花蓮縣第 61 屆國民中小學科學展覽會

作品說明書

科　　別： 生活與應用科學科(一)(機電與資訊)

組　　別： 國中組

作品名稱： 提升先手勝率於「拈」的演算法之研究與程式開發

關 鍵 詞： 拈、勝率、演算法

摘要

我們的研究是開發演算法用於消圈圈(拈)的遊戲，目標是提升電腦在遊戲中先手時的勝率，讓電腦下棋變得更聰明、更有智慧。首先，我們先以隨機的策略針用電腦對電腦的方式對不同遊戲場數時先手的勝率進行分析，再來，針對不同堆疊數進行先手平均勝率的探討。最後，我們開發了三個演算法，對 3 層、4 層、5 層堆疊數進行勝率提升效果的分析，以電腦對電腦的方式進行多場次的對弈，實驗結果發現我們開發的演算法確實能夠有效地提升勝率，並且開發人機互動程式，讓我們有機會可以與電腦進行對弈。

壹、研究動機

小時後玩過一種叫消圈圈(拈)的遊戲，遊戲規則是：有兩位玩家和一張紙，紙上有一個由 15 個圓圈排成的正三角形，決定誰先誰後，每人可畫一條劃過 1~3 個圓圈的直線，畫到最後一個圓圈的玩家為輸家。這個遊戲看似簡單，卻隱藏了很多勝利的技巧。對於演算法有興趣的我們，於是我們開始思考，能不能自己開法遊戲，讓我們有機會跟電腦下棋呢？以及如何讓電腦變得較為聰明呢？於是開啟了我們的研究之路。

貳、研究目的

一、利用 Python 程式語言撰寫人機互動遊戲

二、撰寫當電腦先下時，提升平均勝率的演算法程式開法

三、探討演算法對各種拈堆疊數的勝率影響

參、研究設備及器材

一、可執行 Python 語言的電腦環境(使用 Visual Studio Code 的 IDE 進行開法)

二、筆以及計算紙(用於人機互動時結果的呈現)

肆、研究方法

一、遊戲規則說明：

本遊戲的每一個回合可以消去 1 到 3 個圈圈，其中只能以直線消去，且不能夠轉折(如圖 1)或穿越圈圈(如圖 2)，目標是留下最後一個圈圈給對方，對方消去最後一個圈圈則對方輸。例如：若 A 先下(順序：A1->B1->A2->B2->A3->B3->A4->B4->A5)，根據圖 3 則 A 最後輸。

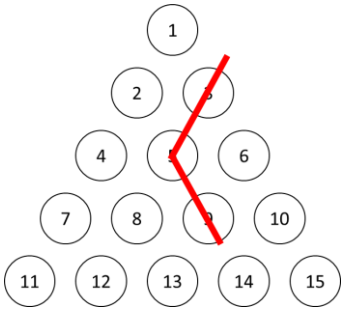

圖 1

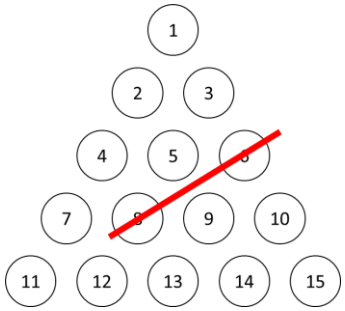

圖 2

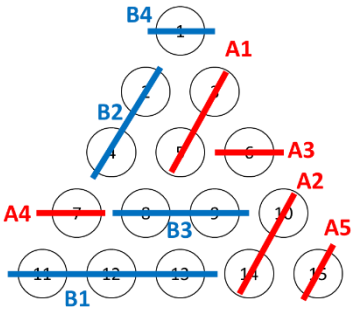

圖 3

二、隨機策略(程式碼儲存於附錄，文章裡的紅字表示函式、藍色表示變數)

(一) 先把圈圈編號，以利後續電腦的邏輯處理，如上圖 1 為五層圈圈編號示意圖。

(二) 預先以 genChoice(layer)產生可以刪除的各種選擇(oneChoice、twoChoice、thrChoice)

(三) 電腦先以隨機方式先選擇要刪除個數(deleteNum)，例如：1 個、2 個或 3 個圈圈。

(四) 個數決定後，再決定要刪除的圈圈號碼(readyToDeleteSet)。

(五) 將要刪除的圈圈記錄下來，儲存於 deleteOrder。

(六) 抽出已經被刪除圈圈所對應的各種選擇(choiceExtract())，電腦下次就不會選到。

(七) 如果圈圈還沒有被消完，則換另一位玩家。

(八) 之後重複步驟一到步驟七，直到所有圈圈都被消完。

三、演算法一(策略 1：strategy_vicChess(vicChessList, deleteOrder))

(一) 預先以 genChoice(layer)產生可以刪除的各種選擇(oneChoice、twoChoice、thrChoice)

(二) 以 genVicChess(vicChessNum, layer)產生之後先手可以參考的勝利棋譜。此處勝利棋譜的產生是以隨機策略的方式讓電腦與電腦自行對決，並把先下而且最後獲勝的棋譜儲存下來(vicChessList)，儲存的數量可以預先設定(vicChessNum)。

(三) 先下的電腦會從這些勝利棋譜中隨機挑選一個下第一步

(四) 下完後電腦會抽出與目前步驟相同的棋譜(vicChessExtract(vicChessList_temp, deleteOrder))，所以可以參考的勝利棋譜也會越來越少。

(五) 後下的電腦會以隨機策略下第二步，下完後一樣執行 vicChessExtract()抽出勝利棋譜

(六) 換先下的電腦，若還有勝利棋譜可以參考，則根據勝利棋譜隨機挑選其中一個再刪除圈圈；如果已經沒有勝利棋譜可以參考，則採用隨機策略。

(七) 後下的電腦仍然以隨機策略繼續刪除圈圈

(八) 之後重複步驟一到步驟七，直到所有圈圈都被消完。

四、演算法二(策略 1 + 策略 2：strategy_failChess(failChessList, deleteOrder, choiceListNow))

(一) 預先以 genChoice(layer)產生可以刪除的各種選擇(oneChoice、twoChoice、thrChoice)

(二) 以 genVicChess(vicChessNum, layer)產生之後先手可以參考的勝利棋譜。此處勝利棋譜的產生是以隨機策略的方式，讓電腦與電腦自行對決，並把先下而且最後獲勝的棋譜儲存下來(vicChessList)，儲存的數量可以預先設定(vicChessNum)。

(三) 以 genFailChess(failChessNum, layer)產生之後先手可以參考的失敗棋譜。此處失敗棋譜的產生是以隨機策略的方式，讓電腦與電腦自行對決，並把先下而且最後失敗的棋譜儲存下來(failChessList)，儲存的數量可以預先設定(failChessNum)。

(四) 先下的電腦會從這些勝利棋譜中隨機挑選一個下第一步

(五) 下完後電腦會抽出與目前步驟相同的勝利棋譜(vicChessExtract(vicChessList_temp, deleteOrder))，所以可以參考的勝利棋譜也會越來越少；同時抽出與目前步驟相同的失敗棋譜(failChessExtract(failChessList_temp, deleteOrder))，所以可以參考的失敗

棋譜也會越來越少。

(六) 後下的電腦會以隨機策略下第二步，下完後一樣抽出勝利棋譜與失敗棋譜

(七) 換先下的電腦，若還有勝利棋譜可以參考，則根據勝利棋譜隨機挑選其中一個再刪除圈圈；如果已經沒有勝利棋譜可以參考，則採用策略 2，即參考失敗棋譜，避免做出與失敗棋譜相同的選擇，也因此策略 2 也只會被執行到一次，因為若避開與失敗棋譜相同的選擇，後面也不會再有相對應的失敗棋譜可以參考；如果沒有失敗棋譜可以參考，則採用隨機策略。

(八) 後下的電腦仍然以隨機策略繼續刪除圈圈

(九) 之後重複步驟一到步驟八，直到所有圈圈都被消完。

五、演算法三(策略 1 + 策略 2 + 策略 3：strategy_oddNumber(deleteOrder, choiceListNow, totalCircle))

演算法三大致上與演算法二雷同，只是換先下的電腦，若還有勝利棋譜可以參考，則根據勝利棋譜(策略 1)隨機挑選其中一個再刪除圈圈；如果已經沒有勝利棋譜可以參考，則參考失敗棋譜(策略 2)，避免做出與失敗棋譜相同的選擇；如果也沒有失敗棋譜可以參考，最後則採用奇數原則(則當最後只剩下奇數個單個圈圈留給對方時，則我方獲勝，如下圖 4)，因此若剩下的圈圈有奇數個，則刪除偶數個圈圈(即刪除 2 個)、若剩下的圈圈有偶數個，則刪除奇數個圈圈(即刪除 1 或 3 個，以隨機的方式選擇)，直到所有圈圈都被消完。

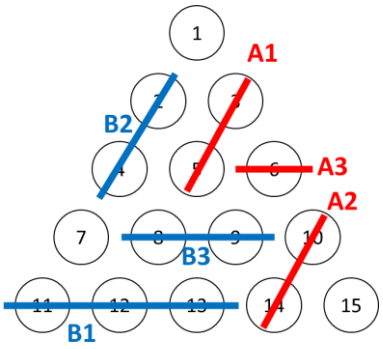

圖 4

伍、研究結果

一、先手勝率與遊戲次數的關係(隨機策略 V.S 隨機策略)

以「隨機策略」針對 5 層堆疊進行「先手勝率」與「遊戲次數」的關係分析

表 1

	100 場	1000 場	10000 場	100000 場
第一次	0.55	0.514	0.4934	0.49872
第二次	0.40	0.476	0.4965	0.49921
第三次	0.52	0.500	0.4964	0.49726
第四次	0.46	0.488	0.4959	0.49764
第五次	0.44	0.467	0.4929	0.49472
平均	0.474	0.5056	0.49502	0.49751
標準差	0.0607	0.0103	0.00173	0.00175

二、圈圈層數對勝率的影響

以「隨機策略」進行「先手勝率」與「各種堆疊數」的進行分析(每次 10000 場)

表 2

	3 層堆疊	4 層堆疊	5 層堆疊	6 層堆疊
第一次	0.3965	0.4628	0.4934	0.4959
第二次	0.4051	0.4582	0.4965	0.4959
第三次	0.4042	0.4642	0.4964	0.4960
第四次	0.3982	0.4580	0.4959	0.5034
第五次	0.4046	0.4666	0.4929	0.5025
平均	0.4017	0.4620	0.4950	0.4987

三、演算法一對勝率的影響

以「演算法一」針對各層堆疊進行「先手勝率」分析，其中每次的平均勝率為進行1000 場遊戲所得的結果，總共執行 3 次分析，而 3 次中所參考的勝利棋譜都是用同一個。

(一) 針對 5 層堆疊進行「先手勝率」與「勝利棋譜數量」的關係分析

表 3

	100 個	1000 個	10000 個	100000 個
第一次	0.499	0.516	0.512	0.504
第二次	0.487	0.508	0.503	0.517
第三次	0.507	0.497	0.485	0.534
平均	0.498	0.507	0.500	0.518

(二) 針對 4 層堆疊進行「先手勝率」與「勝利棋譜數量」的關係分析

表 4

	100 個	1000 個	10000 個	100000 個
第一次	0.481	0.544	0.613	0.661
第二次	0.468	0.561	0.608	0.695
第三次	0.489	0.537	0.622	0.665
平均	0.479	0.547	0.614	0.674

(三) 針對 3 層堆疊進行「先手勝率」與「勝利棋譜數量」的關係分析

表 5

	100 個	1000 個	10000 個	100000 個
第一次	0.596	0.658	0.660	0.689
第二次	0.587	0.663	0.671	0.675
第三次	0.592	0.681	0.676	0.666
平均	0.592	0.667	0.669	0.677

四、演算法二對勝率的影響

以「演算法二」針對各層堆疊進行「先手勝率」分析，其中每次的平均勝率為進行 1000 場遊戲所得的結果，總共執行 3 次分析，而 3 次中所參考的勝利棋譜以及失敗棋譜都是用同一個，其中勝利的棋譜數量與失敗的棋譜數量一樣多。

(一) 針對 5 層堆疊進行「先手勝率」與「勝利棋譜與失敗棋譜數量」的關係分析

表 6

	100 個	1000 個	10000 個	100000 個
第一次	0.500	0.492	0.524	0.541
第二次	0.491	0.512	0.508	0.520
第三次	0.493	0.503	0.498	0.537
平均	0.495	0.502	0.510	0.533

(二) 針對 4 層堆疊進行「先手勝率」與「勝利棋譜數量」的關係分析

表 7

	100 個	1000 個	10000 個	100000 個
第一次	0.481	0.574	0.600	0.661
第二次	0.49	0.524	0.619	0.681
第三次	0.505	0.552	0.596	0.662
平均	0.492	0.550	0.605	0.668

(三) 針對 3 層堆疊進行「先手勝率」與「勝利棋譜數量」的關係分析

表 8

	100 個	1000 個	10000 個	100000 個
第一次	0.613	0.689	0.695	0.680
第二次	0.593	0.684	0.695	0.710
第三次	0.593	0.662	0.678	0.672
平均	0.600	0.678	0.689	0.687

五、演算法三對勝率的影響

以「演算法三」針對各層堆疊進行「先手勝率」分析，其中每次的平均勝率為進行 1000 場遊戲所得的結果，總共執行 3 次分析，而 3 次中所參考的勝利棋譜以及失敗棋譜都是用同一個。

(一) 針對 5 層堆疊進行「先手勝率」與「勝利棋譜數量」的關係分析

表 9

	100 個	1000 個	10000 個	100000 個
第一次	0.583	0.549	0.598	0.611
第二次	0.561	0.595	0.600	0.576
第三次	0.545	0.572	0.586	0.593
平均	0.563	0.572	0.595	0.593

(二) 針對 4 層堆疊進行「先手勝率」與「勝利棋譜數量」的關係分析

表 10

	100 個	1000 個	10000 個	100000 個
第一次	0.580	0.626	0.663	0.686
第二次	0.584	0.671	0.695	0.693
第三次	0.567	0.643	0.659	0.667
平均	0.577	0.647	0.672	0.682

(三) 針對 3 層堆疊進行「先手勝率」與「勝利棋譜數量」的關係分析

表 11

	100 個	1000 個	10000 個	100000 個
第一次	0.680	0.668	0.663	0.695
第二次	0.664	0.685	0.689	0.678
第三次	0.664	0.673	0.699	0.689
平均	0.669	0.675	0.684	0.687

陸、討論

一、隨機策略結果分析

(一) 平均勝率與遊戲次數的關係

根據表 1 所繪製如下圖 5 可以發現，以隨機策略對決隨機策略的測試結果發現，當遊戲場數越多時，先手的勝率會越來越趨近於 0.5，也就是說雙方在沒有任何策略的情況下，兩者的勝率幾乎是一樣的，從一個遊戲的角度來看，這似乎算是一個公平的遊戲，不會因為誰先誰後使得勝率雙方不相同。

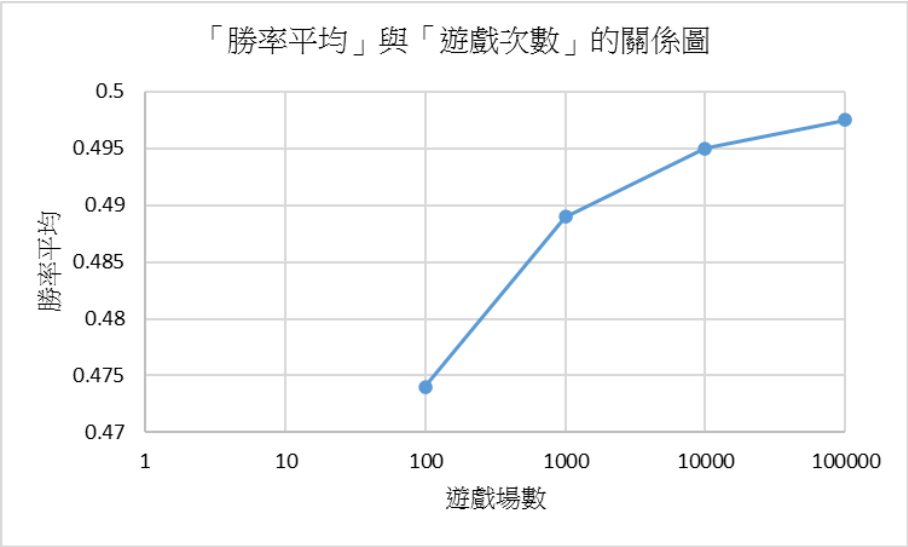

圖 5

(二) 平均勝率的標準差與遊戲次數的關係

根據表 1 所繪製的圖 6 可以發現，隨著遊戲次數越多，平均出來的數值之間會越接近，也就是說標準差的值會比較小，例如如果只用 100 場來平均的話，每次出來的數值會比較不集中，而從數據也可以觀察到，用 10000 場與 100000 場所得到標準差的值結果差不多，也就是多以 10000 場所得到的平均結果應該是滿穩定的，平均勝率的結果可信度也比較高，不過相對的，進行 10000 場則遊戲時間會較久(這部分之後會繼續研究平行運算，讓運算加速)，在後續的演算法對決隨機策略的平均勝率分析上，我們採用 1000 場來看不同演算法對於勝率的影響。

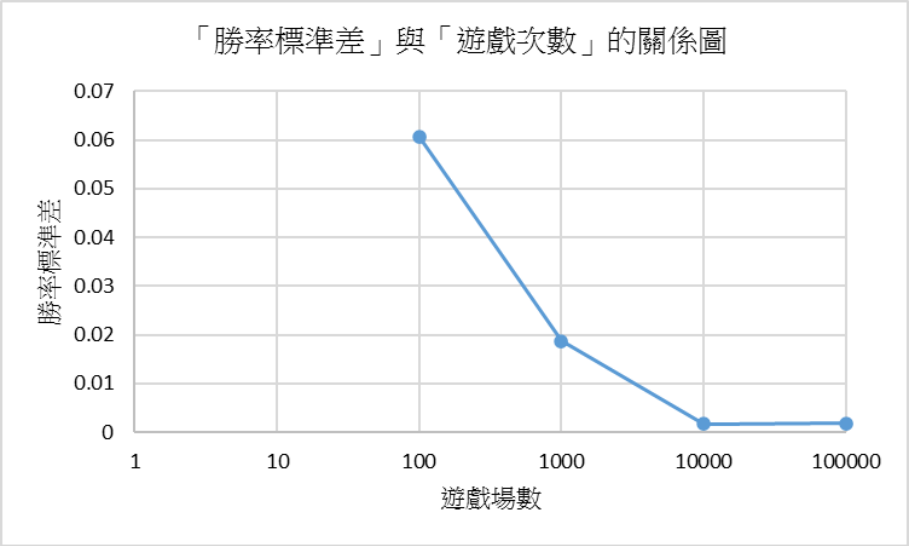

圖 6

(三) 隨機策略對各種堆疊數的平均勝率分析

根據表 2 所繪製的圖 7 可以發現，若圈圈的堆疊數越少，先手的勝率越低，以 3 層堆疊為例，進行 10000 場遊戲，先手的平均勝率大約只有 0.402 左右，而 4 層堆疊則是 0.462，以一個遊戲來說，3 層堆疊與 4 層堆疊並不是一個公平的遊戲。不過當堆疊數來到 5 層與 6 層時，平均勝率分別為 0.495 與 0.499，非常接近 0.5，其實當我做出這個結果時，我們團隊都很佩服當初設計遊戲的人，怎麼知道 5 層堆疊是一個公平的遊戲，直到現在，消圈圈遊戲都還是以 5 層堆疊為主。

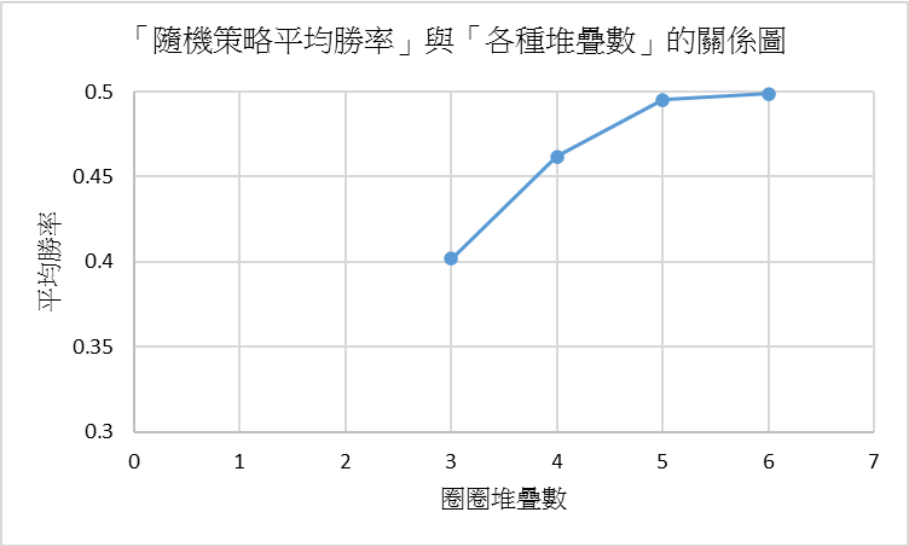

圖 7

二、演算法對各種堆疊的平均勝率分析

根據測試結果，由下圖 8 可以發現，演算法一(即參考勝利棋譜做出選擇)，當參考的勝利棋譜數量越多，各層堆疊的平均勝率皆可以有效提升，其中 3 層堆疊與 4 層堆疊的平均勝率增加的幅度較為明顯。3 層堆疊在勝利棋譜數量為 100 筆時的勝率已經接近 6 成，4 層堆疊在勝利棋譜數量為 10000 筆時的勝率也超越 6 成，而 5 層堆疊在勝利棋譜數量為 10000 筆時勝率有稍微掉下來，可見 1000 筆與 10000 筆的效果差不多，當 100000 筆時勝率有提升到 0.518 左右，可見參考的棋譜數量要到一定程度效果才會出來。不過當參考的勝利棋譜數量超過 1000 筆時，演算法一對各層堆疊的平均勝率皆超過 5 成。

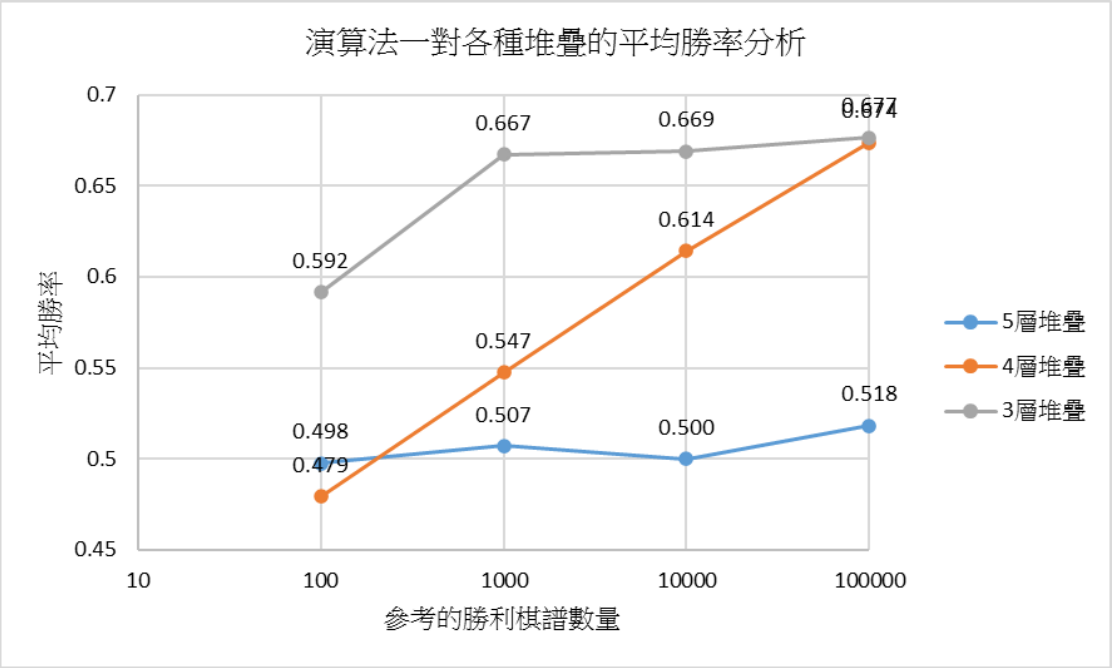

圖 8

由下圖 9 可以發現，演算法二(即參考勝利棋譜與失敗棋譜做出選擇)，當參考的勝利棋譜數量越多，各層堆疊的平均勝率仍然可以有效提升。3 層堆疊在勝利與失敗棋譜數量為 100 筆時的勝率已到達 6 成，不過跟演算法一相似，當棋譜數量超過 1000 筆之後，勝率提升的效果較為趨緩，不過已經接近 7 成勝率。4 層堆疊隨著勝利棋譜數量變為 10 倍時，勝率跟演算法一相似，勝率提升的幅度都差不多(比較像是線性增加，一條斜直線)，大約都增加 0.05 左右。而 5 層堆疊在棋譜數 100000 筆時勝率提升到 0.533 左右，演算法二隨著勝利棋譜數量增加，對各種堆疊的勝率都能有效提升。

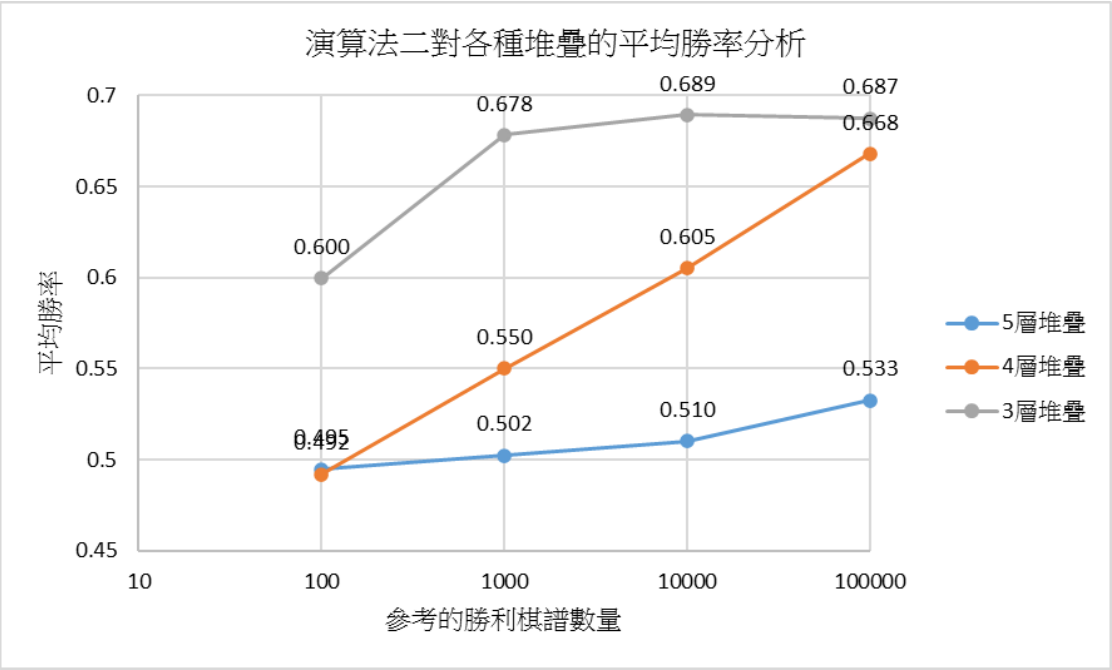

圖 9

由下圖 10 可以發現，演算法三(即參考勝利棋譜、失敗棋譜與奇數原則做出選擇)，3 層堆疊在勝利與失敗棋譜數量為 100 筆時的勝率已到達 6 成 7。4 層堆疊在棋譜數量為 1000 筆時，勝率已到達 0.647 左右，最後勝率趨近於 3 層堆疊。而 5 層堆疊在棋譜數 10000 筆時勝率提升到 6 成左右，由此可知演算法三對於 5 層堆疊的勝率可以有效提升。

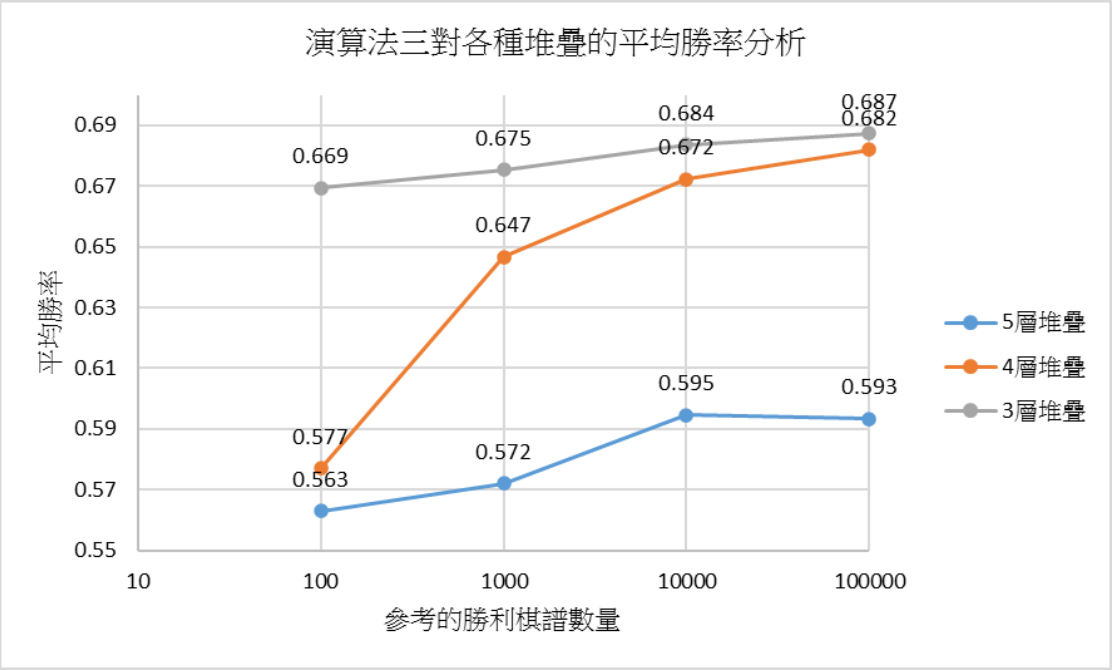

圖 10

三、各種堆疊對演算法的平均勝率分析

由下圖 11 可以發現，演算法一與演算法二對於 5 層堆疊在勝利棋譜數超過 1000 筆時勝率皆超過 5 成，不過演算法三對於勝利的提升有很明顯的效果，當勝利棋譜數 10000 筆時的勝率以達到接近 6 成，因此藉由我們的演算法三，確實讓電腦變得比較聰明一些。

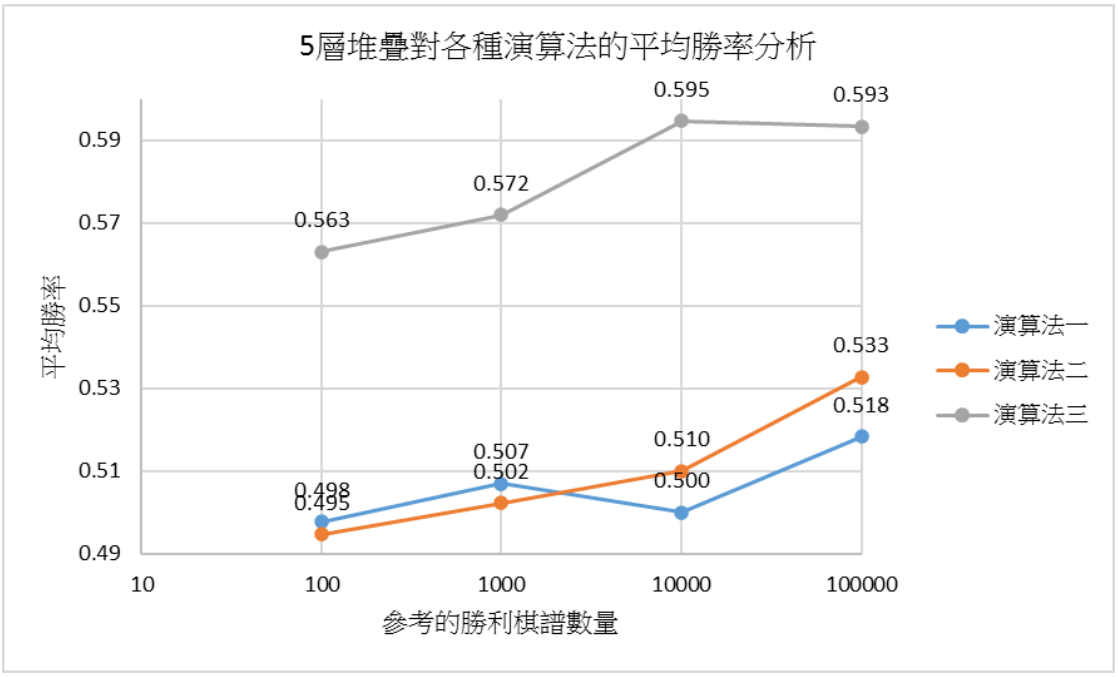

圖 11

由下圖 12 可以發現，三種演算法對於較少的 4 層堆疊時，勝率皆有很明顯的提升，三種演算法在勝利棋譜數 100000 筆時平均勝率高達接近 7 成。

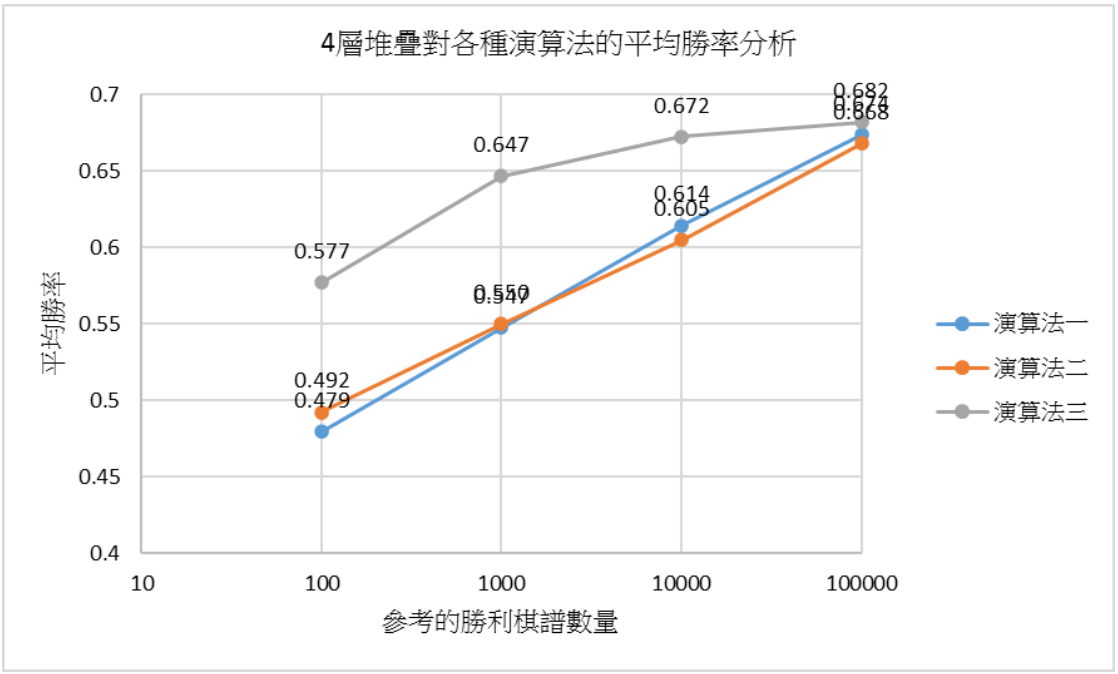

圖 12

由下圖 13 可以發現，由於 3 層堆疊本來圈圈數量就比較少的關係，採用演算法一開始對於勝率的提升就會有明顯效果，不過當勝利棋譜數 1000 筆時，之後的提升則會趨於穩定，因此如果只玩 3 層堆疊的話，所使用的勝利棋譜數則不用採用太多。

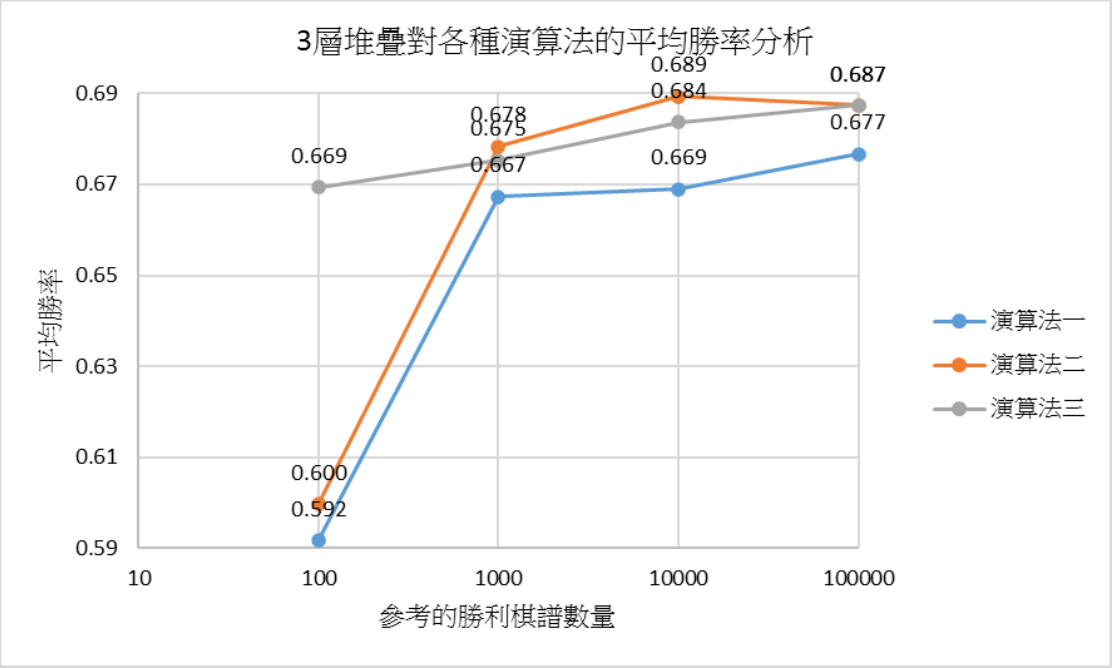

圖 13

四、人機互動畫面顯示

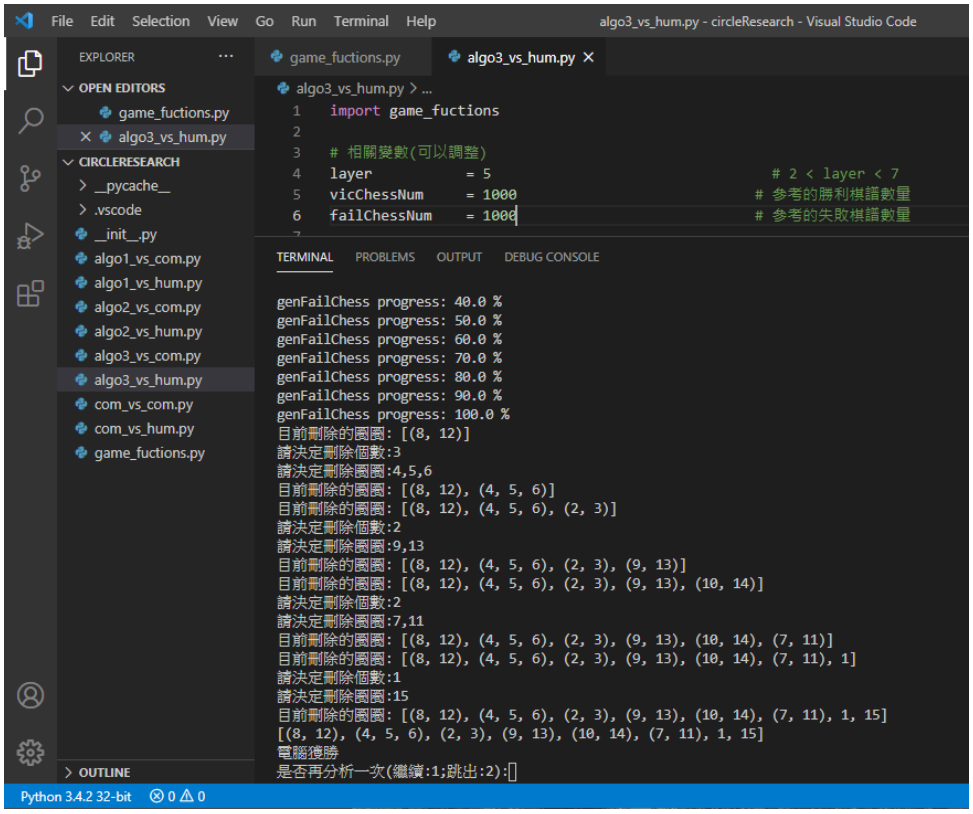

圖 14

柒、結論

一、由測試結果可知，我們開發的三種演算法針對各種堆疊數，確實可以提升勝率，成功達到讓電腦變得更加聰明與智慧的目的。

二、以主流的 5 層堆疊來說，我們的演算法三在當使用的勝利與失敗棋譜 10000 筆時，對決隨機策略時的勝率高達 6 成左右。

三、確實利用 Python 程式語言撰寫出人機互動遊戲。

捌、參考文獻資料

一、畫過時代，圈在一起。彰化縣 106 年第 57 屆中小學科學展覽會

二、均一教育平台：Python3 主題創作坊

https://www.junyiacademy.org/computing/programming/python/python-b01

三、Python Tutorial- W3Schools

https://www.w3schools.com/python/

玖、附錄

game_fuctions.py

```
import random
from random import choice

################################
###   產生可以刪除的各種可能   ###
################################
def genChoice(layer):

    if      layer == 3:
            oneChoice   = [ 1, 2, 3, 4, 5, 6]
            twoChoice   = [ (2,3), (4,5), (5,6), (1,2), (2,4), (3,5), (1,3), (3,6), (2,5)]
            thrChoice   = [ (4,5,6), (1,2,4), (1,3,6)]
    elif    layer == 4:
            oneChoice   = [ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
            twoChoice   = [ (2,3), (4,5), (5,6), (7,8), (8,9), (9,10),
                            (1,2), (2,4), (4,7), (3,5), (5,8), (6,9),
                            (1,3), (3,6), (6,10), (2,5), (5,9), (4,8)]
            thrChoice   = [ (4,5,6), (7,8,9), (8,9,10),
                            (3,5,8), (1,2,4), (2,4,7),
                            (2,5,9), (1,3,6), (3,6,10)]
    elif    layer == 5:
            oneChoice   = [ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]
            twoChoice   = [ (2,3), (4,5), (5,6), (7,8), (8,9), (9,10), (11,12), (12,13), (13,14), (14,15),
                            (1,2), (2,4), (4,7), (7,11), (3,5), (5,8), (8,12), (6,9), (9,13), (10,14),
                            (1,3), (3,6), (6,10), (10,15), (2,5), (5,9), (9,14), (4,8), (8,13), (7,12)]
            thrChoice   = [ (4,5,6), (7,8,9), (8,9,10), (11,12,13), (12,13,14), (13,14,15),
                            (6,9,13), (3,5,8), (5,8,12), (1,2,4), (2,4,7), (4,7,11),
                            (4,8,13), (2,5,9), (5,9,14), (1,3,6), (3,6,10), (6,10,15)]
    elif    layer == 6:
            oneChoice   = [ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21]
            twoChoice   = [ (2,3), (4,5), (5,6), (7,8), (8,9), (9,10), (11,12), (12,13),
                            (13,14), (14,15), (16,17), (17,18), (18,19), (19,20), (20,21),
                            (1,2), (2,4), (4,7), (7,11), (11,16), (3,5), (5,8), (8,12), (12,17),
                            (6,9), (9,13), (13,18), (10,14), (14,19), (15,20),
                            (1,3), (3,6), (6,10), (10,15), (15,21), (2,5), (5,9), (9,14), (14,20),
                            (4,8), (8,13), (13,19), (7,12), (12,18), (11,17)]
            thrChoice   = [ (4,5,6), (7,8,9), (8,9,10), (11,12,13), (12,13,14), (13,14,15),
                            (16,17,18), (17,18,19), (18,19,20), (19,20,21),
                            (10,14,19), (6,9,13), (9,13,18), (3,5,8), (5,8,12), (8,12,17),
                            (1,2,4), (2,4,7), (4,7,11), (7,11,16),
                            (7,12,18), (4,8,13), (8,13,19), (2,5,9), (5,9,14), (9,14,20),
                            (1,3,6), (3,6,10), (6,10,15), (10,15,21)]
    else:   print("Can not analyze lower than 3 layers and larger than 6 layers !")

    return [oneChoice, twoChoice, thrChoice]
```

```
################################
###   電腦決定要刪除的圈圈     ###
################################
def computerRound(choiceListNow):

    # 決定刪除圈圈個數
    if      len(choiceListNow[0]) != 0 and len(choiceListNow[1]) != 0 and len(choiceListNow[2]) != 0 :
            deleteNum = random.randrange(1,4)
    elif    len(choiceListNow[0]) != 0 and len(choiceListNow[1]) != 0 :
            deleteNum = random.randrange(1,3)
    else :  deleteNum = 1

    # 決定刪除圈圈號碼
    if      deleteNum == 1  : readyToDeleteSet  = choice(choiceListNow[0])
    elif    deleteNum == 2  : readyToDeleteSet  = choice(choiceListNow[1])
    elif    deleteNum == 3  : readyToDeleteSet  = choice(choiceListNow[2])

    return deleteNum, readyToDeleteSet
```

```
################################
###   人類決定要刪除的圈圈     ###
################################
def humanRound(choiceListNow):

    delete_str              = input("請決定刪除個數:")
    deleteNum               = int(delete_str)

    if deleteNum == 1:
        readyToDelete_str   = input("請決定刪除圈圈:")
        readyToDeleteSet    = int(readyToDelete_str)
    else:
        readyToDelete_str   = input("請決定刪除圈圈:")
        readyToDeleteSet    = tuple(map(int, readyToDelete_str.split(',')))

    return deleteNum, readyToDeleteSet
```

```
################################
###   抽出被刪除圈圈的號碼     ###
################################
def choiceExtract(deleteNum, readyToDeleteSet, choiceListNow):

    oneChoiceNow    = choiceListNow[0]
    twoChoiceNow    = choiceListNow[1]
    thrChoiceNow    = choiceListNow[2]
    diffCircleSet   = []

    for i in range(0,deleteNum):
        if deleteNum > 1   : readyToDelete = readyToDeleteSet[i]
        else               : readyToDelete = readyToDeleteSet

        # 整理 oneChoiceNow
        for j in range(0,len(oneChoiceNow)):
            if oneChoiceNow[j] == readyToDelete:
                diffCircleSet.append(oneChoiceNow[j])
        oneChoiceNow    = list(set(oneChoiceNow) - set(diffCircleSet))
        diffCircleSet   = []

        # 整理 twoChoiceNow
        for j in range(0,len(twoChoiceNow)):
            for k in range(0,2):
                if twoChoiceNow[j][k] == readyToDelete:
                    diffCircleSet.append(twoChoiceNow[j])
                    break
        twoChoiceNow    = list(set(twoChoiceNow) - set(diffCircleSet))
        diffCircleSet   = []

        # 整理 thrChoiceNow
        for j in range(0,len(thrChoiceNow)):
            for k in range(0,3):
                if thrChoiceNow[j][k] == readyToDelete:
                    diffCircleSet.append(thrChoiceNow[j])
                    break
        thrChoiceNow    = list(set(thrChoiceNow) - set(diffCircleSet))
        diffCircleSet   = []

    return [oneChoiceNow, twoChoiceNow, thrChoiceNow]
```

```
################################
###   電腦 VS 電腦 (一回)     ###
################################
def com_vs_com(layer):

    totalCircle     = 0                 # 圈圈總數
    deleteNum       = 0                 # 當次回合要刪除的圈圈個數
    deleteTotal     = 0                 # 目前總共刪除的個數
    deleteOrder     = []                # 用來儲存刪除的過程
    choiceListNow   = genChoice(layer)  # 產生可以刪除的各種可能

    # 計算圈圈總數
    for i in range(1, layer+1): totalCircle += i

    # 開始遊戲(電腦自行對弈)
    while deleteTotal < totalCircle:
        # 電腦決定刪除圈圈號碼
        deleteNum, readyToDeleteSet = computerRound(choiceListNow)
        # 將要刪除的圈圈記錄下來
        deleteOrder.append(readyToDeleteSet)
        # 抽出刪除圈圈的號碼
        choiceListNow = choiceExtract(deleteNum, readyToDeleteSet, choiceListNow)
        # 計算目前刪除的圈圈數
        deleteTotal += deleteNum
        # 刪完圈圈後停止
        if deleteTotal == totalCircle: break

    return deleteOrder
```

```
################################
###       產生勝利棋譜        ###
################################
def genVicChess(vicChessNum, layer):

    vicChessNow     = 0     # 目前儲存的勝利棋譜數量
    vicChessList    = []    # 勝利棋譜大集合

    while vicChessNow < vicChessNum:

        # 執行一回合遊戲
        deleteOrder = com_vs_com(layer)

        # 儲存勝利棋譜至 vicChessList
        if len(deleteOrder) % 2 == 0:
            vicChessList.append(deleteOrder)
            vicChessNow += 1

            # 進度顯示
            if vicChessNow % (vicChessNum / 10) == 0:
                print("genVicChess progress: " + str(100.0 * vicChessNow/vicChessNum) + " %")

        if vicChessNow == vicChessNum: break

    return vicChessList
```

```
################################
###        抽出勝利棋譜       ###
################################
def vicChessExtract(vicChessList, deleteOrder):

    vicChessListNew = []

    for i in range(0, len(vicChessList)):
        for j in range(len(deleteOrder)-1, len(deleteOrder)): # 只檢查 deleteOrder 最後一個

            if type(vicChessList[i][j]) is int and type(deleteOrder[j]) is int  and vicChessList[i][j] == del
eteOrder[j]:
                vicChessListNew.append(vicChessList[i])

            elif type(vicChessList[i][j]) is tuple and type(deleteOrder[j]) is tuple and vicChessList[i][j] =
= deleteOrder[j]:
                vicChessListNew.append(vicChessList[i])

    return vicChessListNew
```

```
################################
###       產生失敗棋譜        ###
################################
def genFailChess(failChessNum, layer):

    failChessNow     = 0     # 目前儲存的失敗棋譜數量
    failChessList    = []    # 失敗棋譜大集合

    while failChessNow < failChessNum:

        # 執行一回合遊戲
        deleteOrder = com_vs_com(layer)

        # 儲存失敗棋譜至 failChessList
        if len(deleteOrder) % 2 == 1:
            failChessList.append(deleteOrder)
            failChessNow += 1

            # 進度顯示
            if failChessNow % (failChessNum / 10) == 0:
                print("genFailChess progress: " + str(100.0 * failChessNow/failChessNum) + " %")

        if failChessNow == failChessNum: break

    return failChessList
```

```
################################
###        抽出失敗棋譜       ###
################################
def failChessExtract(failChessList, deleteOrder):

    failChessListNew = []

    for i in range(0, len(failChessList)):
        for j in range(len(deleteOrder)-1, len(deleteOrder)): # 只檢查 deleteOrder 最後一個

            if type(failChessList[i][j]) is int and type(deleteOrder[j]) is int and failChessList[i][j] == de
leteOrder[j]:
                failChessListNew.append(failChessList[i])

            elif type(failChessList[i][j]) is tuple and type(deleteOrder[j]) is tuple and failChessList[i][j]
 == deleteOrder[j]:
                failChessListNew.append(failChessList[i])

    return failChessListNew
```

```
################################
###  策略 1: 根據勝利棋譜刪除   ###
################################
def strategy_vicChess(vicChessList, deleteOrder):

    vicChessUsedNum     = random.randrange(0,len(vicChessList))
    readyToDeleteSet    = vicChessList[vicChessUsedNum][len(deleteOrder)]

    if type(readyToDeleteSet) is int    : deleteNum = 1
    else                                : deleteNum = len(readyToDeleteSet)

    return deleteNum, readyToDeleteSet
```

```
################################
###  策略 2: 根據失敗棋譜刪除   ###
################################
def strategy_failChess(failChessList, deleteOrder, choiceListNow):

    failChoiceList      = [[],[],[]]    # 用來儲存失敗棋譜的選擇
    choiceList_temp     = [[],[],[]]    # 刪除失敗棋譜的選擇後所剩的選擇

    # 挑選出失敗棋譜的選擇
    for i in range(0, len(failChessList)):
        if type(failChessList[i][len(deleteOrder)]) is int:
            failChoiceList[0].append(failChessList[i][len(deleteOrder)])
        elif type(failChessList[i][len(deleteOrder)]) is tuple and len(failChessList[i][len(deleteOrder)]) ==
 2:
            failChoiceList[1].append(failChessList[i][len(deleteOrder)])
        elif type(failChessList[i][len(deleteOrder)]) is tuple and len(failChessList[i][len(deleteOrder)]) ==
 3:
            failChoiceList[2].append(failChessList[i][len(deleteOrder)])

    # 把會失敗的選擇刪除
    choiceList_temp[0] = list(set(choiceListNow[0]) - set(failChoiceList[0]))
    choiceList_temp[1] = list(set(choiceListNow[1]) - set(failChoiceList[1]))
    choiceList_temp[2] = list(set(choiceListNow[2]) - set(failChoiceList[2]))

    # 電腦避開失敗棋譜，如果沒有一個圈圈的選擇可以刪除，則隨機刪除
    if len(choiceList_temp[0]) != 0 : deleteNum, readyToDeleteSet = computerRound(choiceList_temp)
    else                            : deleteNum, readyToDeleteSet = computerRound(choiceListNow)

    return deleteNum, readyToDeleteSet
```

```
################################
###  策略 3: 奇數原則          ###
################################
def strategy_oddNumber(deleteOrder, choiceListNow, totalCircle):

    deleteNumNow = 0  # 目前已刪除的圈圈個數

    # 先計算目前已刪除的圈圈個數
    for i in range(0, len(deleteOrder)):
        if type(deleteOrder[i]) is int  : deleteNumNow += 1
        else                            : deleteNumNow += len(deleteOrder[i])

    # 若剩下的圈圈有奇數個，則刪除偶數個圈圈(即刪除 2 個)
    if (totalCircle - deleteNumNow) % 2 == 1:
        if len(choiceListNow[1]) != 0:
            deleteNum = 2
            readyToDeleteSet = choice(choiceListNow[1])
        else:
            deleteNum, readyToDeleteSet = computerRound(choiceListNow)

    # 若剩下的圈圈有偶數個，則刪除奇數個圈圈(即刪除 1 或 3 個)
    else:
        if len(choiceListNow[0]) != 0 and len(choiceListNow[2]) != 0:

            # 決定要刪除 1 個圈圈還是 3 個圈圈
            if random.randrange(1,3) == 1: # 選到 1 就消 1 個圈圈,選到 2 就消 3 個圈圈
                deleteNum = 1
                readyToDeleteSet = choice(choiceListNow[0])
            else:
                deleteNum = 3
                readyToDeleteSet = choice(choiceListNow[2])
        else:
            deleteNum, readyToDeleteSet = computerRound(choiceListNow)

    return deleteNum, readyToDeleteSet
```

```
################################
###  演算法 1: 策略 1           ###
################################
def algo1_Round(choiceListNow, vicChessList, deleteOrder):

    # 若有勝利棋譜則參考勝利棋譜刪除(策略 1)
    if len(vicChessList) != 0:
        deleteNum, readyToDeleteSet = strategy_vicChess(vicChessList, deleteOrder)

    # 沒有勝利棋譜可以參考則隨機刪除
    else:
        deleteNum, readyToDeleteSet = computerRound(choiceListNow)

    return deleteNum, readyToDeleteSet
```

```
################################
### 演算法 2: 策略 1 + 2        ###
################################
def algo2_Round(choiceListNow, vicChessList, failChessList, deleteOrder):

    # 若有勝利棋譜則參考勝利棋譜刪除(策略 1)
    if len(vicChessList) != 0:
        deleteNum, readyToDeleteSet = strategy_vicChess(vicChessList, deleteOrder)

    # 若沒有勝利棋譜，則參考失敗棋譜(策略 2)
    elif len(vicChessList) == 0 and len(failChessList) != 0:
        deleteNum, readyToDeleteSet = strategy_failChess(failChessList, deleteOrder, choiceListNow)

    # 沒有勝利以及失敗棋譜可以參考
    else:
        deleteNum, readyToDeleteSet = computerRound(choiceListNow)

    return deleteNum, readyToDeleteSet
```

```
################################
### 演算法 3: 策略 1 + 2 + 3    ###
################################
def algo3_Round(choiceListNow, vicChessList, failChessList, deleteOrder, totalCircle):

    # 若有勝利棋譜則參考勝利棋譜刪除(策略 1)
    if len(vicChessList) != 0:
        deleteNum, readyToDeleteSet = strategy_vicChess(vicChessList, deleteOrder)

    # 若沒有勝利棋譜，則參考失敗棋譜(策略 2)
    elif len(vicChessList) == 0 and len(failChessList) != 0:
        deleteNum, readyToDeleteSet = strategy_failChess(failChessList, deleteOrder, choiceListNow)

    # 若沒有勝利棋譜、失敗棋譜可以參考，則遵循奇數原則(策略 3)
    else:
        deleteNum, readyToDeleteSet = strategy_oddNumber(deleteOrder, choiceListNow, totalCircle)

    return deleteNum, readyToDeleteSet
```

```
import game_fuctions

################################
###   電腦 VS 人類            ###
################################
# 相關變數(可以調整)
layer       = 5             # 2 < layer < 7 ,除非 genChoice()中有建置 7 層以上的刪除選擇
gameNum     = 1000          # 遊戲次數
# 相關變數(不要調整)
vicNum      = 0             # 儲存電腦獲勝次數
# 只進行一個回合
if gameNum == 1:

    deleteOrder = game_fuctions.com_vs_com(layer)

    print(deleteOrder)
    if len(deleteOrder) % 2 == 0    : print("先手獲勝")
    else                            : print("後手獲勝")

# 要進行多個回合
else:
    for i_game in range(0, gameNum):

        # 進度顯示
        if i_game % (gameNum / 10) == 0:
            print("com_vs_com: " + str(100.0 * i_game/gameNum) + " %")

        deleteOrder = game_fuctions.com_vs_com(layer)

        if len(deleteOrder) % 2 == 0 : vicNum += 1

    print("先手獲勝次數: " + str(vicNum))
    print("先手獲勝勝率: " + str(float(vicNum)/gameNum))
```

algo1_vs_com.py

```
import game_fuctions

# 相關變數(可以調整)
layer           = 5         # 2 < layer < 7
gameNum         = 1000      # 遊戲次數
vicChessNum     = 100      # 參考的勝利棋譜數量
# 相關變數(不要調整)
vicNum          = 0         # AI 獲勝次數
vicChessList    = []        # 勝利棋譜
totalCircle     = 0         # 圈圈總數
continue_flag   = 1         # 1:繼續; 其它數字:不繼續
# 計算圈圈總數
for i in range(1, layer+1): totalCircle += i

# 產生勝利棋譜(需要較長的時間)
vicChessList    = game_fuctions.genVicChess(vicChessNum, layer)

for i_test in range(0,3):
    for i_game in range(0, gameNum):
        # 變數更新
        deleteNum           = 0                                 # 當次回合要刪除的圈圈個數
        deleteTotal         = 0                                 # 目前總共刪除的個數
        deleteOrder         = []                                # 用來儲存刪除的過程
        choiceListNow       = game_fuctions.genChoice(layer)    # 產生可以刪除的各種可能
        vicChessList_temp   = vicChessList.copy()

        # 進度顯示
        if i_game % (gameNum / 10) == 0:
            print("algorithm1_vs_com: " + str(100.0 * i_game/gameNum) + " %")

        # 開始遊戲(電腦自行對弈)
        for i in range(0, totalCircle):
            # 電腦決定刪除圈圈號碼
            if  i % 2 == 0  :  deleteNum, readyToDeleteSet  = game_fuctions.algo1_Round(choiceListNow, vicChessList_temp, deleteOrder)
            else            :  deleteNum, readyToDeleteSet  = game_fuctions.computerRound(choiceListNow)
            # 將要刪除的圈圈記錄下來
            deleteOrder.append(readyToDeleteSet)
            # 抽出勝利棋譜
            vicChessList_temp = game_fuctions.vicChessExtract(vicChessList_temp, deleteOrder)
            # 抽出刪除圈圈的號碼
            choiceListNow = game_fuctions.choiceExtract(deleteNum, readyToDeleteSet, choiceListNow)
            # 計算目前刪除的圈圈數
            deleteTotal += deleteNum
            # 刪完 15 個圈圈後停止
            if deleteTotal == totalCircle: break

        if len(deleteOrder) % 2 == 0: vicNum += 1
    print("AI 獲勝次數: " + str(vicNum))
    print("AI 獲勝勝率: " + str(float(vicNum)/gameNum))

    vicNum          = 0
```

algo2_vs_com.py

```
import game_fuctions

# 相關變數(可以調整)
layer           = 3         # 2 < layer < 7
gameNum         = 1000      # 遊戲次數
vicChessNum     = 100000      # 參考的勝利棋譜數量
failChessNum    = 100000      # 參考的失敗棋譜數量
# 相關變數(不要調整)
vicNum          = 0         # AI 獲勝次數
vicChessList    = []        # 勝利棋譜
failChessList   = []        # 失敗棋譜
totalCircle     = 0         # 圈圈總數
continue_flag   = 1         # 1:繼續; 其它數字:不繼續
# 計算圈圈總數
for i in range(1, layer+1): totalCircle += i
# 產生勝利及失敗棋譜(需要較長的時間)
vicChessList    = game_fuctions.genVicChess(vicChessNum, layer)
failChessList   = game_fuctions.genFailChess(failChessNum, layer)

for i_test in range(0,3):
    for i_game in range(0, gameNum):
        # 變數更新
        deleteNum           = 0                                 # 當次回合要刪除的圈圈個數
        deleteTotal         = 0                                 # 目前總共刪除的個數
        deleteOrder         = []                                # 用來儲存刪除的過程
        choiceListNow       = game_fuctions.genChoice(layer)    # 產生可以刪除的各種可能
        vicChessList_temp   = vicChessList.copy()
        failChessList_temp  = failChessList.copy()
        # 進度顯示
        if i_game % (gameNum / 10) == 0:
            print("algorithm2_vs_com: " + str(100.0 * i_game/gameNum) + " %")
        # 開始遊戲(電腦自行對弈)
        for i in range(0, totalCircle):
            # 電腦決定刪除圈圈號碼
            if  i % 2 == 0 :
                deleteNum, readyToDeleteSet  = game_fuctions.algo2_Round(choiceListNow, vicChessList_temp, fa
ilChessList_temp, deleteOrder)
            else:
                deleteNum, readyToDeleteSet  = game_fuctions.computerRound(choiceListNow)
            # 將要刪除的圈圈記錄下來
            deleteOrder.append(readyToDeleteSet)
            # 抽出勝利棋譜
            vicChessList_temp = game_fuctions.vicChessExtract(vicChessList_temp, deleteOrder)
            # 抽出失敗棋譜
            failChessList_temp = game_fuctions.failChessExtract(failChessList_temp, deleteOrder)
            # 抽出刪除圈圈的號碼
            choiceListNow = game_fuctions.choiceExtract(deleteNum, readyToDeleteSet, choiceListNow)
            # 計算目前刪除的圈圈數
            deleteTotal += deleteNum
            # 刪完 15 個圈圈後停止
            if deleteTotal == totalCircle: break
        if len(deleteOrder) % 2 == 0: vicNum += 1

    print("AI 獲勝次數: " + str(vicNum))
    print("AI 獲勝勝率: " + str(float(vicNum)/gameNum))
    vicNum          = 0
```

algo3_vs_com.py

```
import game_fuctions

# 相關變數(可以調整)
layer           = 5         # 2 < layer < 7
gameNum         = 1000      # 遊戲次數
vicChessNum     = 1000000   # 參考的勝利棋譜數量
failChessNum    = 1000000   # 參考的失敗棋譜數量
# 相關變數(不要調整)
vicNum          = 0         # AI 獲勝次數
vicChessList    = []        # 勝利棋譜
failChessList   = []        # 失敗棋譜
totalCircle     = 0         # 圈圈總數
continue_flag   = 1         # 1:繼續; 其它數字:不繼續

# 計算圈圈總數
for i in range(1, layer+1): totalCircle += i

# 產生勝利及失敗棋譜(需要較長的時間)
vicChessList    = game_fuctions.genVicChess(vicChessNum, layer)
failChessList   = game_fuctions.genFailChess(failChessNum, layer)

for i_test in range(0,3):
    for i_game in range(0, gameNum):
        # 變數更新
        deleteNum           = 0                                 # 當次回合要刪除的圈圈個數
        deleteTotal         = 0                                 # 目前總共刪除的個數
        deleteOrder         = []                                # 用來儲存刪除的過程
        choiceListNow       = game_fuctions.genChoice(layer)    # 產生可以刪除的各種可能
        vicChessList_temp   = vicChessList.copy()
        failChessList_temp  = failChessList.copy()
        # 進度顯示
        if i_game % (gameNum / 10) == 0:
            print("algorithm3_vs_com: " + str(100.0 * i_game/gameNum) + " %")
        # 開始遊戲(電腦自行對弈)
        for i in range(0, totalCircle):
            # 電腦決定刪除圈圈號碼
            if  i % 2 == 0 :
                deleteNum, readyToDeleteSet  = game_fuctions.algo3_Round(choiceListNow, vicChessList_temp, fa
ilChessList_temp, deleteOrder, totalCircle)
            else:
                deleteNum, readyToDeleteSet  = game_fuctions.computerRound(choiceListNow)
            # 將要刪除的圈圈記錄下來
            deleteOrder.append(readyToDeleteSet)
            # 抽出勝利棋譜
            vicChessList_temp = game_fuctions.vicChessExtract(vicChessList_temp, deleteOrder)
            # 抽出失敗棋譜
            failChessList_temp = game_fuctions.failChessExtract(failChessList_temp, deleteOrder)
            # 抽出刪除圈圈的號碼
            choiceListNow = game_fuctions.choiceExtract(deleteNum, readyToDeleteSet, choiceListNow)
            # 計算目前刪除的圈圈數
            deleteTotal += deleteNum
            # 刪完 15 個圈圈後停止
            if deleteTotal == totalCircle: break
        if len(deleteOrder) % 2 == 0: vicNum += 1
    print("AI 獲勝次數: " + str(vicNum))
    print("AI 獲勝勝率: " + str(float(vicNum)/gameNum))
    vicNum          = 0
```

algo3_vs_hum.py

```
import game_fuctions

# 相關變數(可以調整)
layer           = 5                                 # 2 < layer < 7
vicChessNum     = 100000                            # 參考的勝利棋譜數量
failChessNum    = 100000                            # 參考的失敗棋譜數量
# 相關變數(不要調整)
vicChessList    = []                                # 勝利棋譜
failChessList   = []                                # 失敗棋譜
totalCircle     = 0                                 # 圈圈總數
deleteNum       = 0                                 # 當次回合要刪除的圈圈個數
deleteTotal     = 0                                 # 目前總共刪除的個數
deleteOrder     = []                                # 用來儲存刪除的過程
choiceListNow   = game_fuctions.genChoice(layer)    # 產生可以刪除的各種可能
continue_flag   = 1                                 # 1:繼續; 其它數字:不繼續

# 計算圈圈總數
for i in range(1, layer+1): totalCircle += i

# 產生勝利及失敗棋譜(需要較長的時間)
vicChessList    = game_fuctions.genVicChess(vicChessNum, layer)
failChessList   = game_fuctions.genFailChess(failChessNum, layer)

while continue_flag == 1:
    # 變數更新
    deleteNum           = 0                                 # 當次回合要刪除的圈圈個數
    deleteTotal         = 0                                 # 目前總共刪除的個數
    deleteOrder         = []                                # 用來儲存刪除的過程
    choiceListNow       = game_fuctions.genChoice(layer)    # 產生可以刪除的各種可能
    vicChessList_temp   = vicChessList.copy()
    failChessList_temp  = failChessList.copy()
    # 開始遊戲(電腦人類對弈)
    for i in range(0, totalCircle):
        # 電腦決定刪除圈圈號碼
        if  i % 2 == 0  :  deleteNum, readyToDeleteSet  = game_fuctions.algo3_Round(choiceListNow, vicChessLi
st_temp, failChessList, deleteOrder, totalCircle)
        else            :  deleteNum, readyToDeleteSet  = game_fuctions.humanRound(choiceListNow)
        # 將要刪除的圈圈記錄下來
        deleteOrder.append(readyToDeleteSet)
        print("目前刪除的圈圈: " + str(deleteOrder))
        # 抽出勝利棋譜
        vicChessList_temp = game_fuctions.vicChessExtract(vicChessList_temp, deleteOrder)
        # 抽出失敗棋譜
        failChessList_temp = game_fuctions.failChessExtract(failChessList_temp, deleteOrder)
        # 抽出刪除圈圈的號碼
        choiceListNow = game_fuctions.choiceExtract(deleteNum, readyToDeleteSet, choiceListNow)
        # 計算目前刪除的圈圈數
        deleteTotal += deleteNum
        # 刪完 15 個圈圈後停止
        if deleteTotal == totalCircle: break

    print(deleteOrder)
    if len(deleteOrder) % 2 == 0    : print("電腦獲勝")
    else                            : print("人類獲勝")

    continue_str    = input("是否再分析一次(繼續:1;跳出:2):")
    continue_flag   = int(continue_str)
```