個部分，其一為原本獲得之獎勵，另一個則是基於環境熟悉度獲得的獎勵，一般的實作方式是透過預測接下來環境給予的變化及獎勵計算出對於環境的熟悉度，若越不熟悉則該評分越高，反之，則越低，該算法核心可用下列數學式表示:

$$\mathrm{r} = r_t^i + r_t^e$$

2-4 分散式處理系統

分散式系統是用來處理當資料過大，一台機器無法負荷，或是一台機器供給的效能無法滿足時所用的技術，透過路由器、DNS 輪巡、平行處理等方法讓伺服器了解彼此的資訊，並分擔彼此的計算亮。

機器學習是靠著 GPU 或 CPU 進行連續的運算達到想要的效果 ，然而，有時限於硬體效能，需要花費過多的時間，此時便常採用分散式處理系統將不同任務交給不同硬體來計算，增加運算效率。

第三章:環境

3-1 環境搭建

3-1-1 UNITY

Unity 是一款由 Unity Technologies 研發的跨平台 2D / 3D 遊戲引擎，可用於開發 Windows、MacOS 、 Linux 、PlayStation、XBox、Wii、iOS、Android 等 主要遊戲平台的遊戲。

除了主要遊戲平台外，UNITY 近年也結合 tvOS、Oculus Rift、ARKit 等技術被廣泛運闋於建築視覺化、實時三維動畫、AI 研究、VR 等領域。

3-1-2 ML-Agent

Unity 機器學習代理（Unity Machine Learning Agents）是 Unity 推出的機器學習套件。

這款套件基於 Unity 本身出色的環境搭建功能，結合 Python 及 Tensorflow 的機器學習功能，讓使用者可以快速實現視覺內容和現實物理的複雜行為、研究機器人技術、自動駕駛技術和其他工業應用實施大規模的平行訓練制度，更可以將機器學習應用在遊戲中，如使用代理動態地調節遊戲難度等級，自我成長的 AI 等等。

ML-Agents 包含三个高级组件：
(1)環境(Learning Environment) - 其中包含 Unity 场景和所有游戏角色。
環境中包含三個重要部分:

I. Agent

透過繼承 Agent 的腳本我們可以規定我們想要觀測的值，執行我們規定要執行的動作，將腳本附加到 Unity 環境中的某個 GameObject 並執行後，腳本就可以將該 GameObject 觀測到的觀察值及動作回傳到我們希望他回傳的 Brain，並且執行該 Brain 要他執行的動作，此外 Agent 也負責回傳每個動作的 reword，可以把 Agent 理解成線上遊戲的角色，玩家可以操縱角色進行任何活動，Agent 則會告訴玩家他看到的東西及這個動作取得的分數。

II. Brain -

Brain 是決定 Agent 該作什麼動作的決策中樞，Agent 將觀測到的值傳到 Brain 內，Brain 透過觀測到的值及訓練過後的數據告訴 Agent 要如何行動，其中，Brain 分為四種模式，Unity 透過這四種模式實現訓練和實際運作中的預測:

Player - 使用鍵盤或控制器的實際輸入進行決策。這種模式下就等於玩家自己玩操控，Brain 並不會參與操控 Agent。

External - 這種模式下，Brain 將收集的觀測結果和獎勵通過 External Communicator 轉發給 Python API，Python API 將觀測值和獎勵等傳給 Tensorflow 的算法，並將 Tensorflow 返回的動作回傳給 Brain，操控 Agent 採取相應動作，而在一次次訓練的過程中，tensorflow 將算法對映的結果儲存起來，便成了決策模型。

Internal - 將訓練好的 tensorflow 模型放進 UNITY 環境，當 Brain 收到觀測值，直接透過模型學到 policy 進行決策，告訴 Agent 該做的動作

Heuristic - 使用程式寫好的邏輯進行決策，即傳統遊戲的 AI 寫法，可以透過這個模式測試程式邏輯，也可以比較程式邏輯及透過訓練 AI 兩者的表現相較如何。

譬如同一個場景內，我們可以放兩個應該做同一件事情的 AI，已射擊遊戲而言，便是已兩個 AI 代表兩個玩家，其中一個 brain 使用程式邏輯，另一個使用訓練 AI，觀察兩者表現

值得注意的是，一個 Brian 不一定只能對應一個 Agent，其實只要相似行為，相似觀察值的 Agnet 都可以關連到同一個 Brain，譬如射擊遊戲裡如果有兩個玩家，這兩個玩家都可以關連到同一個 Brain，然而如果其中一個玩家操控類似飛機這樣的載具時，則不能關連到同一個 Brain。

此外，除了 External Brain，我們也可以配置 Internal、Player 和 Heuristic 類型的 Brain， 使它們也能通過 External Communicator 將觀測結果及獎勵值等發送給 Python Api，實現更靈活的訓練模式

III. Academy –

Academy 負責協調整個環境的參數，諸如訓練的最大步數、環境初始化、Agent 移動之後呼叫的動作、時間比例、Quality Level 等、此外、負責與 Python 溝通的 External Communicator 也位於 Academy 內

3-1-3 Tensorflow

TensorFlow 是一個開源軟體庫，用於各種感知和語言理解任務的機器學習。目前被 50 個團隊用於研究和生產許多 Google 商業產品，如語音辨識、Gmail、Google 相簿和搜尋，其中許多產品曾使用過其前任軟體 DistBelief。

TensorFlow 最初由 Google 大腦團隊開發，用於 Google 的研究和生產，於 2015 年 11 月 9 日在 Apache 2.0 開源許可證下發布。

3-1-4 溝通機制

Python APi 和 Untiy 內的 External Communicato 使用 Socket 作為進程間的通信，Python 唯 Server，Unity 為 Client，訓練好的模型則通過 Tensorflow 保存(Bytes 文件)。

在訓練結束後，可將模型放入 Unity，透過 Brain 來判斷 Agent 該如何與環境互動。

3-2 驗證環境:

在過去，機器學習的算法無法在過於複雜的選擇下收斂，以現行盛行的「PUPG」為例，玩家需要在一個獎勵稀疏的環境下尋找資源、逃離或消滅追殺者;此外，過去的算法也較無法理解事件的關連，或者需要很多時間及記憶，本篇論文選用 PPO 算法，也正是因為其訓練的高效率與，為了驗證 PPO 算法的效能與決策能力，因此設計一個類似「PUPG」的場景針對這些特性設計實驗。

此場景主要流程分為幾個實驗階段: (1)尋找通往終點的道路 (2)在通往終點的過程中學習自保 (3)收集並學習啟動打開通往終點的大門

建立一個訓練環境環境的左右兩側為資源區，隨機產生資源，中間是主地圖，包括機器人無法通過的圍牆、關閉的門以及一個啟動器，代理需要收集到 10 個資源才有辦法使用啟動器打開通往終點的門，此外，場景中包括會使用武器的敵方 AI，被攻擊到則訓練結束。

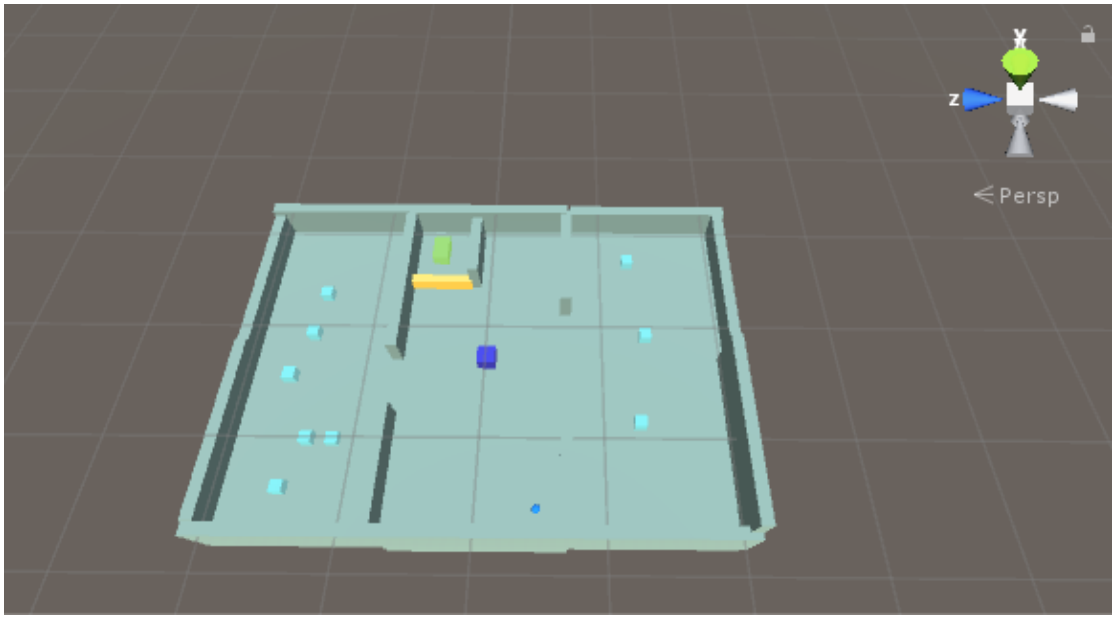

圖 7 驗證環境俯瞰圖

綠色的為終點，黃色的隔板為門，藍色為門的啟動器，此三項都會隨機出現在地圖中的某個位置,淡藍色為資源，獲得一個資源，紅色為敵方 AI，搜索代理並對攻擊代理

我們考慮兩個代理從不同地方出生，兩個代理都有相同的三種武器可以使用，武器 A 攻擊高但冷卻時間也高，武器 B 攻擊中等但冷卻較快，武器 C 冷卻時間最高，但在一定範圍內一定可以擊中，代理必須互相干擾並取得環境裡的資源，開啟大門並進入終點。

3-3 荒野亂鬥場景:

經由第一個場景驗證後，我們已當紅的手機遊戲—"荒野亂鬥”為研究對象，訓練能針對環境、玩家做出恰當行動並取得勝利之代理人，並更進一步始代理人在固定的機制中配合其他代理人產生團隊合作行為。

圖 8 荒野亂鬥遊戲示意圖

（來源:荒野亂鬥）

荒野亂鬥的機制為典型的槍戰遊戲，雙方玩家(各三人)需要搶奪場景中隨著時間出現的寶石，使用槍械及技能攻擊敵人致死後，敵方人攜帶的寶石便會掉落在場景中供人自由撿取，首先獲得 10 個寶石的一方若能在倒數秒數內維持 10 顆寶石便獲得勝利，此外，場景中有各種遮蔽物，只要待在遮蔽物後方，便不會受到攻擊。

經由驗證環境學習到的經驗，我們透過拆分一個 BattleControl 腳本控制環境，提高研究的便利，腳本交互如下圖表。

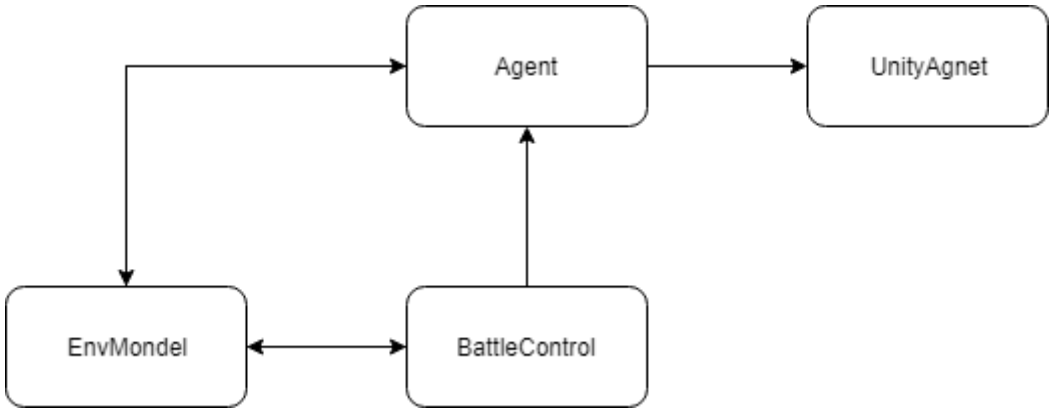

圖 9 架構示意圖

名稱	作用	常調用函式功能
EnvModel	環境模型、光照、角色模型等	
BattleControl	控制 EevModel、提供 Agent 參數	產生寶石、隨機物件位置、初始化環境、環境配分等
Agent	收集獎勵、環境狀態(包括從 BattleControl 讀取的參數等)、操控代理	收集觀察值(觀察值會核並轉換成狀態)、決策行為等
UnityAgnet	收集 Agent 獲得的所有參數，進行訓練	參數轉為狀態、算法更新、PYTHON 與 C#接口等

而在我們的模擬場景中，機制基本上完全複製荒野亂鬥的設計，除為了加快實驗速度，我們在實驗場景中取消技能及攜帶 10 個寶石的機制，改為在時限獲得較多寶石的代理人團隊勝利。

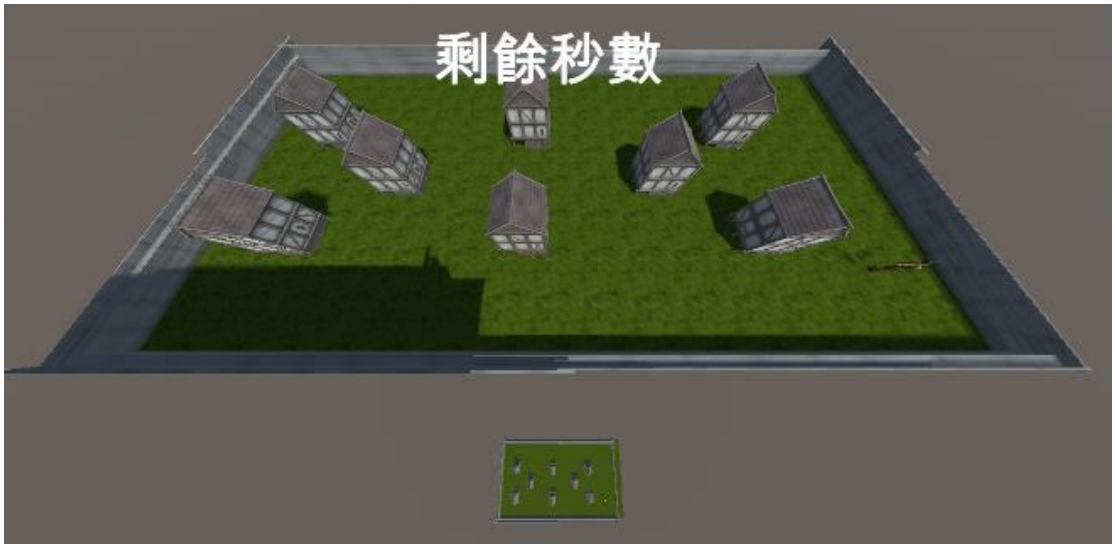

圖 10 荒野亂鬥訓練場景俯瞰圖

第四章:訓練代理人

在機器學習中，環境、參數和算法的設置為重中之重，在這篇論文裡，我們總共有兩個環境及兩個希望代理人達到的目標，分別為簡單環境裡代理可經由學習學會打開機關並到達目標，及第二個環境，兩個代理人團隊必須在固定條件下利用環境及配合隊友達成勝利目標，並且能與玩家抗衡。

4-1 訓練方法

在第一個環境裡，目標相對簡單，我們預設使用 PPO(近端優化策略算法)訓練代理人，在代理周遭每 15 度設置 10 單位射線，使代理能觀察周遭物體但沒辦法綜觀整個環境，並針對各個環境組件回饋觀察值(共 62 組觀察值)，使用器具及行動共 7 個行為，此外，在預先搭建的環境裡，代理的表現確並不如預期，因此針對訓練環境及參數作出改進及調整。

實驗環境(一):

觀察值	獎勵(reword)	行為(state)
30 度仰角 30 度	進入門(+1000)	往前
60 度仰角 30 度	碰觸敵人(-200)	往左
90 度仰角 30 度		往右
120 度仰角 30 度		往後
180 度仰角 30 度		使用策略 1
210 度仰角 30 度		使用策略 2
240 度仰角 30 度		使用策略 3

荒野亂鬥環境:

在第二個環境裡，環境相對困難，針對這個訓練環境，有兩個主要腳本控制並實現不同的訓練方式，第一個腳本為中央控制腳本，負責計時、返回團隊分數、給予團隊獎勵、返回代理人團隊所屬等等，另一個腳本為代理本身腳本，負責回饋觀察值、採取行動、碰觸目標等等。此外，觀察值及行為主要為:周遭每 15 度設置 10 單位射線及 30 單位射線，使代理能觀察周遭物體但沒辦法綜觀整個環境，並針對各個環境組件回饋觀察值(共 234 組觀察值)，使用武器及行動共 4 個行為。

透過腳本的交互，實現並比對幾種代理與環境交互的型態:

1. 在較短的秒數內，使代理獲得寶石便給與少量獎勵，此外，當一方隊伍被殲滅便始獲勝隊伍獲得獎勵，結束一次訓練。

2. 在較短時間內，不給予任何獎勵作為激勵，只給予遊戲時間結束後寶石較多隊伍獎勵

3. 在較短的秒數內，使代理獲得寶石便給與少量獎勵，此外，等到遊戲時間倒數結束後，統計給予獎勵，結束一次訓練。

4. 在較短的秒數內，使代理獲得寶石便給與少量獎勵，此外，殲滅隊伍及倒數時間後寶石較多隊伍皆能獲得獎勵

5. 在較長時間內，使代理獲得寶石便給與少量獎勵，此外，當一方隊伍被殲滅便始獲勝隊伍獲得獎勵，結束一次訓練。

6. 在較長時間內，不給予任何獎勵作為激勵，只給予達成目標後的獎勵

7. 在較長時間內，給與較多環境激勵(如攻擊對手、獲得寶石)，只給予達成目標後的獎勵，

綜合以上等等各種訓練情境及不同獎勵設置方法，如獎勵高低、失敗安慰獎勵，平手獎勵、增加獎勵、固定獲得獎勵等方式進行訓練。

4-2 訓練算法

我們採用的 PPO 算法來自於 TRPO 算法的改良，其與傳統策略梯度算法差別在使用 clip 算法限制策略梯度的過大與過小，採用這種方法可以使訓練結果容易泛化，並使代理不會過於偏向某個策略，將梯度縮小在 1-e,1+e 之內，但槍戰遊戲理論上可以固定採取某些策略，因此，我們將 clip 的限制改為(1-e,z(1+e))，進行代理之訓練。

4-3 訓練共通問題與改善

4-3-2 難以收斂

單場景的訓練容易使代理不容易泛化且收斂困難，使用多場景並隨機每個場景組件的位置、數量、啟動條件等參數，解決此問題。

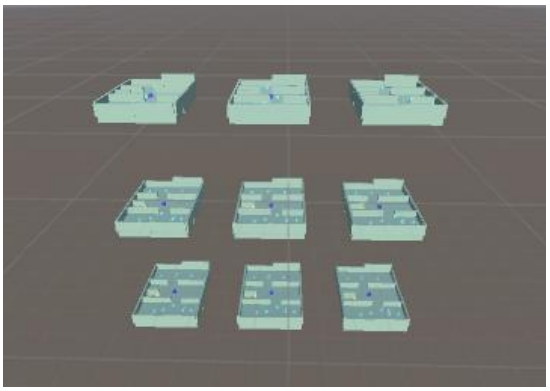

圖 11 多場景並行訓練示意圖

4-3-2 步長

代理在訓練過程中常有閒晃、不知所措的行為，或是過了很久才到達目的地，為使代理學習已有效率的方式探索，當該場景任務組件獎勵值為 R，若總共有 I 個任務組件，總步數為 S，每次行動扣除 X= $\sum_{k=0}^{I}(R^k)/S$的獎勵值，使得代理能以更有效率的方式行動。

4-3-3 稀疏獎勵環境

預設環境為一獎勵稀疏環境，導致代理出現停留原地、只攻擊 AI 敵人等行為，加入 Curiosity 算法使代理擁有內在激勵，使代理活動範圍擴大到整個環境，並且更能嘗試不同選擇，使最後訓練效果最佳化。

4-3-4 決策僵化環境

為使訓練結果可以泛化，我們引進隨機的概念，環境中的組件都會隨機出現在任何位置，這個機制意外讓我們發現能促進代理的探索行為，但過度的隨機也會使收斂更加困難。

4-3-4 訓練時間過長

同時訓練多個環境有助收斂，但訓練速度會隨增加的環境而下降，透過 tensorflow 將大張運算圖分割成多張運算子圖，經由通信交付不同主機進行運算，如圖

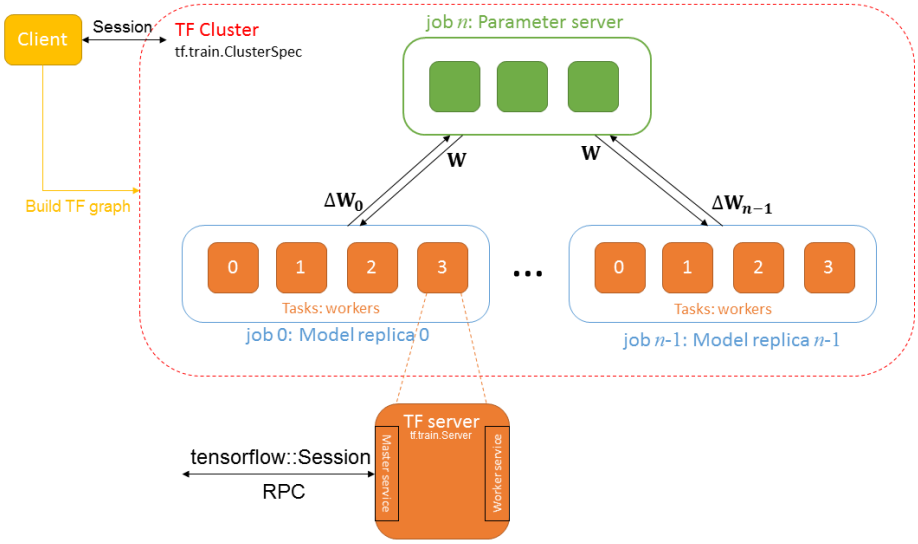

圖 12 Tensorflow 分散式架構途

（來源:tensorflow 官方文件）

第五章:實驗成果與細節

透過不同環境、參數和算法的設置，我們成功使代理達到預期效果，在不用傳統方法下，透過訓練使代理能自然的與環境互動並與隊友合作。

驗證環境:

代理學會在環境內尋找終點，並且學會在發現敵人時攻擊，設定終點獎勵值較高時，尋找終點為重，代理表現的不戀戰，只有接近敵人 AI 時才會進攻。

此外，我們發現導入競爭行為始代理人互相競爭時展現更多人類化的行為，其中一個代理學會在原地等待門開啟後搶先，另一個代理學會先搜尋敵方代理攻擊，而代理也會因 AI 的不同而學習到不同的行為，與制式化 AI 相比，多代理的競爭明顯使最後訓練出來的代理有更多靈活性。

在這個場景，我們明顯發現競爭類型遊戲對於訓練代理來說更加困難，無論是直接設定目標獎勵，或者給予稀疏獎勵都不足以使代理與環境做出良好互動，必須設計不同環境、改善訓練算法與獎勵方法才有辦法使代理達到預期目標，但相對的競爭訓練也容易使代理有更多戰術行為

荒野亂鬥環境:

透過代理的互相競爭訓練，我們實現了代理人的戰術行為，包括在時限未到前躲在隱蔽地點等待，埋伏等行為(如下圖表整理):

圖 13 代理等待示意圖

1.	等待行動
2.	試探性攻擊(預判)
3.	團隊行動
4.	沿牆躲避子彈
5.	獲得一定數量寶石後進行躲閃行為
6.	透過觀察兩方寶石數量，克制吃寶石
7.	固守中央區域
8.	當處於劣勢時，不莽撞
9.	偷襲、利用視野優勢

我們驚喜的發現，預期中只有人類玩家操控的所有行為，都可以通過訓練使兩方代理在互相競爭的情況下自行學會，如沿著牆壁躲避敵方子彈，預判對方行蹤並埋伏、獲得足夠寶石後不貪多而是盡量使對方無法獲得寶石等行為，也發現通過獎勵的變化能輕易使代理實現不同行為模式(安穩、激進等)。

第六章:結論與未來展望

我們建置了一個任務模擬場景，並讓單代理及多代理在環境中執行任務。我們透過認物模擬場景實現了單代理及多代理在不同狀況下展現的不同特質，以及獎勵及參數的差異。

在這次研究中，我們比較專注於觀察希望代理能表現出我們想要的特徵，但容易忽視代理經由學習表現出的其他行為，也比較局限於單一環境，希望能在未來持續對模型泛化及開發清楚的可視化系統，打造出能展現讓人吃驚行為及泛用在各個遊戲的 AI 開發者插件。

第七章:參考文獻

Bansal, T., Pachocki, J., Sidor, S., Sutskever, I., & Mordatch, I. (2017). Emergent complexity via multi-agent competition. *arXiv preprint arXiv:1710.03748*.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

Girija, S. S. (2016). Tensorflow: Large-scale machine learning on heterogeneous distributed systems.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., & Zaremba, W. (2016). Openai gym. arXiv preprint arXiv:1606.01540.

Lowe, R., Wu, Y., Tamar, A., Harb, J., Abbeel, O. P., & Mordatch, I. (2017). Multi-agent actor-critic for mixed cooperative-competitive environments. In Advances in Neural Information Processing Systems (pp. 6379-6390).

Atkeson, C. G., Hale, J. G., Pollick, F. E., Riley, M., Kotosaka, S., Schaul, S., ... & Kawato, E. (2000). Using humanoid robots to study human behavior. IEEE Intelligent Systems and their applications, 15(4), 46-56.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., ... & Dieleman, S. (2016). Mastering the game of Go with deep neural networks and tree search. nature, 529(7587), 484.

Duan, Y., Chen, X., Houthooft, R., Schulman, J., & Abbeel, P. (2016, June). Benchmarking deep reinforcement learning for continuous control. In International Conference on Machine Learning (pp. 1329-1338).

Heess, N., Sriram, S., Lemmon, J., Merel, J., Wayne, G., Tassa, Y., ... & Silver, D. (2017). Emergence of locomotion behaviours in rich environments. arXiv preprint arXiv:1707.02286.

Schulman, J., Levine, S., Abbeel, P., Jordan, M., & Moritz, P. (2015, June). Trust region policy optimization. In International Conference on Machine Learning (pp. 1889-1897).

Kang, B., Jie, Z., & Feng, J. (2018, July). Policy optimization with demonstrations. In International Conference on Machine Learning (pp. 2474-2483).

Watkins, C. J., & Dayan, P. (1992). Q-learning. Machine learning, 8(3-4), 279-292.

Gambardella, L. M., & Dorigo, M. (1995). Ant-Q: A reinforcement learning approach to the traveling salesman problem. In Machine Learning Proceedings 1995 (pp. 252-260).

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

Aggarwal, M., Arora, A., Sodhani, S., & Krishnamurthy, B. (2018, June). Improving Search Through A3C Reinforcement Learning Based Conversational Agent. In International Conference on Computational Science (pp. 273-286). Springer, Cham.

Sutton, R. S., & Barto, A. G. (1998). Introduction to reinforcement learning (Vol. 135). Cambridge: MIT press.

Sutton, R. S. (1992). Introduction: The challenge of reinforcement learning. In Reinforcement Learning (pp. 1-3). Springer, Boston, MA.

LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. nature, 521(7553), 436.

Schmidhuber, J. (2015). Deep learning in neural networks: An

overview. Neural networks, 61, 85-117.

Rosenblatt, F. (1958). The perceptron: a probabilistic model for information storage and organization in the brain. Psychological review, 65(6), 386.

Rowley, H. A., Baluja, S., & Kanade, T. (1998). Neural network-based face detection. IEEE Transactions on pattern analysis and machine intelligence, 20(1), 23-38.

Hagan, M. T., Demuth, H. B., Beale, M. H., & De Jesús, O. (1996). Neural network design (Vol. 20). Boston: Pws Pub..

Zamora, I., Lopez, N. G., Vilches, V. M., & Cordero, A. H. (2016). Extending the OpenAI Gym for robotics: a toolkit for reinforcement learning using ROS and Gazebo. arXiv preprint arXiv:1608.05742.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., ... & Petersen, S. (2015). Human-level control through deep reinforcement learning. Nature, 518(7540), 529.

Sutton, R. S., McAllester, D. A., Singh, S. P., & Mansour, Y. (2000). Policy gradient methods for reinforcement learning with function approximation. In Advances in neural information processing systems (pp. 1057-1063).

Littman, M. L. (1994). Markov games as a framework for multi-agent reinforcement learning. In Machine Learning Proceedings 1994 (pp. 157-163).

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., ... & Kavukcuoglu, K. (2016, June). Asynchronous methods for deep reinforcement learning. In International conference on machine learning (pp. 1928-1937).

August, M., & Hernández-Lobato, J. M. (2018). Taking gradients through experiments: LSTMs and memory proximal policy optimization for black-box quantum control. arXiv preprint arXiv:1802.04063.

August, M., & Hernández-Lobato, J. M. (2018). Taking gradients through experiments: LSTMs and memory proximal policy optimization for black-box quantum control. arXiv preprint arXiv:1802.04063.

Song, J., & Wu, Y. (2018). An Empirical Analysis of Proximal Policy Optimization with Kronecker-factored Natural Gradients. arXiv preprint arXiv:1801.05566.

Norman, M. D., Koehler, M. T., Kutarnia, J. F., Silvey, P. E., Tolk, A., & Tracy, B. A. (2018, July). Applying Complexity Science with Machine Learning, Agent-Based Models, and Game Engines: Towards Embodied Complex Systems Engineering. In International Conference on Complex Systems (pp. 173-183). Springer, Cham.