

第十九屆旺宏科學獎

成果報告書

參賽編號：SA19-120

作品名稱：智慧物聯暗棋機器人

姓名：林昕銳

關鍵字：AIoT、強化學習、翻棋機器手臂

摘要

智慧物聯 (AIoT) 在生活中逐漸普遍，應用在機器人、智慧家電、語音助理...等各種用途上。本研究將以 AIoT 概念開發暗棋機器人，可讓任何程式開發者，透過開放的 http 協議要求雲端應用程式介面(API)，開發各種形式的暗棋對弈程式或機器人。暗棋對弈系統基礎架構運作，是具有雲端連線的 IPCAM 及機器手臂的機器人，系統向 IPCAM 要求暗棋的盤面影像，再透過卷積神經網路 (CNN) 辨識暗棋盤面，將棋譜提供給對弈人工智慧運算下一步的棋步，提供給機器手臂做出翻棋、移動、吃子的動作。此一設計方便的大量收集棋譜資料庫，作為未來暗棋對弈深度強化學習的資料樣本，AI 棋力將可提高，搭配機器手臂作下棋動作，CNN 辨識盤面，以 AIoT 概念架構，可望讓對弈機器人大眾化，逐漸普遍於各個地方。

在先前創意說明書階段，將建構 API 提供客戶端（包括機器手臂）使用，先前建構方法採用了 Node-Red，而後改成 Python Flask 模組，可將訓練的模型放在雲端上進行棋局與決策分析，也可以透過抓取資料達到持續訓練的效果。

在盤面影像辨識方面，先前以兩層卷積池化-256-128-64-14 結構進行網路架構，Accuracy 可達到 90.5%~98.2%，loss 低達 8.9%~1.7%。而後本研究參考了著名的 VGGnet 結構並且改良至辨識暗棋的模型，最終 Accuracy 高達 99.9%（極趨近 100%），loss 則為 7.2336e-06，而實際測試過後辨識成功率更高達 98.9%。

對奕人工智慧方面先前驗證了決策演算法 (Min-Max)與學習演算法(Q-learning)的效能，棋力方面與正常棋力的軟體與玩家對奕，勝率只達 10%~20%。而後決策演算法則改用 Alpha-Beta 進而提高搜索速度和深度，勝率達到 50%~60%，有正常棋力之水準，而學習演算法則參考 github 開源碼中，由 Surag Nair 先生開發的 alpha-zero-general 程式，將暗棋 AI 概念程式加入，採用 DRL+MCTS 進行訓練，透過自我對下產生訓練樣本，自身判斷棋步，帶到模型進行訓練，最後再驗證兩個演算法的效能。

在機器手臂的部分，先前因力矩處在極限值，導致馬達的咬合不完整，因此對機器手臂的一軸二軸改為兩顆馬達的動力，穩定性提高許多。

為了讓使用者有更好的體驗，拿掉了利用影像判斷自動換手的功能，取而代之的是僅有一顆按鈕的遙控器，而遙控器功能則有換手與重新開始遊戲，而整個作品的運作流程皆以 AIoT 概念來建立，使用者只要連接網路再啟動開關，即可方便遊玩。

壹、研究動機

近年來高齡化社會比例日益增長，依內政部不動產資訊平台數據，在短短六年內，獨居老人居住宅數就增長了 16 萬以上，如圖 1 所示。隨著老年化趨勢漸漸明朗，老人照護的需求也更形重要。而陪伴及照顧老人的機器人構想也成為一個很重要的主題。

項目、年度 地點	65歲以上老年人口數（人）		僅65歲以上老年人口居住宅數		中低收入老人人口數（人）	
	2013年	2019年	2013年	2019年	2013年	2019年
全國	262萬3868	348萬2816	37萬1168宅	53萬2188宅	11萬8654	14萬4867
台北市	35萬2133	46萬4030	5萬5620宅	7萬8761宅	1萬737	1萬680
新北市	35萬8434	55萬595	6萬95宅	9萬9497宅	9608	1萬4241
桃園市	17萬5964	25萬8439	2萬2311宅	3萬7601宅	6299	7498
台中市	24萬5630	34萬6909	2萬9846宅	4萬5320宅	1萬846	1萬6675
台南市	22萬4986	28萬6947	3萬2874宅	4萬3112宅	8237	1萬1475
高雄市	30萬5808	42萬2444	4萬6441宅	6萬7584宅	3萬	3萬3914
六都小計	166萬2955	232萬9364	24萬7187宅	37萬1875宅	7萬5727	9萬4483
六都佔 全國比率	63.38%	66.88%	66.60%	69.88%	63.82%	65.22%

註：均為當年第一季數字 資料來源：內政部不動產資訊平台 製表：記者徐義平

圖 1 2013 至 2019 年老人獨居宅數數據

(資料來源：<https://ec.ltn.com.tw/article/paper/1308381>)

在過去學長們的研究中，實際訪問公園及家中老人近 50 位老人的需求後發現，有接近 8 成的老人有想跟別人下棋的想法，而根據受訪數據，暗棋的選擇偏好竟高達了 40%。而走訪了一些公園也常常看到老人家圍在一起，下著暗棋談天說地十分快活，讓我們體驗到一個老人在頤養天年的時候，有個陪伴玩樂的對象是多重要的事情。

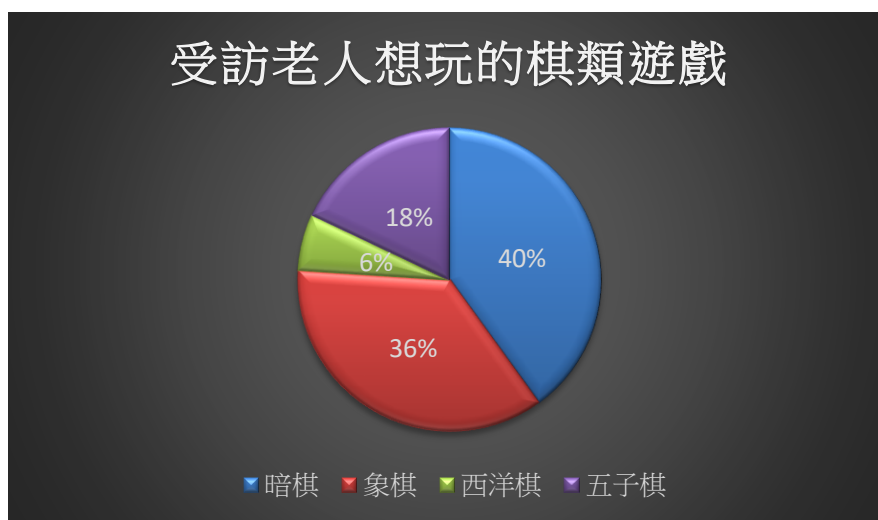


圖 2 受訪數據(本研究調查結果)

(資料來源：由公園、家人中的老人訪問數據，年齡分布於 55~75 歲)

在獨居老人的數量逐年增加的趨勢下，一個人少了下棋的夥伴。若此時有可以陪伴下棋的機器人，那將可以使更多獨居老人受惠。由先前的研究對弈機器人多著重於本機的系統，機器手臂、視訊及電腦間，存在著一個封閉的關係，無法對外聯繫。這讓我們想到如果我們的對弈系統，可以透過物聯網的技術，讓對弈機器人與廣大網路群眾連結，那是否能擴大生活圈。

搭配人工智慧形成的智慧物聯網 AIoT 概念開發暗棋機器人系統，將系統中的各個函式包裝成 API 藉由 HTTP 協定散播給各個開發者去開發對弈機器人，讓 Maker、廠商、家中對電腦熱情的孫子...都可以開發機器人。久而久之，對弈機器人將普遍於各個公園、家戶中，存在各型各樣的暗棋機器人。此外使用者所下的棋譜數據將會傳至 API 進行資料分析，當資料越龐大時，亦可作照護的應用，對於照護人員的資訊也獲得多了一種管道。

貳、研究目的

本研究方面，主要研究目的為以下敘述：

- 一、 以手機攝像頭當作機器視覺，將棋局畫面透過訓練之 CNN 模型辨識棋盤上的棋子。
- 二、 編寫暗棋對弈人工智慧，採用 Alpha-beta 剪枝法，預測棋局走向，找出最有利的棋步。
- 三、 以強化學習訓練暗棋對弈人工智慧，並試驗其效果。
- 四、 參考 github 開源碼中，由 Surag Nair 先生開發的 alpha-zero-general 程式，加入訓練暗棋 AI 架構之程式碼，嘗試建構出以 MCTS 搭配 DRL 所訓練出的電腦。
- 五、 使用高耐操之 AI 伺服馬達建構三軸機器手臂，並設計暗棋機器人專用的夾爪。
- 六、 整合機器視覺、對弈人工智慧及翻棋機器手臂來建構一組可以與人對弈的實體機器人。
- 七、 設計暗棋機器人雲端服務，將程式部分整合成 API，以 ESP8266 當作 API 與機器手臂間溝通的橋樑。
- 八、 蒐集使用者與機器手臂下的棋譜資料，讓對弈人工智慧能夠達到持續訓練的效果。
- 九、 將手機架設於平台上方，開啟作品開關並連上 WI-FI 後，以自製無線遙控器上方按下按鈕即可開始遊玩。

參、研究過程

一、智慧物聯網(AIoT)簡要說明

物聯網(IoT)主要分為三個階層：感知層、網路層、應用層，感知層接收各種狀態、資訊，透過無線感測區域網路(如：藍芽、RFID、Wi-Fi 等)傳送至網路層；網路層被當作感知層與應用層之間溝通的橋樑，透過 TCP/IP 網際網路、Wi-Fi 等將資料傳至應用層；應用層用於資料分析、雲端計算，對接收到的各種資料進行運算及分析，計算出各種結果。隨著資料量愈加龐大，一般的計算難以負荷，於是利用 AI 進行深度分析，可將計算的時間大幅減少，產生了智慧物聯網(AIoT)的概念。本研究以 AIoT 概念建構 API 應用至暗棋機器人系統，可望將蒐集來的資料搭配 AI 進行深度分析，圖 3 為物聯網於各層的簡要說明。

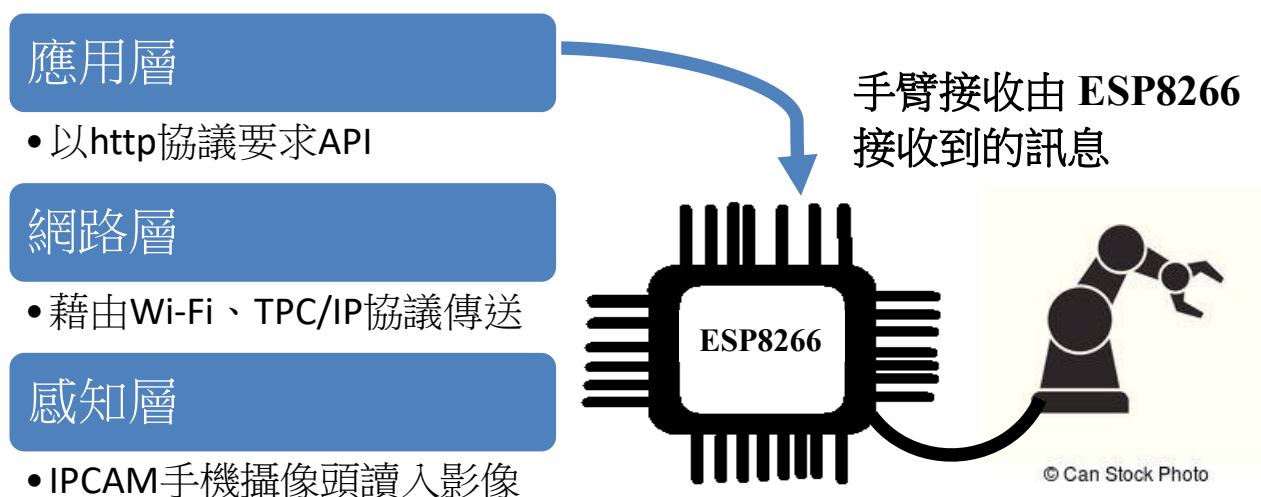


圖 3 物聯網各層簡要資訊

二、研究架構

本研究延續黑白棋對弈機器人，針對暗棋機器人進行開發，主要的差別將針對暗棋，設計象棋視覺辨識及具有翻棋動作的夾爪。對弈人工智慧方面除了傳統審局函數搭配演算法(如 Min-Max、Alpha-Beta)製作暗棋人工智慧外，目前仍未嘗試的暗棋搭配深度學習的人工智慧。最後將所需的 (1)象棋視覺辨識開發 (2)對弈人工智慧的功能，簡化建立成雲端服務 API 供各種客戶端連線，使廣大的開發者可以開發不同形式的對弈機器人。使用者對弈過程中，可不斷取得對弈棋譜，使雲端可蒐集大量的棋譜資料庫，最後得以為深度學習所用進行資料分析。以上各部的功能整合成智慧物聯網 (AIoT) 的架構。然而，於是本研究將嘗試以強化學習的深度強化學習 (RL) 製作，使 AIoT 的後端部分，可以透過多次對弈的及自我學習的棋譜，從無到有學會暗棋的下棋及贏棋。本研究之系統架構圖如圖 4。

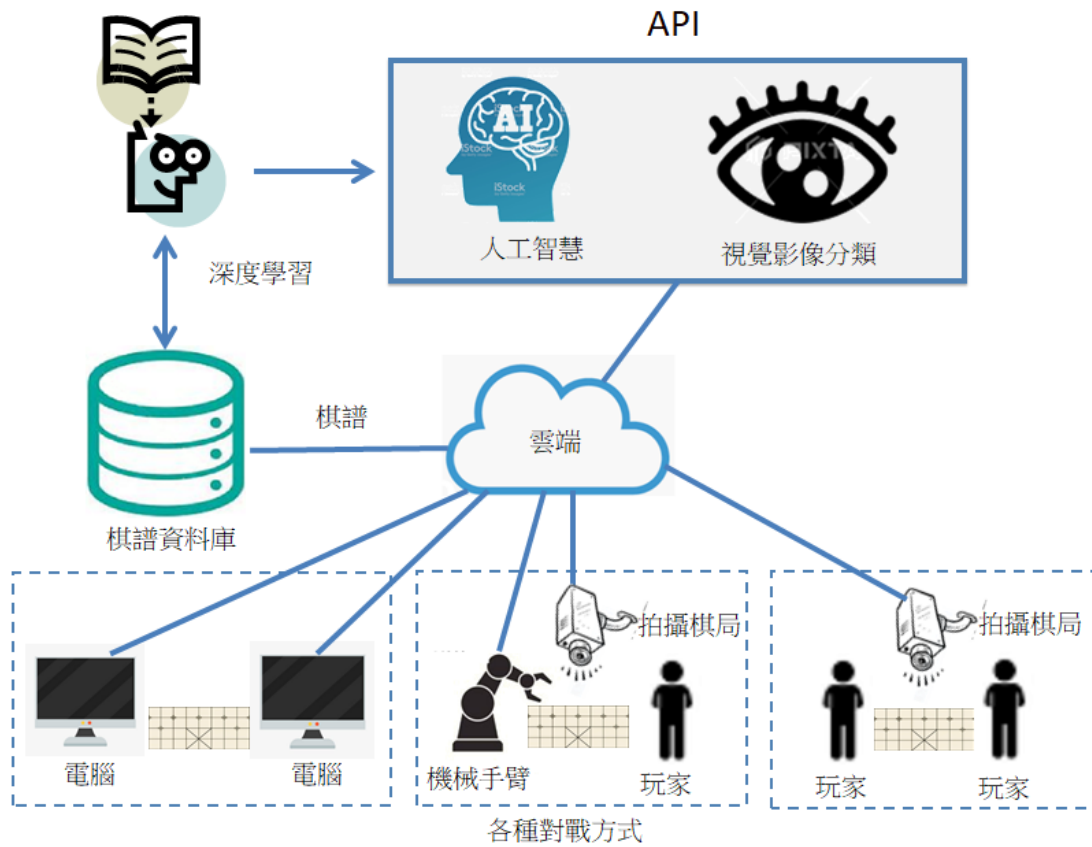


圖 4 系統架構圖

三、暗棋機器人系統架構

本研究主要基於「機器手臂 vs 玩家」情境作出暗棋機器人系統，此系統由 IP 攝像頭、電腦控制系統與機械手臂組成一個閉迴路系統，作用於暗棋棋局上。透過電腦控制系統分析影像所得數據傳送指令給手臂完成準確動作。圖 5 為暗棋機器人運作流程圖：

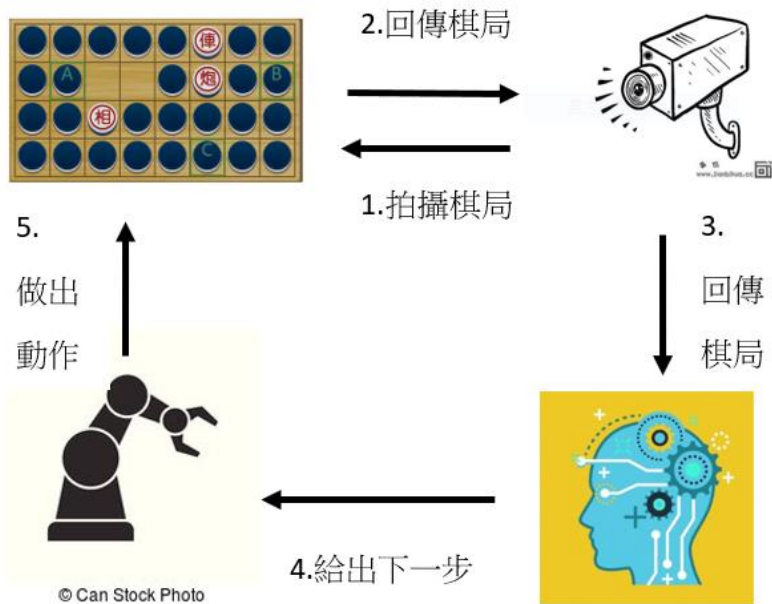


圖 5 暗棋機器人運作流程圖

四、暗棋對弈人工智慧

(一) 暗棋基本規則

1. 棋子放在 8*4 棋盤大小的格子，棋盤上放滿 32 顆亂數隨意排列的暗面象棋棋子。還沒翻的為暗棋，翻過的則為明棋。
2. 雙方輪流下棋，每次可選擇翻開暗棋、移動自己的明棋或吃對手的明棋。
3. 以先手方翻開的棋子決定黑方紅方，如：電腦 A 先手並翻開紅色棋子，則電腦 A 為紅方，電腦 B 則為黑方。
4. 每種明棋只能一次只能前後左右四個方向移動一格。
5. 每顆明棋大小位階為如：紅方為「帥>仕>相>車>馬>炮>兵」，黑方為「將>士>象>車>馬>包>卒」，依照這個位階大的可以吃小的，小的不能吃大的(特殊規則：帥、將不能吃卒、兵，卒、兵可以吃帥、將；炮、包不能吃前後左右的明棋，但可以跳過前後左右任何一個明棋，去吃任何敵方的明棋)。
6. 遊戲的勝負：將對方的所有棋子都吃掉可以獲得勝利，如果雙方都無法將對方的棋子完全吃掉，為了避免無限走棋，當雙方走步合計達 25 步或 50 步均無翻棋或吃棋時就算和局。

(二) 棋子代碼及陣列規劃

首先，程式必須定義象棋所有紅黑棋子的代碼如表 3，並且將棋局的棋子放置在一個 32 (8*4)大小的一維陣列中。

表 3 棋子代號對應圖

K	G	M	R	N	C	P	*
帥	仕	相	俥	傴	炮	兵	暗
k	g	M	r	n	c	p	0
將	士	象	車	馬	包	卒	空

(三) 合法步產生函數

程式必須要根據暗棋的規則下出合理棋步，這被稱為「合法步」，程式必須以函數產生出所有的合法步，從裡頭判斷出最有利棋步。以下為合法步的產生方式：

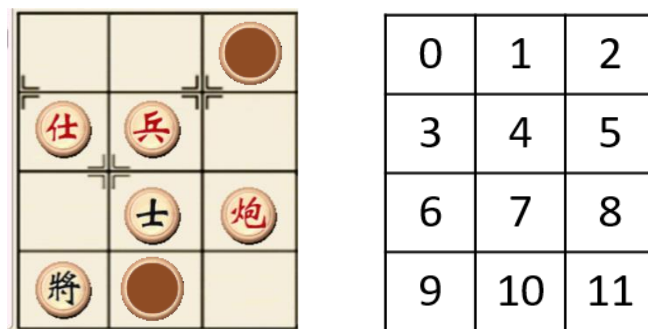


圖 6 合法步舉例

定義每個可用棋子所在地及棋子落子地，格式如下：

(棋子所在地, 棋子落子地)

以上圖黑方為例子，可用的棋子有兩個暗子及黑士與黑將，棋子位置為：2,7,9,10，可走的合法步為 2,4,6,8,10，由於 2 是暗子，可以走的合法步為 2，所以該位置的棋子與可走合法步表示為 (2,2)；以 7 為例，可走的合法步為 4,6,8，可走的合法步為 (7,4), (7,6) 及(7,8)。以此類推，則這個盤面所的可用的位置及合法步共有 6 個，此一及集合 C 為：

$$C = \{ (2,2), (7,4), (7,6), (7,8), (9,6), (10,10) \}$$

(四) 遊戲樹簡要說明

遊戲樹指的是當遊戲進行到某回合，將後續遊戲變化的所有可能性，利用樹狀結構擴展出數個節點。遊戲樹做為人工智慧的基礎，搭配像是 Min-Max、Alpha-beta 剪枝、蒙地卡羅樹搜尋...等演算法，來做為審局、提高運算效率及嘗試更多獲勝的可能性。圖 7 為暗棋棋局中一節點的遊戲樹。

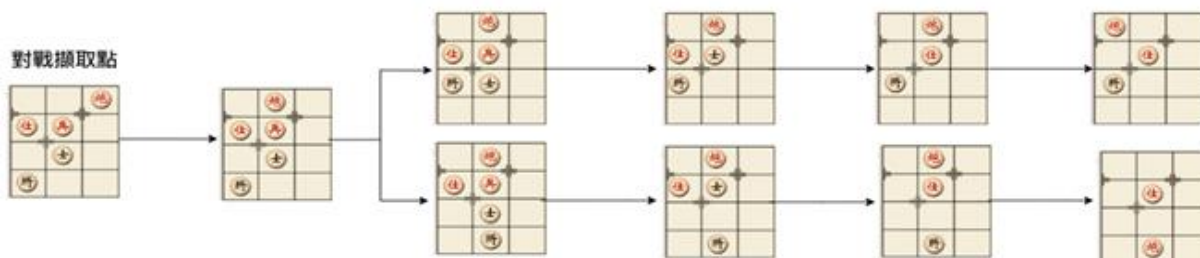


圖 7 暗棋遊戲樹舉例

(五) 暗棋審局函數

審局函數指的是對某一節點或者的權重比，也可以設定每個棋子的權重，通常審局函數都跟一些演算法進行搭配。表 4 為玩家對每個棋子的權重，若雙方顏色互換，則

分數進行反轉。

表 4 棋子權重比 (黑方為進攻，紅方防守)

帥	仕	相	俎	馮	炮	兵	移動
100	90	50	30	10	70	5	-1
將	士	象	車	馬	包	卒	移動
-100	-90	-50	-30	-10	-70	-5	1

(六) Alpha-Beta 剪枝搜尋演算法

在過去的研究中，本研究嘗試了使用 MIN-MAX 演算法進行對弈，因搜索深度只有到兩層，實際與人對戰過後勝率不高，於是為了提高勝率，本研究將搜尋深度增加至 6 層，但因 6 層運行速度較慢，於是使用了 Alpha-Beta 演算法提高速度，與 MIN-MAX 一樣，都是需要計算出每一葉節點的分數，再將其反向傳播，以下為葉節點計算說明：



設 6 層後的走法：
 (8→7)，(4→1)，(0→1)，(2→1)，(6→3)，(1→2)
 則此走法葉節點分數為：
 $(-1)+1+5+(-90)+(-1)+(+1) = -85$

圖 8 為 Alpha-Beta 剪枝法的舉例，路徑與葉節點為 6 個，剪枝後減少了時間：

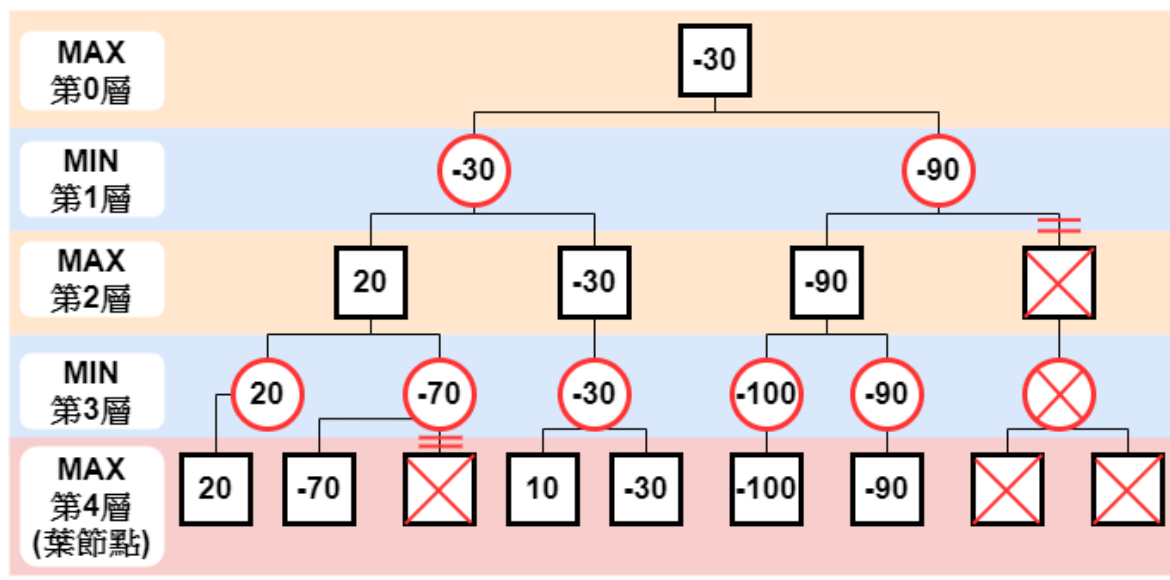


圖 8 Alpha-Beta 剪枝法舉例

由以上說明，剪枝的第一個走法因第 3 層怎麼選都無法大於 20，於是將不搜索此走法；同理剪枝的第二個走法因無法大於 -30，這樣的演算法與最有利走法與不利走法的排序有關，從此走法開始搜索，可有效減少運行的時間。

(七) 定義強化學習(Reinforcement Learning,RL)各個資訊

暗棋屬於「不完全資訊」之棋類，不像圍棋、象棋能掌握盤面資訊，每一步都可以運算出準確的審局函數結果，本研究提出利用強化學習，學會如何下暗棋，嘗試能夠戰勝正常棋力人類。訓練的做法讓電腦「依照合法步規則亂下棋」進行左右互博，經過上萬次的對弈後，棋力水準將提高至正常水平。以下為暗棋在強化學習中的說明：

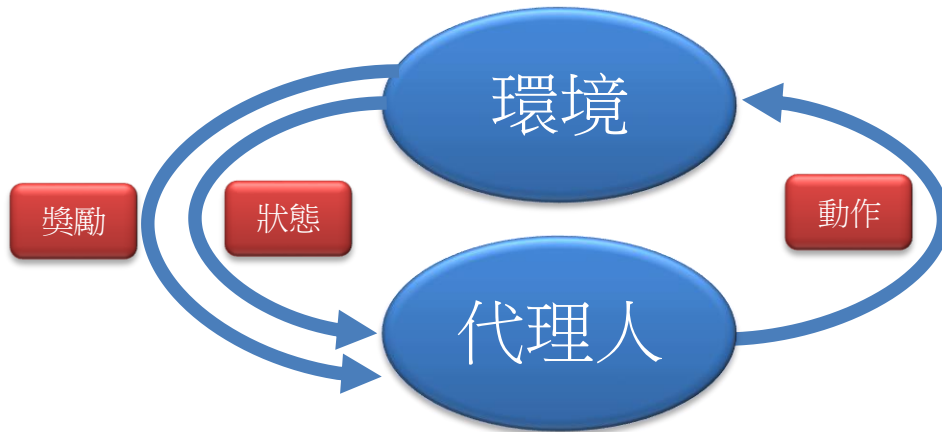


圖 9 強化學習流程圖

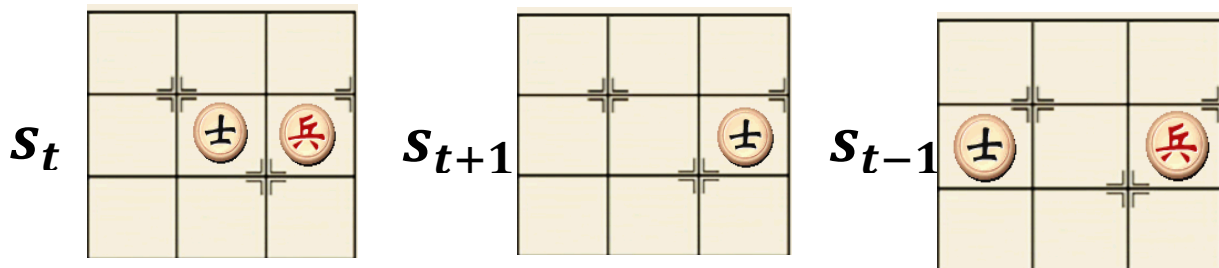
1. 動作：定義從動作集 A 選擇一個動作 a_t ，下一個動作為 a_{t+1} ，包含翻棋、移動、吃子動作，以座標存取，型態為大小 2 的一維陣列，在合法步函數中有說明。
2. 狀態：定義從狀態集 S 對應目前狀態 s_t ，下一個會發生的狀態為 s_{t+1} ，起始狀態為 s_0 ，包含棋局盤面的狀態(棋局盤面 32 大小一維陣列)。
3. 獎勵：定義為 r ，分數規則為自訂，這裡自訂為「贏一局得+1 分，輸一局得-1 分，平手得 0 分」。
4. 代理人：電腦本人(AI)，負責接收狀態 s_t 和獎勵 r ，並做出動作 a_t 。
5. 環境：有一套規則的空間，裡面由代理人依照狀態和獎勵作出動作。這裡規則如下「由代理人進行回合制下棋，棋步依照規則來下，分出勝負即重新一局」。

(八) 強化學習(RL)簡要說明

強化學習常利用 Q-Learning 或 DL 進行訓練，QL 方式為得到的 Q 值存入 Q-table 裡，當訓練得越多次，實力也會增加，缺點是當狀態集或動作及暴增，就無法運作了，於是 DL 會搭配強化學習改善問題，也就是深度強化學習(DRL)，利用深度神經網路(DNN)作出類似於 Q 表，而更新公式為著名的 Bellman 公式，如下：

$$Q(s, a) = r + \gamma * \underbrace{\max Q(s_{t+1}, a_{t+1})}_{\text{根據 Q 表選擇最大 Q 值}}$$

在公式中， γ 為折扣因數，定義為 $\gamma \in [0,1]$ ，將得到的 r 逐漸下降並產生出各個 Q 值， Q 值的產生簡要範例如下($\gamma=0.9$)：



第 N 局對弈：當 s_t 變換至 s_{t+1} 時

$\text{random}(A) = a_t = (4, 5)$

$r = +100$

$Q(s_t, (4,5)) = 100 + 0.9 * \max(0, 0, 0, 0) = 100$

Q 值更新如下表：

A \ S	s_{t-1}	s_t	s_{t+1}
(0,0)	0	0	0
省略...	0	0	0
(4,5)	0	100	0
(3,4)	0	0	0

第 $N+1$ 局對弈：當 s_{t-1} 變換至 s_t 時

$\text{random}(A) = a_t = (3, 4)$

$Q(s_{t-1}, (3,4)) = 100 + 0.9 * \max(0, 0, 100, 0) = 90$

Q 值更新如下表：

A \ S	s_{t-1}	s_t	s_{t+1}
(0,0)	0	0	0
省略...	0	0	0
(4,5)	0	100	0
(3,4)	90	0	0

以上方所述，當棋局到了 s_{t+1} 時，便會根據公式產生 Q 值，則第 N 局時回到 s_t 時，將會學到作出 a_t 動作時，可以得到 $\text{reward}(+100)$ ，並且將 $Q(s_{t-1}, a_{t-1}) * \gamma$ 計為 90 分，把強化學習套用在暗棋對弈上，重複訓練多次之後，可將 Q -table 產生出來，電腦亦會越下越強，但是暗棋遊戲中的狀態集實在太多，於是將嘗試利用 DRL+MCTS 做出改善。

(九) 蒙地卡羅樹搜尋(Monte Carlo Tree Search, MCTS)與 UCT 演算法

蒙地卡羅是以大量的模擬資料來取得一個近似的解，其中著名的例子是用亂數取得圓周率的值，而對暗棋狀態多又不完全資訊的遊戲來說，利用 MCTS 搭配 DNN 去做訓練或許是好方法。利用 MCTS 與 UCT 產生自我對下的棋譜，可讓神經網路不過於離散的狀況，在 MCTS 裡，會搭配 UCB 公式達到更好的效果，讓棋譜更凝聚，說明如下：

UCB 公式：

$$\frac{w_i}{N_i} + \sqrt{\frac{2 \ln N}{N_i}}$$

- w_i 為此子節點獲勝次數
- N_i 為此子節點模擬次數
- $\sqrt{2 \ln}$ 為常數，可調整(本研究使用此)
- N 為此節點總模擬次數

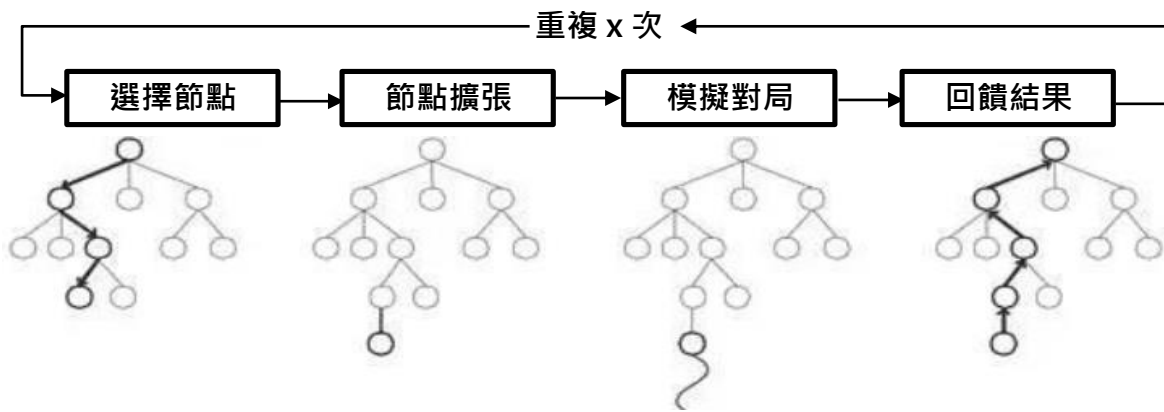
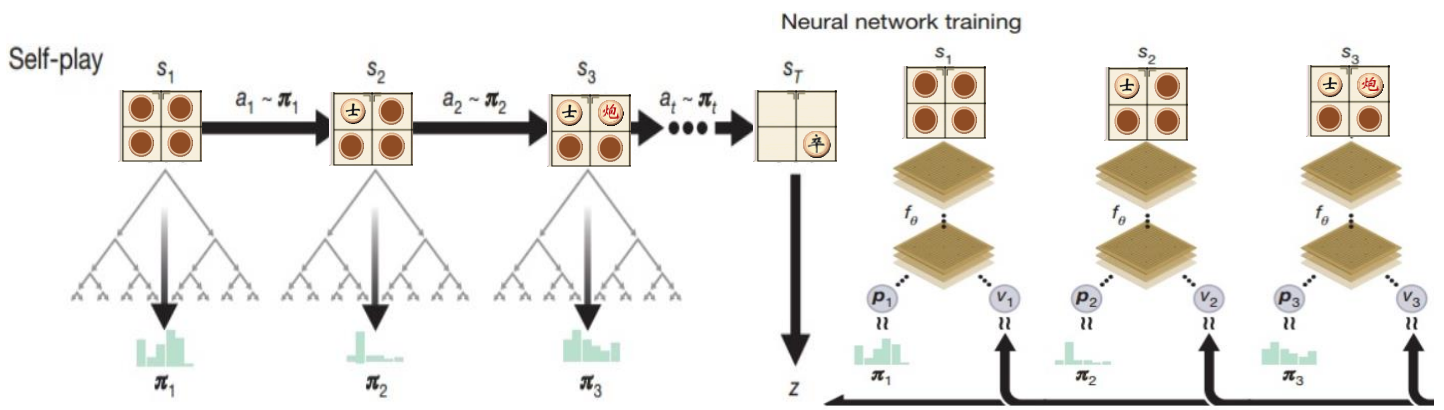


圖 10 蒙地卡羅樹搜尋流程圖

(十) AlphaZero

AlphaZero 為 AlphaGo、AlphaGoZero 後的版本，其最大特點為利用自身左右互博對下，多次訓練後，達到學習的效果。其架構大致分為兩個階段，第一階段：電腦進行自我對下產生棋譜訓練樣本，可透過 MCTS 產生凝聚的樣本，第二階段：再將訓練樣本代入模型進行訓練，調整模型的權重，反覆迭代訓練變得更強。以下流程說明：



圖(a)以 MCTS 自我對下產生訓練樣本

圖(b)訓練樣本帶入網路進行訓練

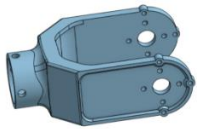
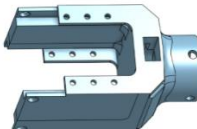
圖 11 AlphaZero 訓練架構圖




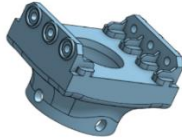
五、機器手臂設計

(一) 關節設計

本次研究，手臂關節打算利用 AI 馬達做出具有三軸關節之訪人機器手臂，經過了 2 個月的耗時，最終成功組合出有效的三軸關節手臂，表 5 為零件資訊：

表 5 零件資訊

照片	數量	用途	照片	數量	用途
	2 個	連接馬達轉盤		1 個	連接手臂與管子

照片	數量	用途	照片	數量	用途
	1 個	控制手臂水平底座		2 個	鋁製管子，手臂的 A、B 臂
	1 個	水平底座旋轉連接座		1 個	連接手臂與管子

以上零件由 Onshape 網站畫出，再組合出一個模擬圖，圖 12 為三軸的模擬圖。

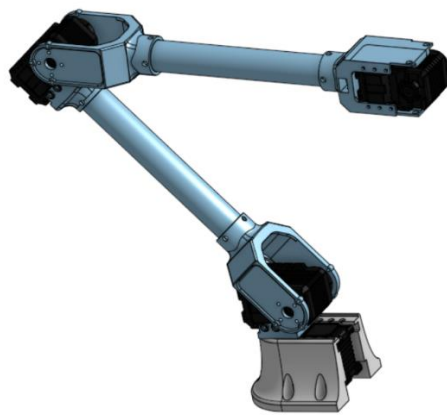
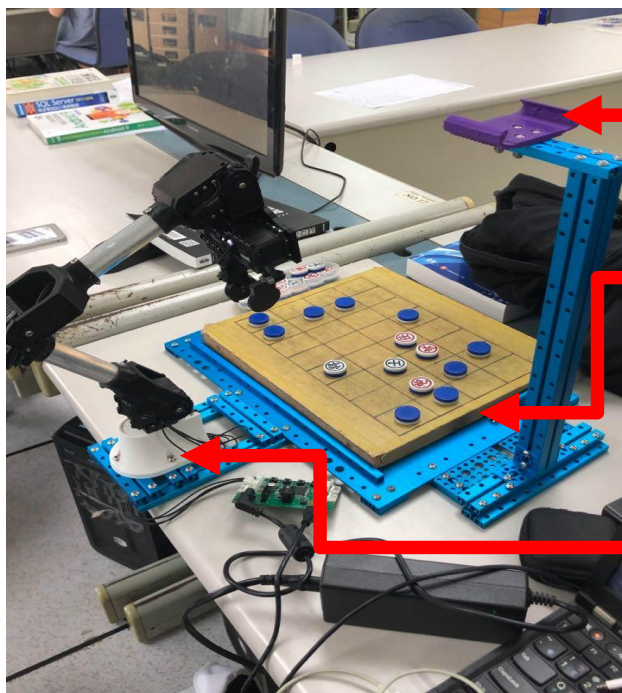


圖 12 三軸關節手臂模擬圖

(二) 平台設計

平台打算用 makeblock 機械零件來設計，以下為平台說明：



手機座：作為手機 IPCAM，用於拍攝棋局。

平台面：放棋盤的平面，有固定輔助位置以方便擺放。

手臂平台：擺放手臂的平台，可控制與棋盤的距離。

圖 13 平台設計說明

(三) XYZ 立體作標設計

由於機器手臂必須有自動下棋的功能，因此本研究以餘弦定理和極作標，將馬達旋轉角度轉換為 XYZ 立體作標，以下為推算方法：

1. Z 軸旋轉角(θ)與手臂伸長距離(L)推導：

為了判斷 Z 軸旋轉角(第 1 軸馬達)縮需旋轉的角度，需要用到極座標的概念，圖 14 為極座標示意圖，於上方俯視棋盤，可以觀察出棋盤與手臂為垂直 90° ，分別為水平 X 軸與垂直 Y 軸，藉由輸入的座標，可以推出手臂伸長的距離 L 與馬達旋轉的角度 θ ，以下為公式推導：

手臂伸長距離 L：

$$L = \sqrt{Lx^2 + Ly^2}$$

馬達旋轉角度 θ ：

$$\theta = \tan^{-1} \frac{Ly}{Lx}$$

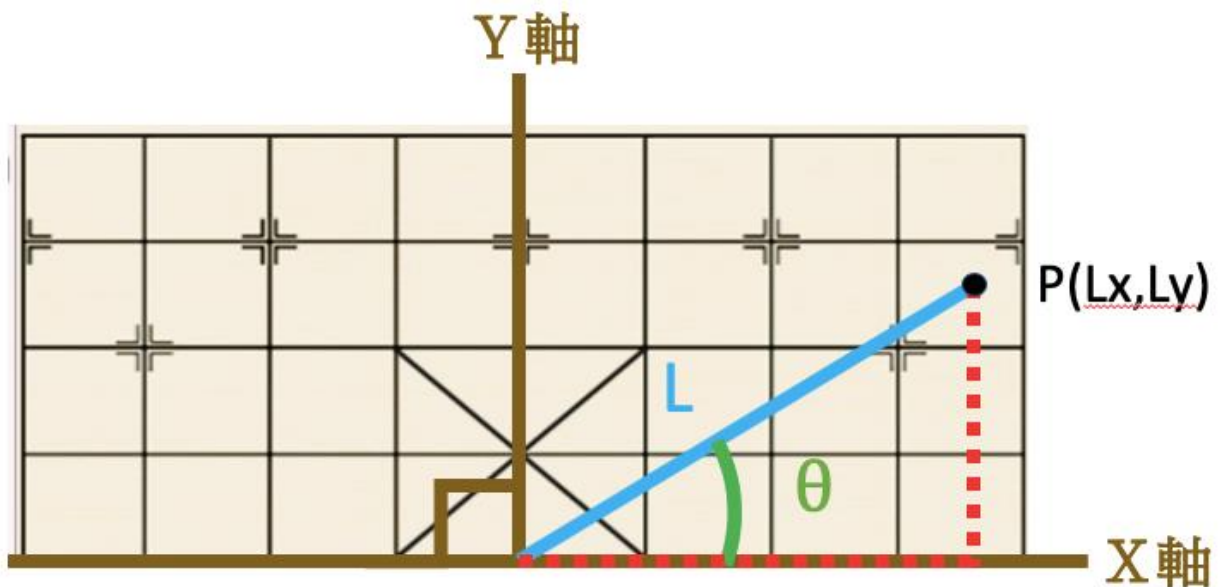


圖 14 極座標於棋盤上示意圖

2. 各軸馬達旋轉角度推導：

手臂依靠馬達旋轉運動，讓手臂作出垂直抓取動作(即 $d \perp L$)，為了求出第 2、3、4 軸馬達所需旋轉的角度 θ_1 、 θ_2 、 θ_3 需要用到餘弦定理，藉由上述說明得出 L，推出 θ_1 、 θ_2 、 θ_3 ，以下為公式推導：

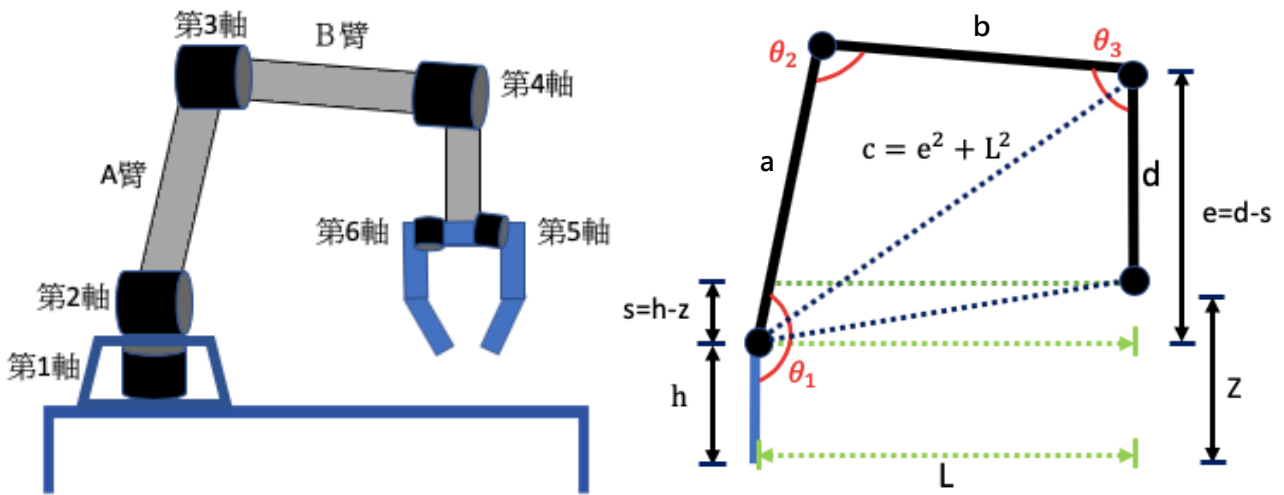


圖 15 機器手臂與餘弦定理示意圖

$$\theta_3 = \cos^{-1} \frac{c^2 + e^2 - l^2}{2ce} + \cos^{-1} \frac{b^2 + c^2 - a^2}{2bc}$$

$$\theta_2 = \cos^{-1} \frac{a^2 + b^2 - c^2}{2ab}$$




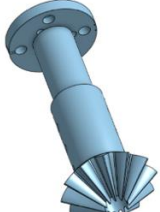
$$\theta_1 = 360 - \theta_2 - \theta_3$$


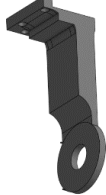

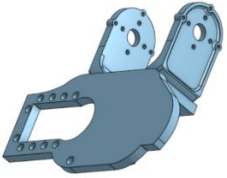
有了以上公式後，只需棋盤上的 4 個角落的座標，就可以將其餘的格子座標計算出來。

(四) 夾爪設計：

在暗棋遊戲中，除了有夾取的動作，最難克服的地方為翻棋動作，耗時了四個月的研究並經歷了兩代的失敗後，最終第三代夾爪利用了比較特別的設計，夾爪零件資訊為表 6。

表 6 夾爪零件資訊

照片				
用途	作為被動式圓盤	構成主動式夾爪的連接零件	作為主動式夾爪面，用於夾取動作	用來轉動斜切齒輪圓盤的零件，由法蘭面、細鋁棒、斜切齒輪所構成

照片				
用途	作為被動式斜切齒輪圓盤	作為被動式夾爪面	構成主動式夾爪的主要零件	整個夾爪的包覆零件，也用於第四軸馬達轉動

將上述零件印完組裝後，做出具有翻棋與抓取動作的夾爪，夾爪模擬圖為圖 16。

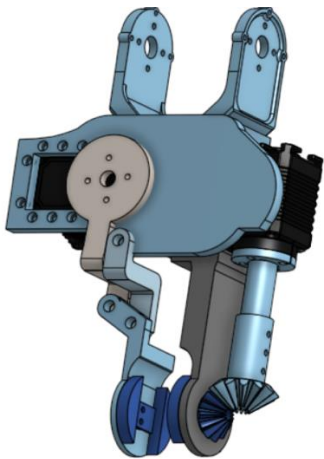


圖 16 第三代夾爪模擬圖

左方的馬達採用了四邊形結構來帶動左邊的夾爪面，當四邊形角度夠大，越不輕易撞到其它棋子，可從上方往下夾起，而右邊夾爪面則採取被動式。右方的馬達用來作翻棋動作，它用了一個較為特殊的齒輪為斜切齒輪，此齒輪可與其它齒輪呈現 90°帶動，再搭配於類似培林的零件，進而作出翻棋的動作，圖 17 為翻轉解析順序說明。

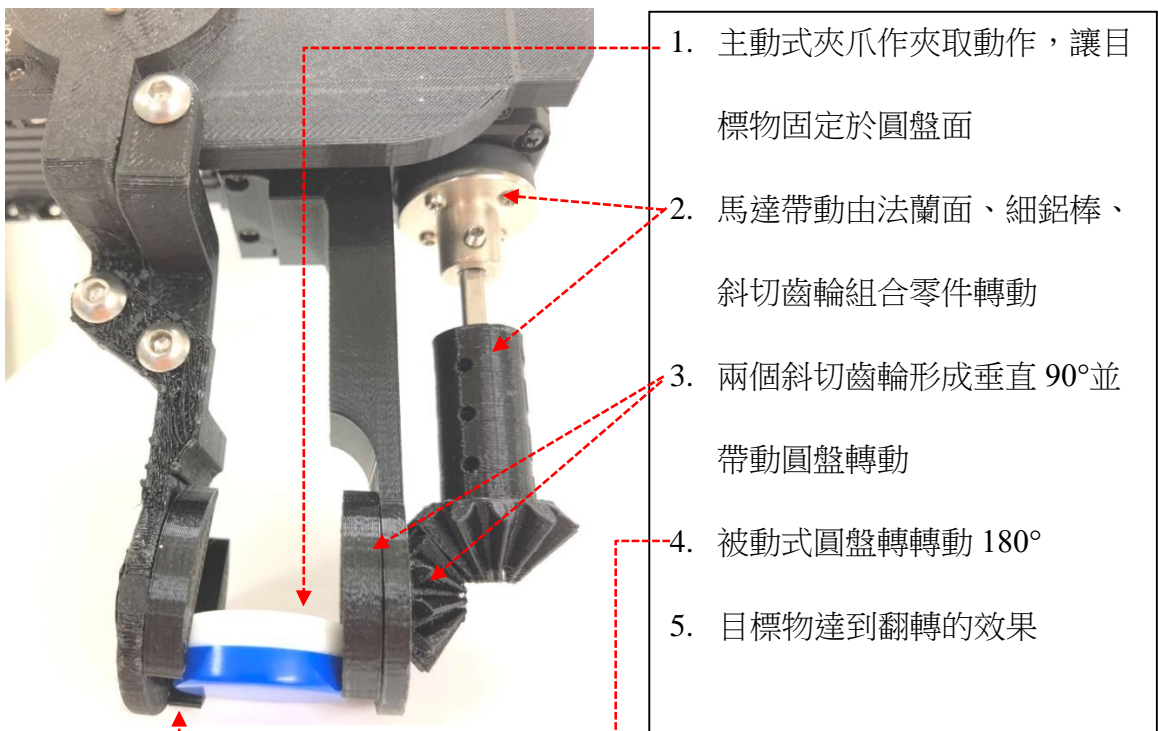


圖 17 可翻轉夾爪實體機構解析說明

(五) 馬達負載測試：

AI 馬達扭力為 25 kgw-cm，此手臂總重量為 750 gw，扣除掉底座 300 gw，馬達將抬起規格為 0.45 kgw, 臂長 54 cm 的手臂，帶入力矩公式為：

$$L = F * d$$

而輸入值帶入公式後為：

$$0.45 * 54 = 24.3 \text{ kgw-cm}$$

經計算後，24.3 kgw-cm 的力矩小於額定 25 kgw-cm AI 馬達是可以抬起機器手臂。

(六) 程式設計：

本研究以智慧物聯網概念架構系統，手臂只需透過網路連線即可運行，而前一代系統手臂連線方式只能於本機上遊玩，程式也必須存在於本機上，於是第二代系統包裝成 API，只需向其要求並收到手臂動作，不用安裝程式即可遊玩，表 7 為手臂系統差異表。

表 7 手臂系統差異表

差異 \ 版本	本機遊玩版本	物聯網遊玩板本
CPU 規格	ATmege1280	ESP8266
通訊方式	序列阜通訊，USB 線連接	要求 http 協議回傳
遊玩方式	利用 C#介面開啟遊戲，程式指存在於本機內	打上 URL 並連至 API 可遊玩，不需安裝程式
程式撰寫	程式全部包裝在一起，且需要安裝環境才能進行遊玩	全部放在雲端上供使用者運用，只需透過網路連線即可運用

手臂以 AI 馬達做出動作，而 AI 馬達的角度範圍為 0°~330°，在 ArduinoIDE 裡有專用的程式庫，其中有讓馬達轉動的基本函式，如下：

```
#include <XYZrobotServo.h> //將函式庫加進程式內
SoftwareSerial Serial1(10, 11); //與 AI 馬達聯繫之序列阜
XYZrobotServo xxx(Serial1, ID) //宣告 xxx 為某 ID 的馬達，與此 ID 馬達通訊
xxx.setPosition(DEGREE, PLAYTIME) //對 xxx 設定 DEGREE 角度//運行時間為
```

以上方所述補充，DEGREE 參數範圍為 0~1023，而實際轉動角度轉換如下：

$$1024 / 330 = 3.103030303 = \text{實際轉動 1 度的數值}$$

圖 18 為公式轉程式碼說明：

```

float theta[4];
theta[0] = atan2( y , x ) * 180 / PI; //極座標公式

//餘聞定理公式-----
theta[3] = (acos( (c*c + e*e - 1*1) / (2*c*e) ) * 180 / PI) +
           (acos( (b*b + c*c - a*a) / (2*b*c) ) * 180 / PI);

theta[2] = acos( (a*a + b*b - c*c) / (2*a*b) ) * 180 / PI;
theta[1] = 360-theta[3] - theta[2];
//-----
target[0] = Degree2Steps(0,-theta[0]); //角度轉換成馬達參數值
target[1] = Degree2Steps(1,theta[1]); //角度轉換成馬達參數值
target[2] = Degree2Steps(2,theta[2]); //角度轉換成馬達參數值
target[3] = Degree2Steps(3,theta[3]); //角度轉換成馬達參數值

```

圖 18 公式轉程式碼

六、暗棋棋局視覺辨識

(一) 影像處理

透過 IPCAM 擷取影像後，為了提升辨識的效能，必須作影像處理的動作，以便後續進行蒐集樣本及運行辨識系統的作用。

1. 灰階化：

在進行影像處理的過程中，將圖片轉為灰階必要動作，此功能將原本 RGB 圖片之三維陣列轉換為只有灰暗圖片之一維陣列，數值範圍為 0~255，圖片數值更小小更好作分析及運算，圖 19 為灰階示意圖。

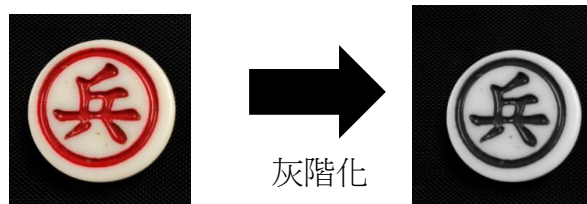


圖 19 灰階示意圖

2. 霍夫找圓：

擷取每個格子圖後，欲判斷該位置是否有棋子，利用了霍夫找圓方法偵測圓的存在，此函式可限制半徑大小以及控制閾值，回傳圖中所有符合條件的圓的資訊，格式為三維陣列：[[[a,b,r]]]，(a,b)為圓心座標，r 為半徑，利用此數據可對圖片進行標記及裁切，如圖 20。

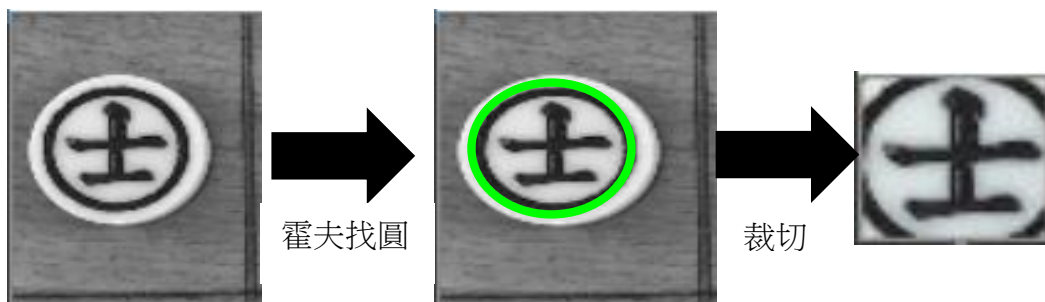


圖 20 進行霍夫找圓並裁切

3. 四角定位：

在暗棋對局中，棋子必須放在棋局格子上進行操作，為了計算出棋子的位置，需要找到長方形中四角的座標，再將座標帶入公式運算，最終得出為 32 個格子數據，圖 21 為示意圖：

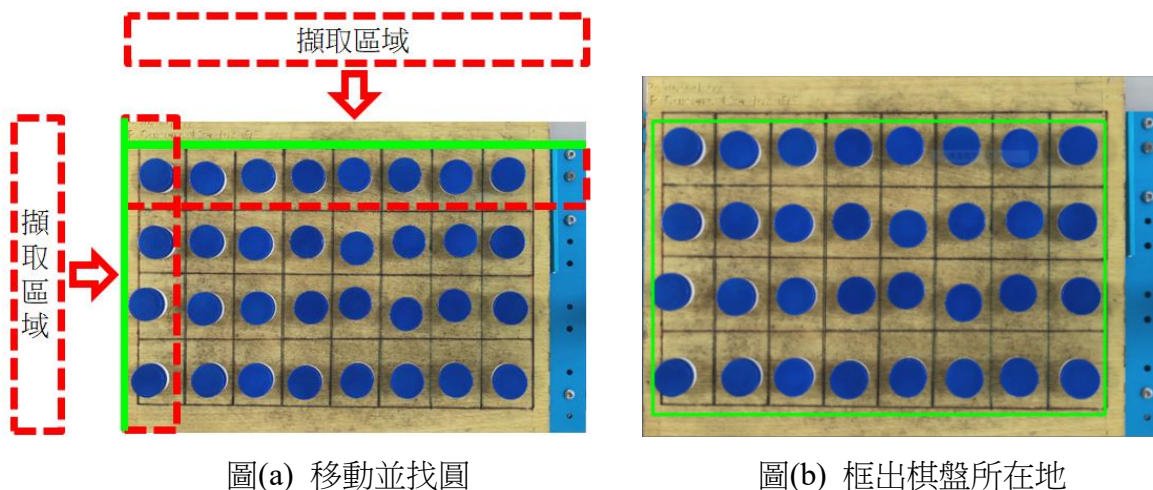


圖 21 四角定位示意圖

將擷取區域依序從上下左右往內移動，當偵測到任何的圓時，則取該區域最外圍邊線，進而框住並裁出棋盤所在位置，進而得知四角座標帶入 python 函式裡，如圖 22。

<p>座標資訊(x,y)：</p> <p>左上角 = LU</p> <p>右上角 = RU</p> <p>左下角 = LD</p> <p>右下角 = RD</p>	<pre> angle = np.array([LU, RU, LD, RD]) d1 = abs(angle[0]- angle[2]) / 4 d2 = abs(angle[1]- angle[3]) / 4 for r in range(4): s1 = angle[0]+d1*r s2 = angle[1]+d2*r dx = (s2-s1) / 8 for c in range(8): s = s1+dx*c p.append(tuple(s.astype('int'))) </pre>
---	---

圖 22 格子座標函式程式碼

取出座標後，將可以標記並裁出所有格子的位置，圖 23 為標記所有格子示意圖。

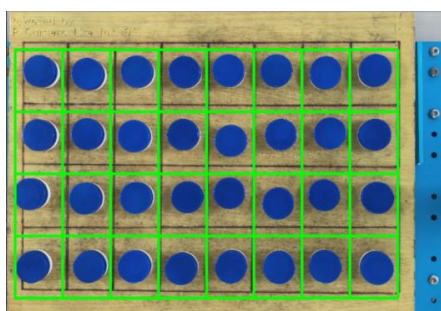


圖 23 標記格子示意圖

(二) 蒐集訓練樣本

為建構卷積神經網路模型，蒐集了所有角度變化的 14 種明棋各 50 張，並且丟入模型進行訓練，產生棋子的數量為：

$$360 \times 100 \times 14 = 252000 \text{ 張}$$

表 8 為取得訓練樣本的處理方法，順序為(a)~(d)。

表 8 取得訓練樣本的處理方法

	
(a) IPCAM 取得棋局畫面的串流	(b) 利用四角定位擷取棋盤位置
	
(c) 計算並擷取棋子的位置	(d) 縮成 28*28 並進行霍夫找圓裁切取得辨識樣本，每旋轉一度後存取樣本

(三) 類神經網路(NN)與卷積神經網路(CNN)簡介

CNN 為加強版 NN，與 NN 的差別在於，NN 是單純以圖片的數值去做訓練，而 CNN 會抓圖片中的特徵點，作出局部的感知判斷，提高辨識的效果。其運作方式為將圖片丟進卷積層、池化層作處理，最後形成全連接層，代替類神經網路的輸入層。

(四) CNN 模型建構

本研究採用 CNN 進行訓練，模型部份參考了著名 LeNet-5 模型與 AlexNet 模型進行結合改良版，前者為 CNN 之父，由最簡單的架構(2 層卷積層、2 層池化層、1 層全連接層)所組成；後者則是 2012 年 ImageNet LSVRC 比賽的冠軍，首度加入了 Relu 激活函數和 Dropout 防止 Overfitting，指的是神經元互相敏感程度太多，類似死腦筋的概念，而導致辨識率降低的情況，以下為模型的建構說明：

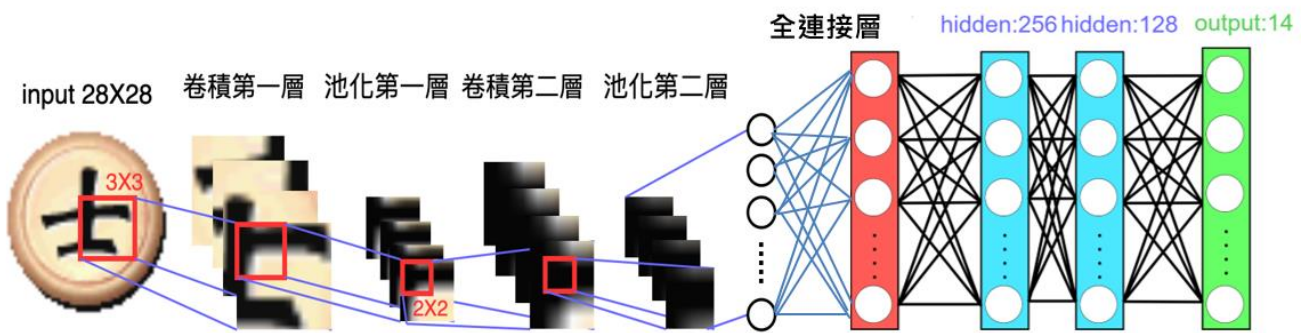


圖 24 CNN 模型圖

```

model = Sequential()
model.add(Conv2D(kernel_size=(5, 5), filters=128, strides=(1,1), activation='relu', input_shape=(28,28,1))) #宣告一個模型 #加入卷積層
model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2))) #加入池化層
model.add(Conv2D(kernel_size=(5, 5), filters=128, strides=(1,1), activation='relu')) #加入卷積層
model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2))) #加入池化層
model.add(Flatten()) #將圖片坦平，由二維轉為一維
model.add(Dense(256, activation = 'relu')) #加入隱藏層
model.add(Dense(128, activation = 'relu')) #加入隱藏層
model.add(Dropout(0.5)) #將神經元的0.5的份量(一半)給捨棄掉
model.add(Dense(14, activation='softmax')) #輸出14個結果(softmax)

```

圖 25 使用 Keras 進行 CNN 模型建模程式設定

(五) 辨識系統

完成訓練模型階段後，必須要有辨識系統得以處理 IPCAM 所傳入的資料，並做到回傳棋局的函數，圖 26 為系統流程圖：

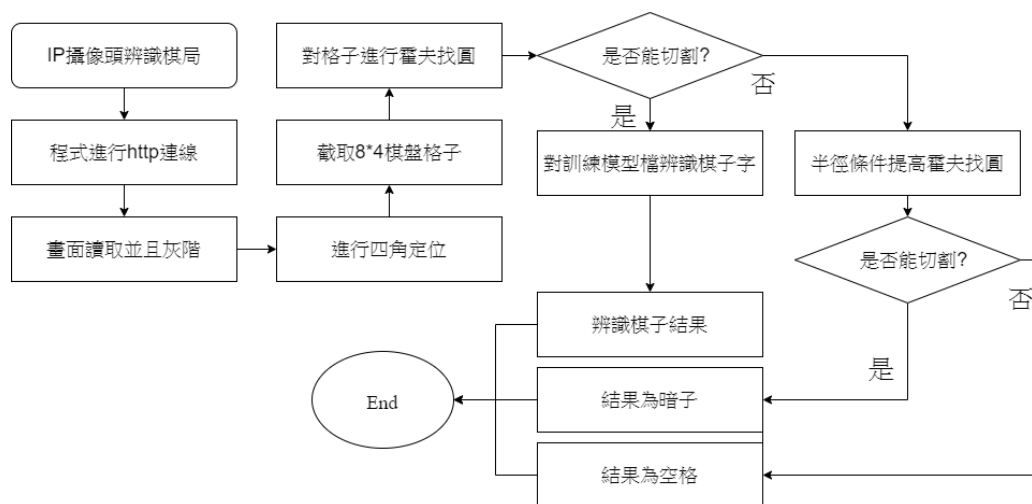


圖 26 辨識系統流程圖

七、智慧物聯應用程式介面(API)

為了讓暗棋的人工智慧運算，可以提供大眾方便取得，採用與開放資料平台提供的 API 格式對外開放。客戶端透過要求(Request)，雲端經過運算後回應(Response) JSON 的資料格

式。以下為 API 的通用格式：

http://{Server:port}/DarkchessAI/Fnunction?data = xxxxx...xxxx

- Server:port：電腦與 API 連接之編號與連接埠。
- DarkchessAI：本研究取之名字，所有客戶端透過此名字選取函數。
- Function：客戶端所可選的函數。
- ?data = xxxxx...xxxx：客戶端輸入給函數之值。

(一) 暗棋對弈 API

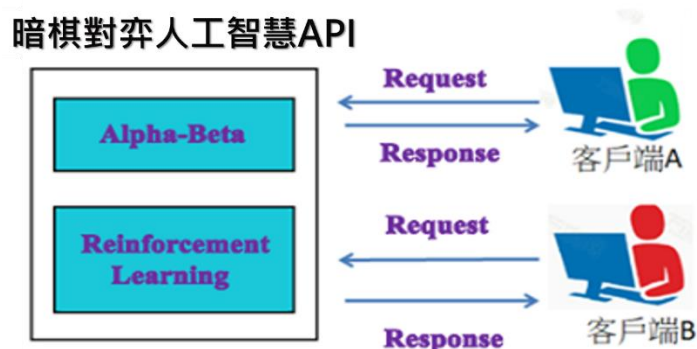


圖 27 暗棋對弈函數 API 說明

在對弈 AI 的 API 裡有 Alpha-Beta 函數及 Reinforcement Learning 函數可供客戶端選擇，客戶端的 Request 之 http 協議 data 為棋局資料，將原本的一維陣列轉為字串後傳入，而回傳為該棋局最有利的棋步，JSON 格式為 {Position:xx, Moveto:xx}。

(二) 棋局辨識函數

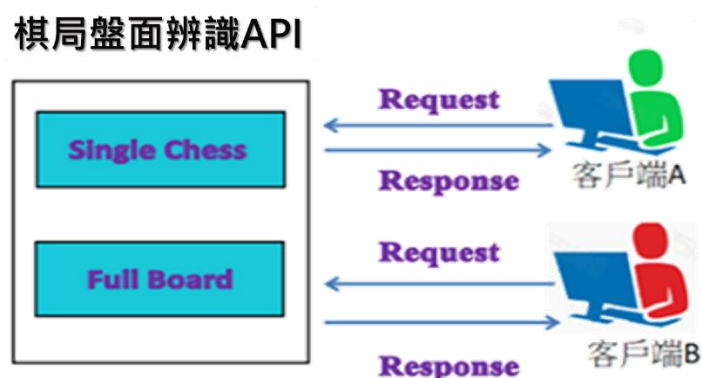


圖 28 棋局辨識函數 API 說明

在視覺 AI 的 API 裡有 SingleChess 函數及 FullBoard 函數可供客戶端選擇，客戶端的 Request 之 http 協議 data 為 IPCAM 之 URL，函數從 URL 中提取視訊影像，而 SingleChess 函數回傳為影像中辨識的單一棋子結果，JSON 格式為 {chess:x}；FullBoard 函數回傳為影像中辨識的棋局結果，JSON 格式為 {board:xxxxx...xxxxx}(長度為 32 的字串)。

實驗1-1 測試各個決策演算法與廠商開發的暗棋程式及人對戰的棋力

(一) 實驗說明：

1. 我方演算法有 Random 亂數、 $\alpha - \beta$ 兩層、四層、六層。
2. 測試對象為亂數：Alpha-Beta 兩層、四層、六層，與真人和 APP 對弈並記錄棋力。
3. 棋盤大小為 8*4，APP 為 IOS 手機版「暗棋王」，程式評價頗高。
4. 電腦軟體棋力分為：容易、適中、困難；人棋力分為：不會玩、有玩過。
5. 測試對象與電腦對弈各 10 局，對局結束後記錄非和局數據並計算勝率。

(二) 實驗結果：

表 9 決策演算法勝率數據(紀錄分出勝負之勝率共十場，和局則不記)

演算法 測試對象	Random 亂數	$\alpha - \beta$ 兩層	$\alpha - \beta$ 四層	$\alpha - \beta$ 六層
暗棋王(容易)	0%_和:0	60%_和:8	100%_和:4	100%_和:1
暗棋王(適中)	0%_和:0	40%_和:4	90%_和:1	100%_和:0
暗棋王(困難)	0%_和:0	0%_和:0	30%_和:0	50%_和:0
人(不會玩)	0%_和:0	80%_和:2	100%_和:1	100%_和:1
人(有玩過)	0%_和:0	10%_和:0	50%_和:3	60%_和:3



圖 29 黑方為 Alpha-Beta 六層與暗棋王(困難)對弈畫面

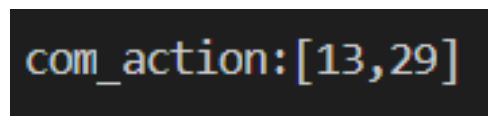


圖 30 電腦判斷與演算法吻合
(棋局為圖 29)

(三) 實驗討論：

測試了 Alpha-Beta 演算法的一些效能，觀察各個深度的效能，以單純的演算法去做評估，只能判斷吃子與移動的策略，如加入一些翻棋略後，與暗棋王對弈時，已經可以輾壓適中難度以下棋力，並且與困難難度並列相當；其中四層以上的深度與真人對弈的情況則有在正常人棋力水平之上了。

實驗1-2 測試學習演算法運用於暗棋對弈是否可以學會策略

(一) 實驗說明：

1. 設定強化學習的各項資訊，並讓零經驗的電腦進行左右互博訓練，。
2. 我方學習演算法為：QL(Q-table)、QL(Q-table) with MCTS、DRL、DRL with MCTS。
3. 各個演算法訓練 10000 場，並且每 100 場進行迭代(學習)。
4. 每 1000 場後與實驗一的演算法對弈 20 場(算非和局)，並且記錄勝率。

(二) 實驗結果：

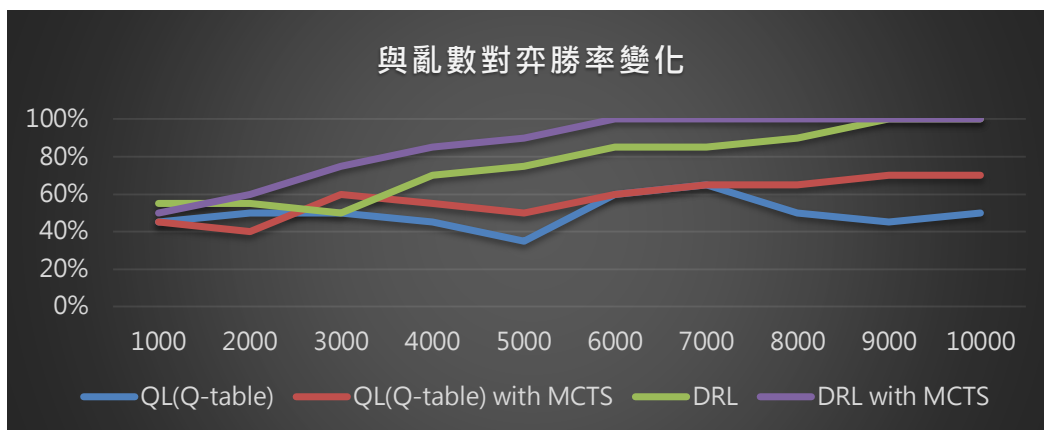


圖 31 亂數對弈勝率變化折線圖

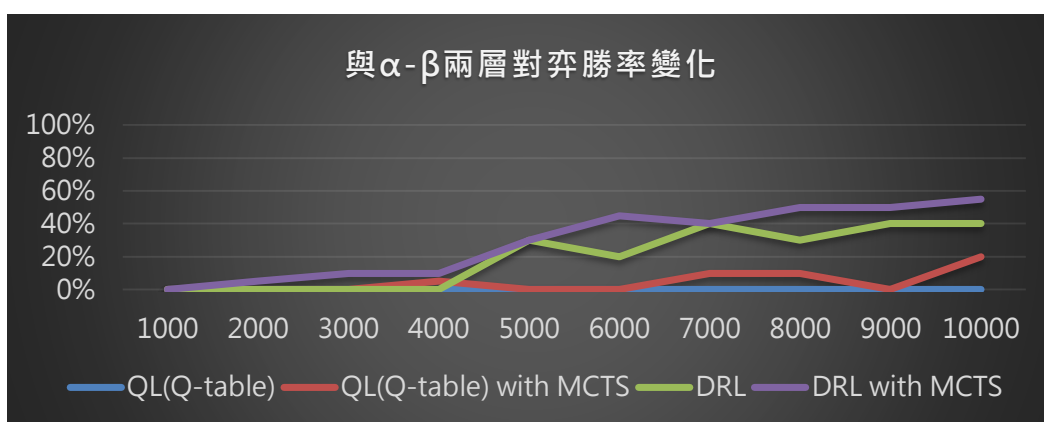


圖 32 α - β 兩層對弈勝率變化折線圖

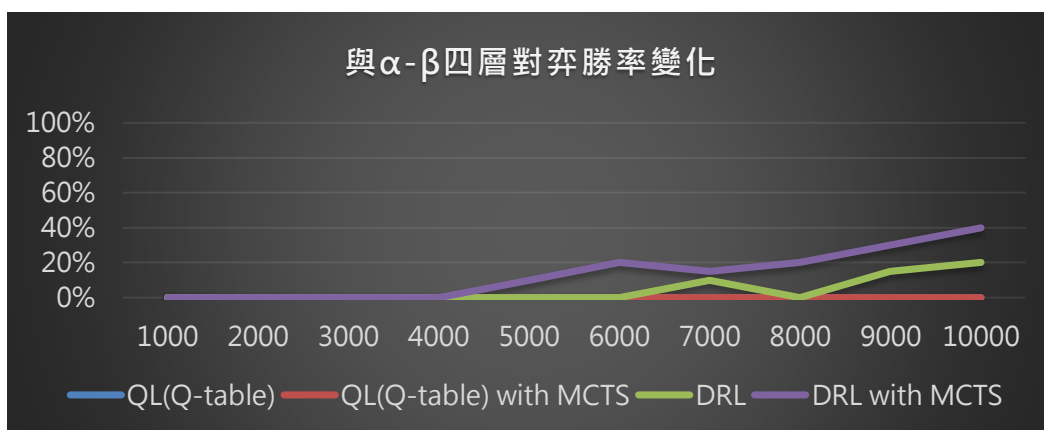


圖 33 α - β 四層對弈勝率變化折線圖

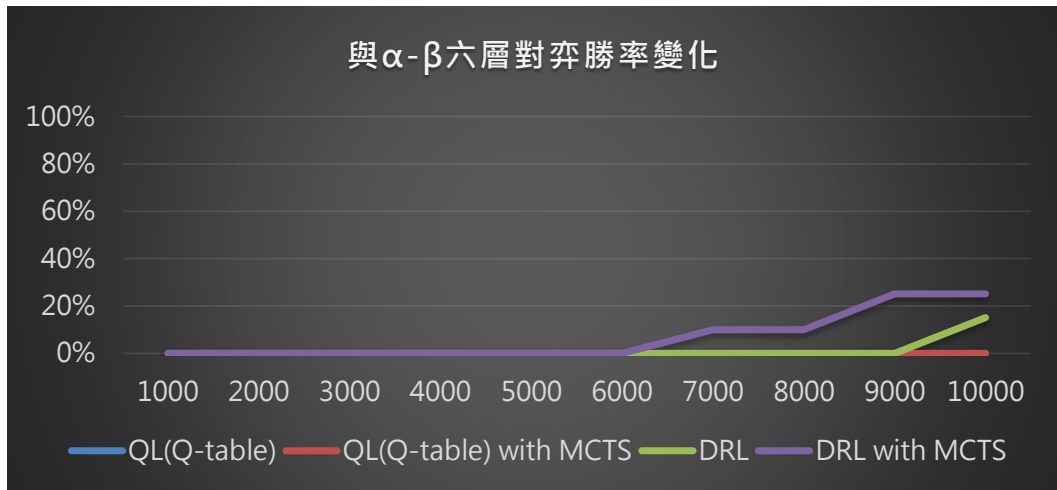


圖 34 $\alpha - \beta$ 六層對弈勝率變化折線圖

(三) 實驗討論：

(在實驗 1-2 中，和局變化將放置於附錄)。以 Q-table 學習猜測，因無法學習暗棋中龐大的狀態集，Q-table 中的內容是相對稀鬆的，但是 MCTS+UCT 去搜集棋譜後，資料有相對凝聚些。而在 DRL 方面，加上 MCTS+UCT 亦有較好的結果，但是僅限於對手較弱的情況，對手較強，AI 相對較難戰勝，猜測是 MCTS 或 DRL 中的參數未能達到效果，未來將繼續探討問題的根源。

實驗1-3 驗證將狀態數減少時，學習演算法是否可以學會策略，並且提高效能

(一) 實驗說明：

1. 銜接實驗 1-2 的全部說明。
2. 將棋盤改為 3*4 大小，並且設定棋子為：{將(帥)x1、士(仕)*1、馬(馮)x1、包(炮)x1 卒(兵)x2}共 12 顆棋子。

(二) 實驗結果：

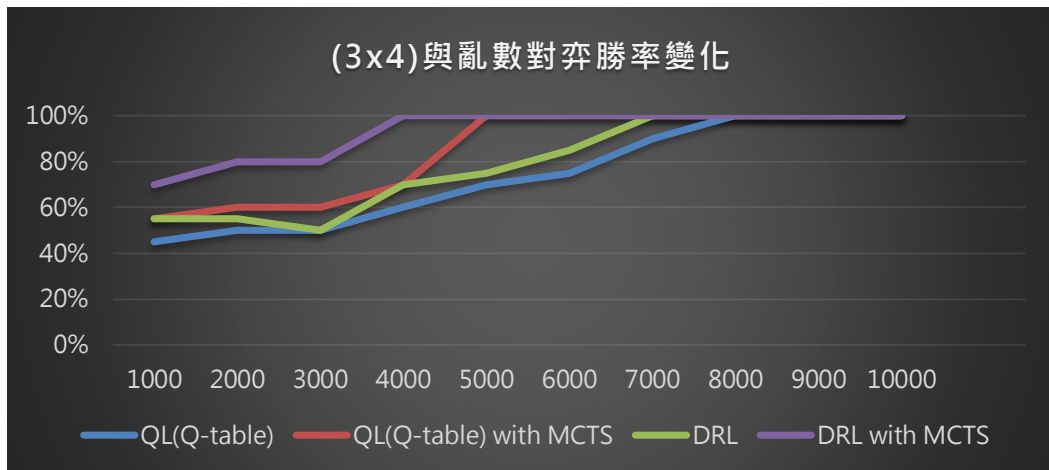


圖 35 (3x4)亂數對弈勝率變化折線圖

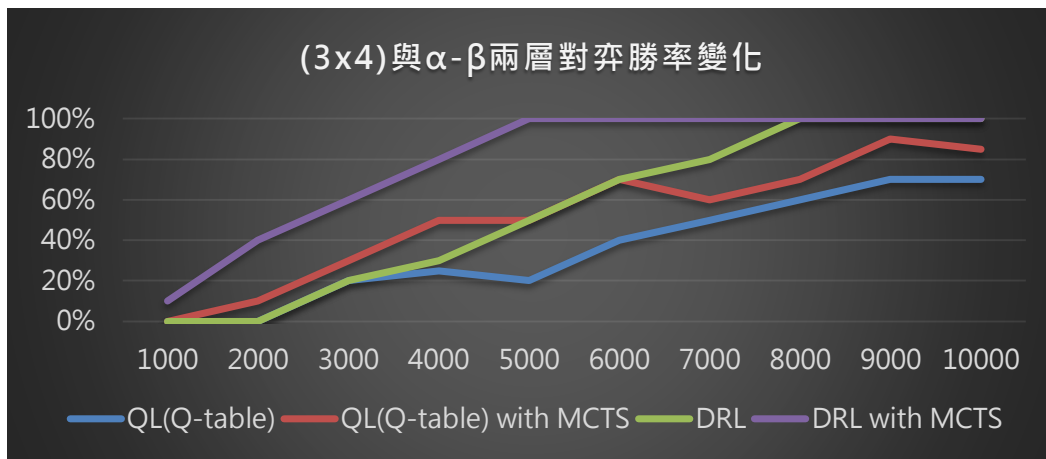


圖 36 (3x4) α - β 兩層對弈勝率變化折線圖

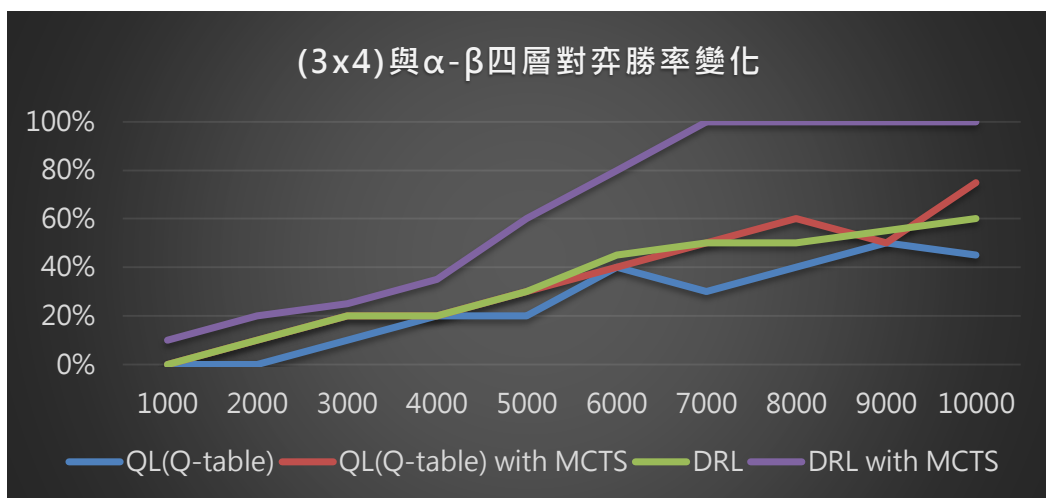


圖 37 (3x4) α - β 四層對弈勝率變化折線圖

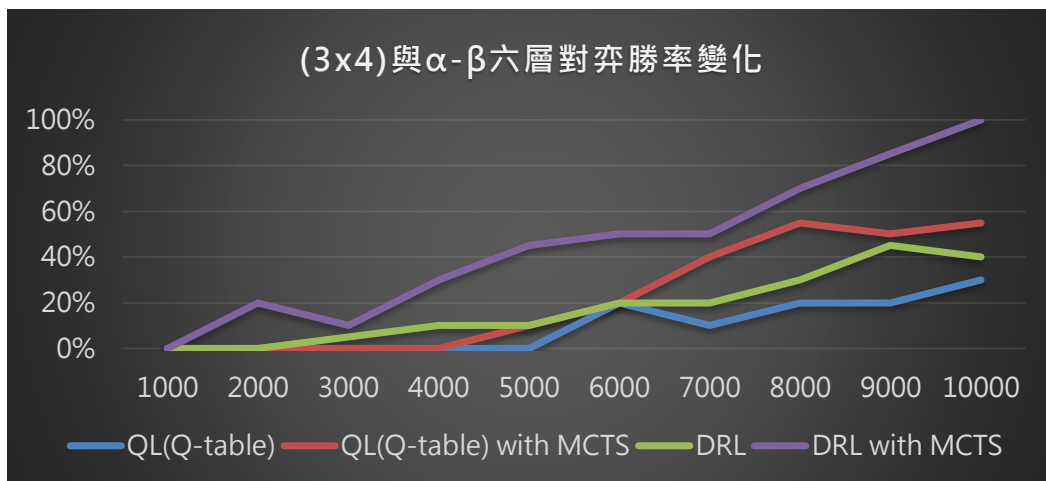


圖 38 (3x4) α - β 六層對弈勝率變化折線圖

(三) 實驗討論：

(在實驗 1-3 中，和局變化將放置於附錄)。驗證了將狀態減少時，Q-table 與 DRL 是可以學習的，印證了當 AI 要學會暗棋遊戲並壓制對手時，可能棋譜與訓練的時間要更加的多。

實驗2-1 驗證將神經網路深度加深，以 VGGNet 改版建構效能提高的程度

(一) 實驗說明：

1. 架構 AlexNet 改良網路(圖 24)與 VGGNet 改版網路(附錄圖 3)。
2. VGGNet 訓練樣本皆相同，每 360 batch 進行一次梯度下降，進行 15 回合。
3. 測試訓練好的網路，實際運用在已打亂明棋全盤面局，測試 20 次並且記錄數據。

(二)實驗結果：

於 Jupyter 上訓練結果：

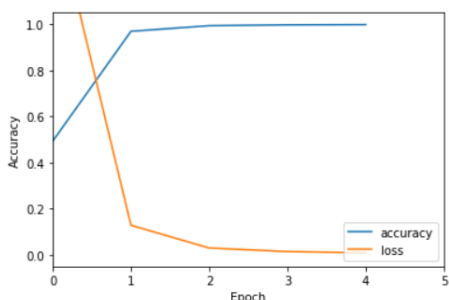


圖 39 AlexNet 改良神經網路訓練結果

(訓練時間：1076s)

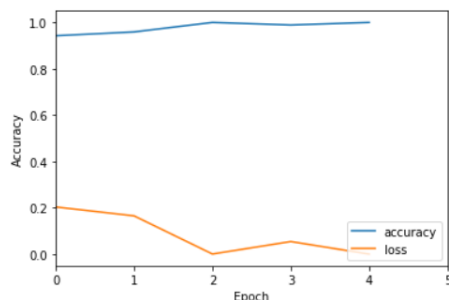


圖 40VGGNet 改良神經網路訓練結果

(訓練時間：15210s)

[0.00019519782929884496, 1.0]

圖 41 AlexNet 模型最終驗證

(陣列索引 0：loss, 陣列索引 1：accuracy)

[7.2336139225085255e-06, 1.0]

圖 42 VGGNet 模型最終驗證

實際於棋盤上辨識結果：

表 10 AlexNet 與 VGGNet 在各個棋子上辨識 20 次錯誤個數

棋子	帥	仕	相	俥	馮	炮	兵
AlexNet	2	3	1	2	1	3	6
VGGNet	0	0	3	0	1	1	0
棋子	將	士	象	車	馬	包	卒
AlexNet	1	1	2	3	3	4	8
VGGNet	0	0	1	0	0	1	0

表 11 NN 與 CNN 數據比對

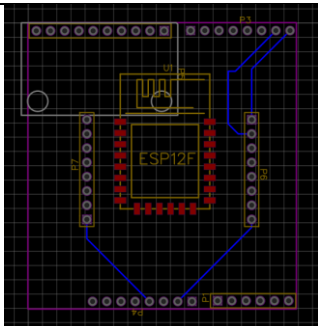
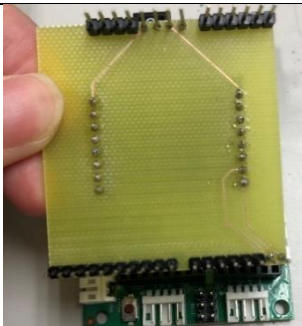
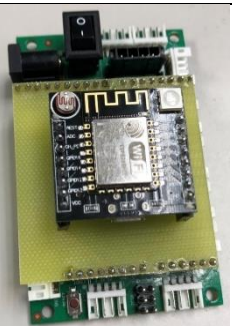

測試項目	模型種類	AlexNet 改良版	VGGNet 改良版
總辨識盤面(每盤 32 子)		20 盤	20 盤
辨識錯誤棋子總數目		40 枚	7 枚
平均辨識 100%成功機率		93.75%	98.9%

(三)實驗討論：

實測過後，在 Jupyter 上觀察訓練結果會發現兩個模型的正確率都相當的高，VGGNet 改良略勝於 AlexNet 改良。AlexNet 改良因深度沒那麼高，訓練時間相較是短的，而 VGGNet 改良則長了很多，到了實際測試，兩個模型的效能卻與數據上顯示略為不同。最後結論為加深了深度後的 VGGNet 改良確實比 AlexNe 改良更有效益，整體的實際成功率提高，由原本的 93.75%提升至 98.9%。

實驗3-1 AIoT 暗棋機器人遙控器製作流程

表 12 遙控器製作流程

			
繪製電路板，並且將電路圖 Layout	將電路板與 arduino 開發版進行焊接結合	將 ESP8266 針腳插入電路板中	物聯遙控器成品圖

實驗討論：

將電路規劃與遙控器外殼設計完，實際繪製電路板，以網站 EasyEDA 所繪製，並且存於 USB 後，放置機器進行實體電路繪製，設定與等待約一小時後，板子繪製完畢，將針腳部分進行焊接約一小時，完成後，於 3D 列印機取得遙控器外殼。最後組裝後，完成實品。

實驗3-2 AIoT 暗棋機器人整合及測試

表 13 翻棋流程

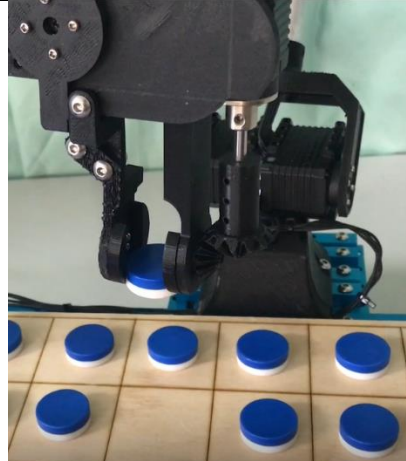
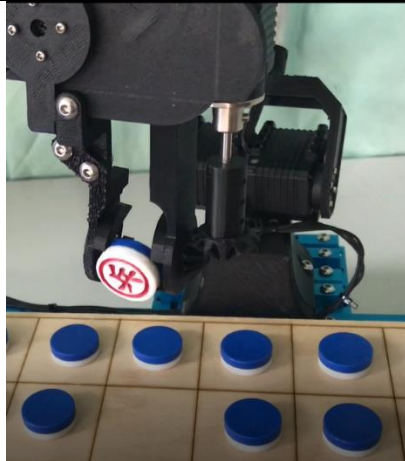

		
將棋子抓起	利用斜切齒輪進行翻轉	翻棋完畢

表 14 吃子流程(仕吃士)

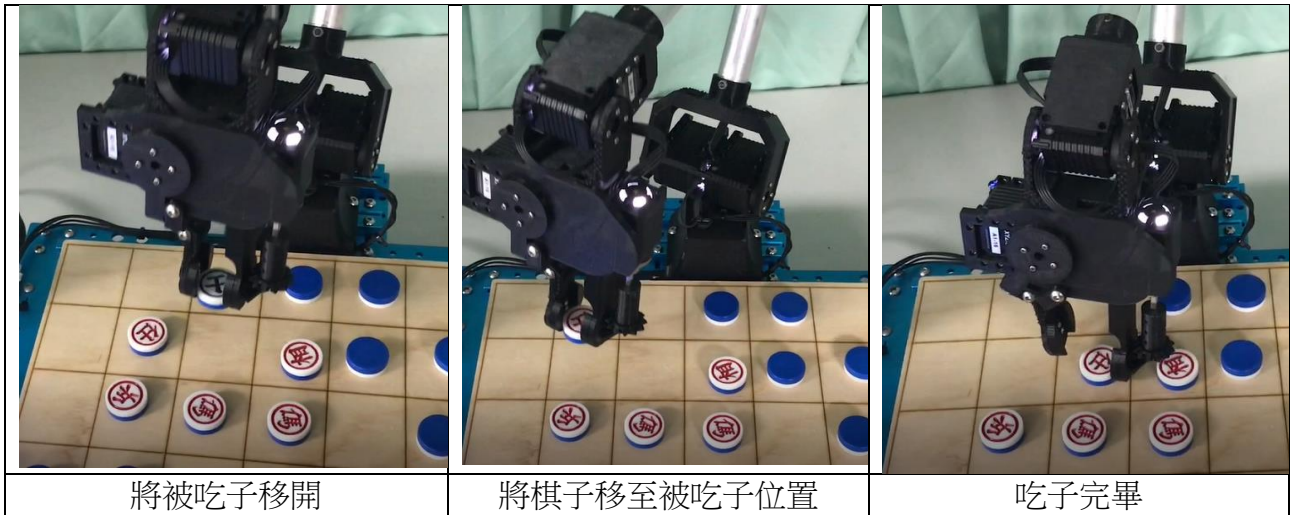


表 15 系統穩定度數據

玩家	影像辨識無誤	手臂吃子及移動無誤	手臂翻棋無誤
玩家一	Yes	Yes	Yes
	Yes	Yes	No
	Yes	Yes	Yes
平均三場	100%	100%	66.6%
玩家二	No	Yes	Yes
	Yes	Yes	Yes
	Yes	Yes	Yes
平均三場	66.6%	100%	66.6%
玩家三	Yes	Yes	Yes
	Yes	Yes	Yes
	Yes	Yes	Yes
平均三場	100%	100%	100%
總平均	77.733%	100%	77.733%
系統穩定度		85.155	

實驗討論：

觀察並紀錄與玩家對弈的情況，將各個情境圖記錄下，每個玩家都只開啟電源即可遊玩。在玩家一的對局中，手臂因定位稍為失準，出現了翻棋失誤的情況，該對局中玩家一因實力較強，戰勝了機器手臂；在玩家二對局中出現了辨識錯誤的情況，發現原因為玩家還沒等到手臂完整做完動作，而導致棋局辨識錯誤，該玩家實力也較不錯，但還是輸給機器手臂，玩家三對局情況較為樂觀，在過程中可以順利下完整局棋，玩家三實力較弱，因此輸給機器手臂。最終觀察，系統穩定度達到 85.155%。

肆、結論

將實體對弈機器人建構為 AIoT 架構，由於對弈及影像分類人工智慧已經建置成雲端的 API，可以簡化軟體的運算負擔。因此所有的裝置只要具備網路連線，便可以整合成一套複雜的人機對弈架構，任何程式開發者運用簡化的相關人工智慧的程式設計程序，著重於更細緻的流程及人機介面即可。運用 API 的過程，也可以收集大量的棋譜，對於深度強化學習有很大的助益，可以讓對弈人工智慧能力不斷的提升。希望未來可以將此 API 提供更多設計出人機對弈、電腦程式對弈的系統，造福更多愛下棋的大眾。以下為本研究成果：

- 一、本研究實現 AIoT 的概念設計的對弈機器人裝置，運用雲端服務強大的運算取得及擴充對弈人工智慧的能力，精簡硬體裝置，簡化及普及對弈機器人的產品開發或 Maker 自造的流程。
- 二、AIoT 對弈只需按下一個按鈕便可進行操作，當對 API 進行要求後，服務程式會抓取 IPCAM 影像，透過 CNN 進行盤面辨識，再將辨識結果及人工智慧的下一步棋路傳至客戶端，視覺辨識總成功率為 98.9%。
- 三、客戶端可輸入盤面資訊傳至 API，API 回傳最佳的棋步。對奕 AI 演算法的選擇有 Alpha-Beta 與 DRL+MCTS，本研究選用 Alpha-Beta(6 層)的暗棋人工智慧，與真人不同棋力具有 60%-100%的勝率，與暗棋王不同等級亦有 50%-100%的勝率。
- 四、機器手臂可以透過餘弦定理，將落地座標轉變成馬達的轉動角度，將馬達變為兩顆後，手臂的結構相對更加穩固，在抓取與翻棋的動作上失誤率下降許多。

伍、討論與應用

以下為在研究過程中所遇到的問題與其解決方法：

- 一、在與 Alpha-Beta 電腦遊玩當中，發現其動作只會判斷移動與吃子策略，爾後添加翻棋策略後，其勝率略為提高。
- 二、以學習演算法訓練電腦，發現其遊戲樹走向過於平衡，倒置學習速率過慢甚至無法學習，嘗試以 UCT 與 MCTS 蒐集訓練樣本後，學習效果略微提升，但其棋力仍不到正常棋力水準，未來將嘗試更改 MCTS 與 DRL 中變數，持續觀察與訓練。
- 三、影像可能會遇到陰暗的情況，導致辨識出問題，解決方式為利用 Python Image 套件將整體影像亮度調高。
- 四、因夾爪的元件過重導致手臂定位失準，倘若轉換成較輕型的馬達，將有效改善。
- 五、本研究 API 目前只能在區域網路內連線，未來若能將其開放到網際網路內，可能將

實現對奕機器人大眾化的理想，也可大量蒐集棋譜。

陸、參考資料

- [1] 古韋麟(2015)。國立交通大學多媒體工程研究所碩士論文 基於旋積類神經網路的影像搜尋應用中設計具鑑別能力的影像區塊挑選法(Discriminatively-learned Patch Selection for Image Retrieval with CNN)
- [2] 薛筑軒(2019)。國立交通大學資訊工程研究所博士論文 隨機型完全資訊遊戲對局程式之強度分析研究(On Strength Analyses of Computer Programs for Stochastic Games with Perfect Information)
- [3] 陳俊嶸(2010)。國立交通大學資訊工程研究所碩士論文 一個蒙地卡羅之電腦圍棋程式之設計(A Design Monte-Carlo Computer Go Program)
- [4] 謝曜安(2008)。國立臺灣師範大學資訊工程研究所碩士論文 電腦暗棋之設計及實作(The Design and Implementation of Computer Dark Chess)
- [5] 施宣丞(2012)。國立臺灣師範大學資訊工程研究所碩士論文 暗棋程式 DarkCraft 的設計與實作(The Design and Implementation of Dark Chess Program DarkCraft)
- [6] 吳天宇(2019)。國立臺灣師範大學資訊工程研究所碩士論文 基於 AlphaZero General Framework 實現 Breakthrough 遊戲(On Implementing Breakthrough Game Based on AlphaZero Feneral Framework)
- [7] 王鈞平(2019)。國立臺灣師範大學資訊工程研究所碩士論文 六貫棋遊戲實作與強化學習應用(Hex Game Implement and Reinforcement Learning)
- [8] 龔長偉(2011)。國立東華大學資訊工程研究所碩士論文 蒙地卡羅樹狀搜尋在暗棋上的應用(Monte Carlo Tree Search in Chinese Dark Chess)
- [9] 張俊雄、林后鍾 (2017)。程式設計實習。臺灣：台科大。
- [10] 王建堯、王家慶、吳信輝、李宏毅、高虹安、張智星、曾新穆、陳信希、蔡炎龍、鄭文皇、蘇上育 (2019)。人工智慧導論。新北市：全華圖書股份有限公司。
- [11] Rowel Atienza (2019)。深度學習使用 Keras。臺北市：基峰資訊股份有限公司。
- [12] Valentino Zocca、Gianmario Spacagna、Daniel Slater、Peter Roelants(2017)。Python 深度學習。新北市：博碩文化股份有限公司。
- [13] 黃一烜、鄧絜陽、陳皇宇(2017)。使用 UCT 演算法建構具學習能力的對弈機器人。中華民國第 57 屆中小學科學展覽會。
- [14] 陳冠學(2018)。使用機器手臂完成橋牌之人機對弈。第十七屆旺宏科學獎成果報告書。
- [15] alpha-zero-general github。網址：<https://github.com/suragnair/alpha-zero-general>。

附錄

表 1 硬體部分

編號	名稱	規格	數量	用途
1	AI 伺服馬達	A1-16	8 顆	作為手臂各個關節
2	Arduino 開發版	ATmege1280	1 塊	撰寫程式並控制手臂之開發板
3	智慧型手機	Iphone8	1 臺	用來當作攝像頭
4	筆記型電腦	MSI	1 臺	用來編寫、運作程式
5	3D 列印線材	PLA 材質	多捆	印製手臂材料必要素材
6	Makeblock 零件	Makeblock	多個	建構平台之零件
7	IoT 開發板	ESP8266	2 塊	連接雲端之開發版
8	象棋	塑膠材質	1 盒	本研究的主要棋類
9	棋盤	雷射切割之木板	1 塊	象棋的棋盤

表 2 軟體部分

編號	名稱	用途
1	Python3.7	人工智慧與影像辨識之程式語言
2	Csharp	介面設計之程式語言
3	OpenCV	撰寫影像處理之模組
4	IP 攝像頭簡化版(ios)	可進行 http 連接之手機程式，用於拍攝棋局
5	ArduinoIDE	設計手臂程式平台
6	Jupyter	python 設計平台可分段執行程式碼
7	Onshape	3D 繪圖網站
8	Visual Studio 2019	撰寫 C#設計平台
9	Visual Studio Code	撰寫 python 設計平台
10	Keras	訓練類神經網路與卷積神經網路模組，用於影像辨識
11	Flask	撰寫 API 之模組

表 3 筆電詳細規格

編號	名稱	規格
1	CPU	Intel(R) Core(TM)i7-10750H CPU @2.60Hz
2	GPU	NVIDIA GeForce GTX 1650 with Max-Q Design
3	記憶體	16.0GB
4	作業系統	Windows 10 家用版 64 位元

表 4 歷年對弈機器人差異

對弈機器人 技術項目	黑白棋機器人	橋牌機器人	AIoT 暗棋機器人
機器手臂	夾爪只能作抓取動作	藉由幫浦及吸盤做吸牌及放牌的動作	夾爪具有 翻轉 、抓取之動作，夾爪準備 申請專利
視覺辨識	利用圖片數值判斷黑子白子	運用 OCR 技術辨識撲克牌文字	利用 CNN 模型訓練即辨識 14 種棋子的字 ，暗子與空格則利用霍夫找圓判斷
人工智慧	利用 UCT 演算法進行決策判斷	運用 Monte Carlo 模擬並作出決策	使用 Alpha-Beta 與翻棋策略 作出決策，利用 RL 訓練電腦並測試效能
流程方式	需要 USB 連接電腦，一切運作皆封閉迴路	需要 USB 連接電腦，一切運作皆封閉迴路	將程式放置雲端變成 API，不需 USB 連接電腦， 連上 WI-FI 要求 API 即可遊玩
作品特點	以 UCT 演算法建構具有學習能力的自動對弈系統	以 OCR 技術建構撲克牌辨識系統	以 AIoT 概念建構對弈系統，一切操作只需按鈕即可搞定，並提供 API 給大眾使用

```

model = keras.Sequential([
    keras.layers.Conv2D(64,(3,3),strides=(1,1),input_shape=(56,56,1),padding='same',activation='relu'),
    keras.layers.Conv2D(64,(3,3),strides=(1,1),padding='same',activation='relu'),
    keras.layers.MaxPooling2D(pool_size=(2,2)),
    keras.layers.Conv2D(128,(3,3),strides=(1,1),padding='same',activation='relu'),
    keras.layers.Conv2D(128,(3,3),strides=(1,1),padding='same',activation='relu'),
    keras.layers.MaxPooling2D(pool_size=(2,2)),
    keras.layers.Conv2D(256,(3,3),strides=(1,1),padding='same',activation='relu'),
    keras.layers.Conv2D(256,(3,3),strides=(1,1),padding='same',activation='relu'),
    keras.layers.Conv2D(256,(3,3),strides=(1,1),padding='same',activation='relu'),
    keras.layers.MaxPooling2D(pool_size=(2,2)),
    keras.layers.Conv2D(512,(3,3),strides=(1,1),padding='same',activation='relu'),
    keras.layers.Conv2D(512,(3,3),strides=(1,1),padding='same',activation='relu'),
    keras.layers.Conv2D(512,(3,3),strides=(1,1),padding='same',activation='relu'),
    keras.layers.MaxPooling2D(pool_size=(2,2)),
    keras.layers.Flatten(),
    keras.layers.Dense(4096,activation='relu'),
    keras.layers.Dropout(0.5),
    keras.layers.Dense(4096,activation='relu'),
    keras.layers.Dropout(0.5),
    keras.layers.Dense(1000,activation='relu'),
    keras.layers.Dropout(0.5),
    keras.layers.Dense(14,activation='softmax'),
])

```

圖 1 利用 Keras 建構 VGGNet 網路改良版設定

```

history = model.fit(gray_frame1,gray_label1, epochs=15
                    ,batch_size=360)

```

```

Epoch 1/15
504000/504000 [=====] - 1461s 3ms/sample - loss: 1.0146 - acc: 0.6373
Epoch 2/15
504000/504000 [=====] - 1340s 3ms/sample - loss: 0.0488 - acc: 0.9875
Epoch 3/15
504000/504000 [=====] - 1034s 2ms/sample - loss: 0.0103 - acc: 0.9974
Epoch 4/15
504000/504000 [=====] - 1031s 2ms/sample - loss: 0.1059 - acc: 0.9867
Epoch 5/15
504000/504000 [=====] - 915s 2ms/sample - loss: 0.0033 - acc: 0.9994
Epoch 6/15
504000/504000 [=====] - 923s 2ms/sample - loss: 0.0674 - acc: 0.9891
Epoch 7/15
504000/504000 [=====] - 918s 2ms/sample - loss: 0.0032 - acc: 0.9995
Epoch 8/15
504000/504000 [=====] - 930s 2ms/sample - loss: 0.0012 - acc: 0.9998
Epoch 9/15
504000/504000 [=====] - 921s 2ms/sample - loss: 0.0437 - acc: 0.9926
Epoch 10/15
504000/504000 [=====] - 947s 2ms/sample - loss: 0.0013 - acc: 0.9998
Epoch 11/15
504000/504000 [=====] - 1019s 2ms/sample - loss: 5.9837e-04 - acc: 0.9999
Epoch 12/15
504000/504000 [=====] - 935s 2ms/sample - loss: 0.0160 - acc: 0.9970
Epoch 13/15
504000/504000 [=====] - 917s 2ms/sample - loss: 7.5249e-04 - acc: 0.9999
Epoch 14/15
504000/504000 [=====] - 929s 2ms/sample - loss: 2.8295e-04 - acc: 1.0000
Epoch 15/15
504000/504000 [=====] - 990s 2ms/sample - loss: 4.4302e-04 - acc: 0.9999

```

```

check_history = model.evaluate(gray_frame1, gray_label1)

```

```

504000/504000 [=====] - 371s 736us/sample - loss: 7.2336e-06 - acc: 1.0000- loss: 7.2395e-06 - acc: 1

```

圖 2 VGGNet 網路改良版訓練過程

=====		
conv2d (Conv2D)	(None, 56, 56, 64)	640
conv2d_1 (Conv2D)	(None, 56, 56, 64)	36928
max_pooling2d (MaxPooling2D)	(None, 28, 28, 64)	0
conv2d_2 (Conv2D)	(None, 28, 28, 128)	49280
conv2d_3 (Conv2D)	(None, 28, 28, 128)	147584
max_pooling2d_1 (MaxPooling2D)	(None, 14, 14, 128)	0
conv2d_4 (Conv2D)	(None, 14, 14, 256)	295168
conv2d_5 (Conv2D)	(None, 14, 14, 256)	590080
conv2d_6 (Conv2D)	(None, 14, 14, 256)	590080
max_pooling2d_2 (MaxPooling2D)	(None, 7, 7, 256)	0
conv2d_7 (Conv2D)	(None, 7, 7, 512)	1180160
conv2d_8 (Conv2D)	(None, 7, 7, 512)	2359808
conv2d_9 (Conv2D)	(None, 7, 7, 512)	2359808
max_pooling2d_3 (MaxPooling2D)	(None, 3, 3, 512)	0
flatten (Flatten)	(None, 4608)	0
dense (Dense)	(None, 4096)	18878464
dropout (Dropout)	(None, 4096)	0
dense_1 (Dense)	(None, 4096)	16781312
dropout_1 (Dropout)	(None, 4096)	0
dense_2 (Dense)	(None, 1000)	4097000
dropout_2 (Dropout)	(None, 1000)	0
dense_3 (Dense)	(None, 14)	14014
=====		
Total params:	47,380,326	
Trainable params:	47,380,326	
Non-trainable params:	0	
=====		

圖 3 VGGNet 網路改良版模型結構

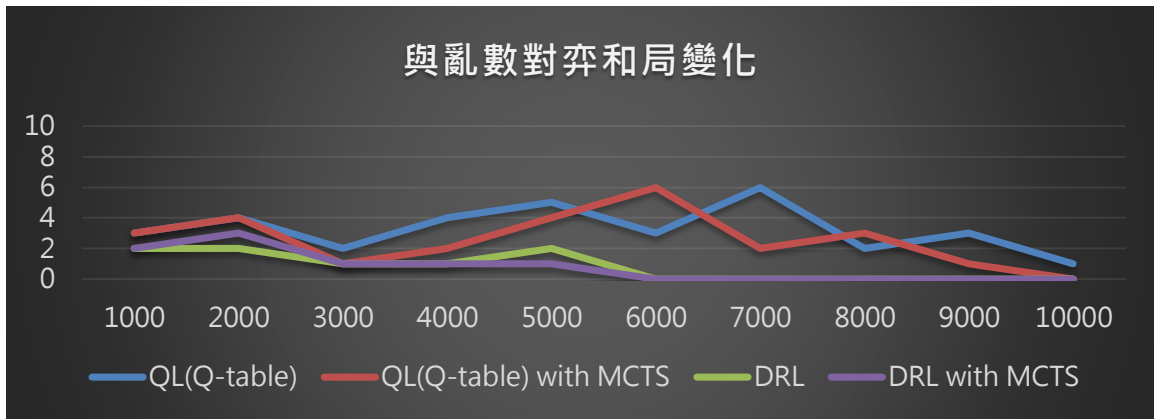


圖 4 亂數對弈和局變化折線圖

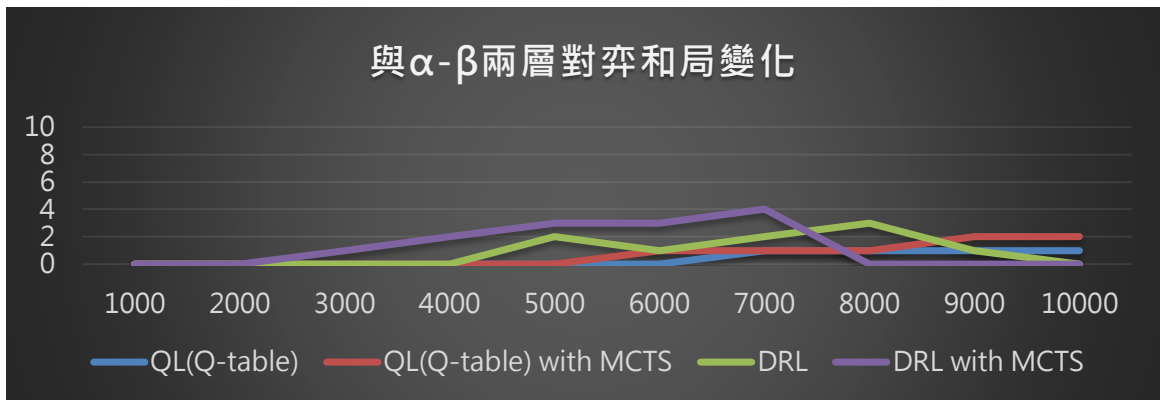


圖 5 α - β 兩層對弈和局變化折線圖

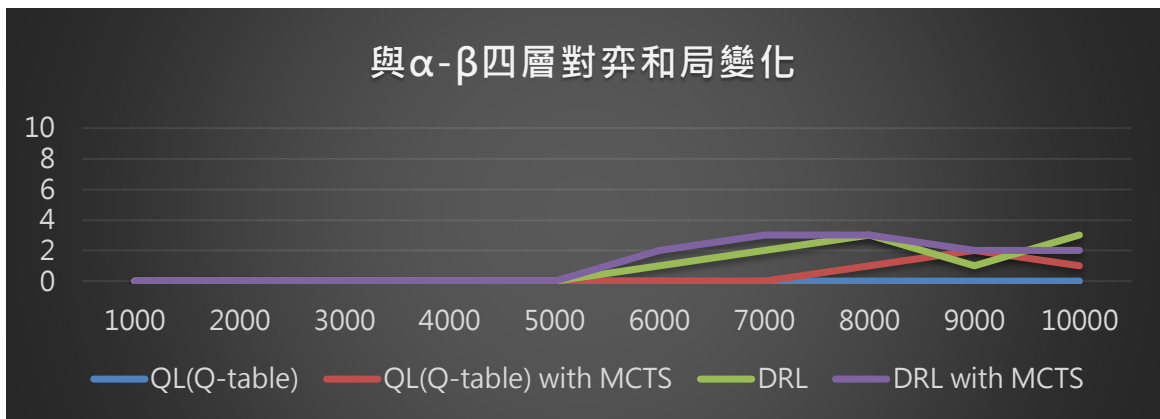


圖 6 α - β 四層對弈和局變化折線圖

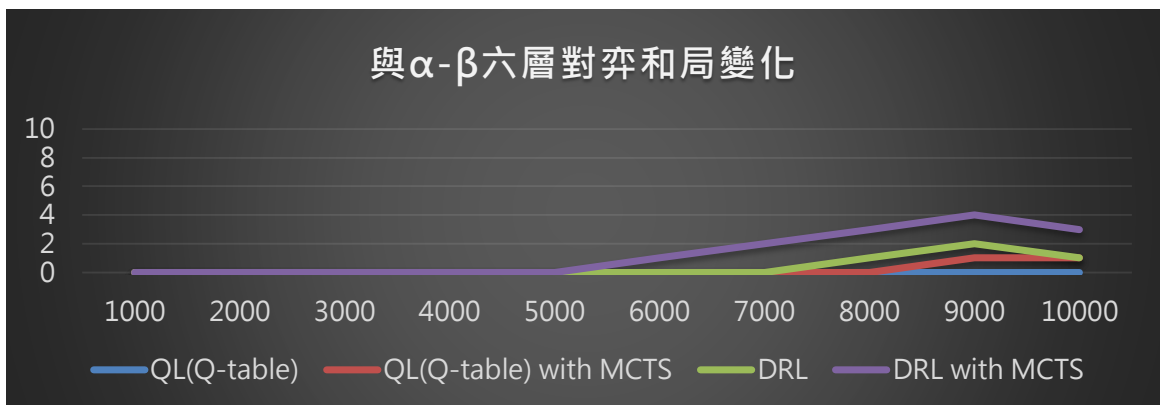


圖 7 α - β 六層對弈和局變化折線圖

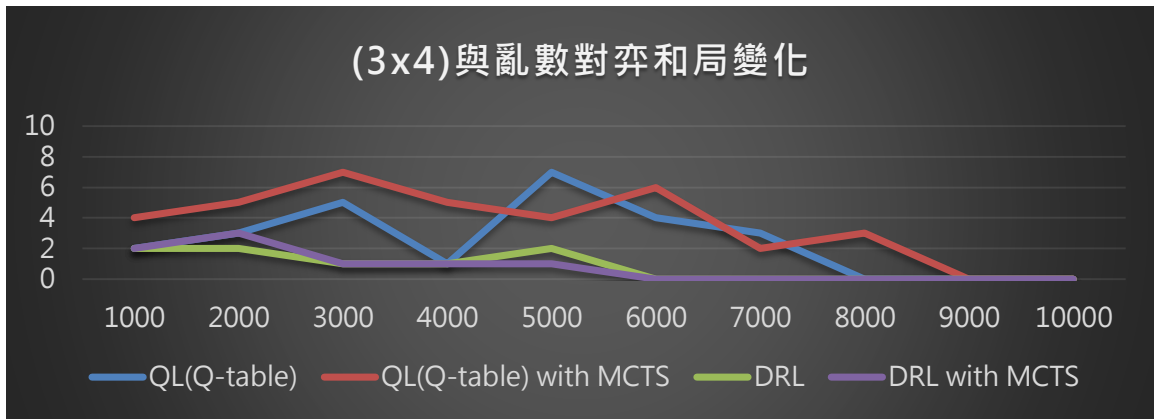


圖 8 (3x4)亂數對弈和局變化折線圖

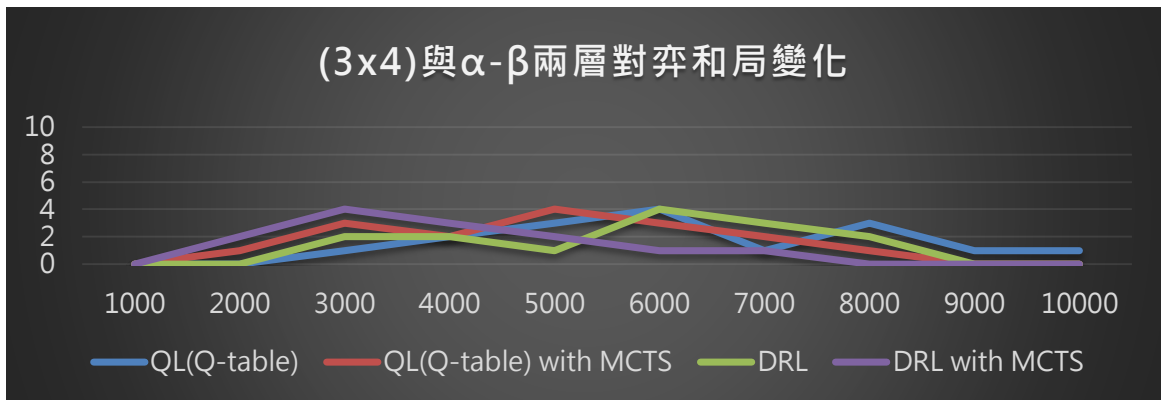


圖 9 (3x4) α - β 兩層對弈和局變化折線圖

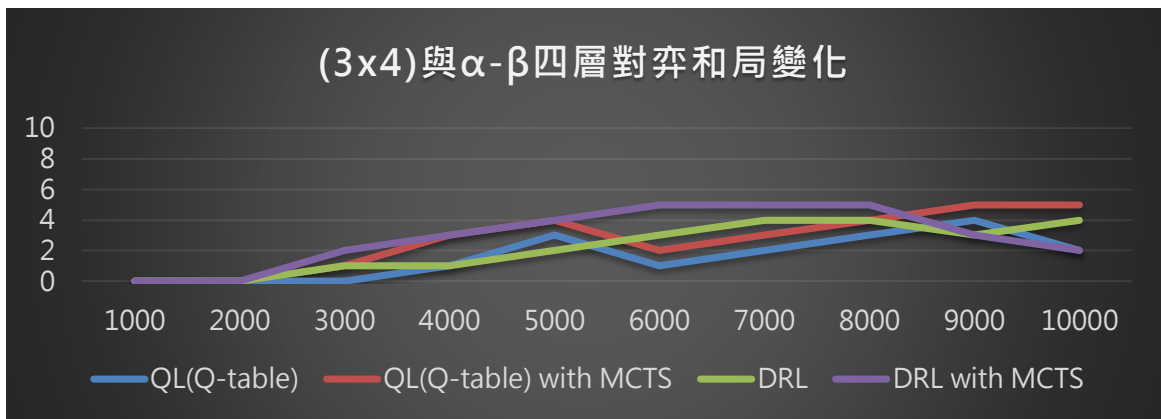


圖 10 (3x4) α - β 四層對弈和局變化折線圖

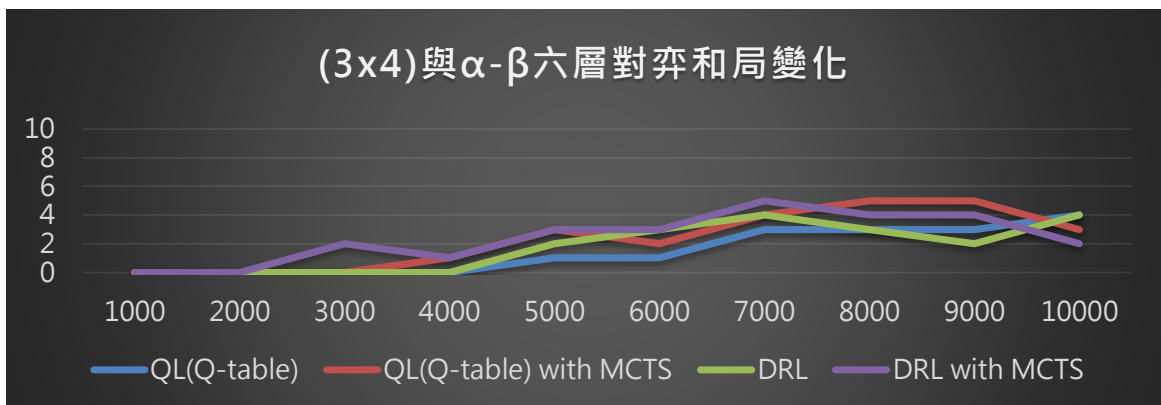


圖 11 (3x4) α - β 六層對弈和局變化折線圖

公園老人訪問

DATE / /

公園						
喜歡下棋	正 下	正 正	正 正	正 下	正	喜歡 39人 不喜歡 11人 共 50人
不喜歡或其他	下	—	下	—	正	

喜歡的棋類(不喜歡下棋的也選一種)

公園						人數
象棋	正 —	下	正 下	下	正 下	20人
圍棋	正	正	正 —	下	下	18人
西洋棋		下	—			3人
五子棋		下	下	正	—	9人

圖 12 訪問紀錄



圖 13 完成智慧物聯暗棋機器人作品