2023 年臺灣國際科學展覽會
優勝作品專輯

作品編號　190017

參展科別　電腦科學與資訊工程

作品名稱　利用增強學習之 Q-Learning，解決數字華容道的比較性發展研究

得獎獎項

就讀學校　新北市立錦和高級中學

新北市立新北高級工業職業學校

指導教師　李長儒、林欽鴻

作者姓名　孫偉峻、陳昱臻、高守之

關鍵詞　AI 增強學習、華容道、Q-Learning

作者簡介

大家好！我們是孫偉峻、陳昱臻與高守之。目前是高一。在國中那三年，我們透過互相討論以及自行探索，成功利用”Q-Learning”製作”數字華榮道”AI 程式，能如此順利成功絕對是倚仗大家的幫助。在當時，大家都是在放學或者假日時間拉出時間空檔，日復一日的嘗試才得以完成。在實作書面報告的部分也是大家一起努力而來。最後，我們要感謝陳懷德主任以及李長儒老師這些年的支持與陪伴，讓我們得以成功的做出這一件作品。

摘要

因為我們一開始對電腦程式語言有濃厚的興趣，所以去學習了 python 程式語言，後來發現到世界三大益智的華容道遊戲，似乎可以加以運用，又從文獻中發現了人工智慧之重要性和增強學習的各類法則。剛好於國中時期寫出了讓電腦產生並解決 3*3 數字華容道之程式。但發現 4*4 的遊戲竟有 20 兆種組合，該無法用 3*3 之程式思維。後來用了增強學習的 Q-Learning 技術，不僅完成任務，而且還可以發展出人與電腦的比賽，造成轟動、受到小朋友的喜愛~最後我們還希望自己能設計出不同的華容道加以測試，並研究深度增強學習（DRL）的原理與應用，來解決更高階的遊戲，達到增進人工智慧學習的發展。

Abstract

Because we had a strong interest in computer programming language at the beginning, we went to learn the python programming language. Later, we found that the Huarong Road game, one of the three major puzzles in the world, seems to be able to be used. We also discovered the importance of artificial intelligence from the literature. and various laws of reinforcement learning. Just in the middle school period, he wrote a program to let the computer generate and solve the 3*3 digital Huarong Road. However, it was found that there are 20 trillion combinations in a 4*4 game, and it is impossible to think with a 3*3 program. Later, the Q-Learning technology of reinforcement learning was used, which not only completed the task, but also developed a competition between people and computers, which caused a sensation and was loved by children. Finally, we also hope that we can design different Huarong Roads to test and test them. Research the principles and applications of Deep Reinforcement Learning (DRL) to solve higher-level games and enhance the development of artificial intelligence learning.

一、 前言

(一)研究動機

近年來，資訊技術越來越多，且日新月異，而 AI 技術發展越來越進步，且我們對於電腦程式的編程有著濃厚的興趣，故學習了程式語言 python。在學習程式語言中，曾聽到某教授說過「要讓電腦幫你做事」！這又使我們對電腦的應用產生了極大的熱忱，之後開始了日以繼夜的學習過程，從中我們希望不只得到知識，而是學了去實踐、去幫助人。然後發現滑塊拼圖這遊戲，其中的數字間藏著有趣的關係，因此想讓電腦解決有著多達 20 兆種排列組合的 4 X 4 滑塊拼圖，讓電腦判斷此排列是否有解還是無解，以及找到解出的時間與最佳的路徑，更希望能運用 AI 當中的增強學習技術，來解決滑塊拼圖之中的一些方法。最後希望除了幫助不會解滑塊拼圖的人，能夠自己解出來外，還可以學到更多的 AI 技術，來解其他不同的益智遊戲。

(二)研究目的

我們希望本言就可以做到三大指標，再藉各項指標之分支依次完成：

1. 利用文獻去探討並分析各種不同類型的華容道以及解決數字華容道的方式。
2. 分析並使用 BFS,Q-Learning 寫出 python 程式，讓電腦可以解決 3*3、4*4 的數字華容道。
3. 能夠以華容道為原點做更多相關的連結與延伸，諸如：利用雷射雕割機與 3D 列印機，創出新型數字華容道並思考其延伸應用、利用更多期刊文獻分析出可改進之 AI 方向。

二、 研究方法與過程

(一)文獻探討部分

1. 華容道樣式探討

世界三大益智遊戲有中國華容道、匈牙利魔術方塊、法國獨立鑽石棋。而我們主要研究的是與中國華容道有關的數字華容道，而華容道是一種滑塊類遊戲。

(1)調查不同類型的華容道

A. 數字容華道

數字華容道是由一塊有凹槽的板和數個寫有數字的大小相同的方塊所組成。4x4 華容道遊戲的板上會有十五個方塊和一個大小相當於一個方塊的空位（供方塊移動用）。而 3x3 華容道遊戲，是九宮格布局，共有八個方塊以及一個空位。遊戲規則是玩家要移動板上的方塊，讓全部的方塊順著數字次序排列，且不能用手直接交換方塊，最後評分規則是看復原華容道的時間或是最短步驟數解出來。

（圖 2-1）3x3 數字華容道

（圖 2-2）4x4 數字華容道

(2)中國傳統華容道

在一個方形盒子內放置了大小不同的方塊，一般為 4x5 布局的大小。在這些方塊中有特殊的一個（一般是最大的，且是 2x2 正方形）必須被移動到設計好的指定地點。玩家不允許拿起方塊，只能向平行或垂直的方向移動方塊，常見的玩法是移動最少次數，或者用最少的時間來完成遊戲。

華容道遊戲由來是取自著名的三國故事，曹操在赤壁之戰中被劉備和孫權的「苦肉計」、「鐵索連舟」打敗，被迫退逃到華容道，又遇上諸葛亮的伏兵，關羽為報答曹操對他的恩情，明逼實讓，最終幫助曹操逃出了華容道。遊戲就是依照「曹瞞兵敗走華容，正與關公狹路逢。只為當初恩義重，放開金鎖走蛟龍」這個故事情節，但此遊戲的起源，卻非一般人認為的是「中國最古老的遊戲之一」。實際上它的歷史可能很短。華容道的現在樣式是 1932 年 John Harold

Fleming 在英國申請的專利，並還附上橫刀立馬的解法。

玩法是幫助曹操從初始位置移到棋盤最下方中部，從出口逃走。不允許跨越棋子。而其中曹操逃出華容道的最大障礙是關羽，關羽立馬華容道，一夫當關，萬夫莫開。關羽與曹操當然是解開這一遊戲裡的關鍵。四個劉備軍兵是最靈活，也最容易對付，如何發揮他們的作用也要充分考慮周全。

（圖 2-3）中國傳統華容道

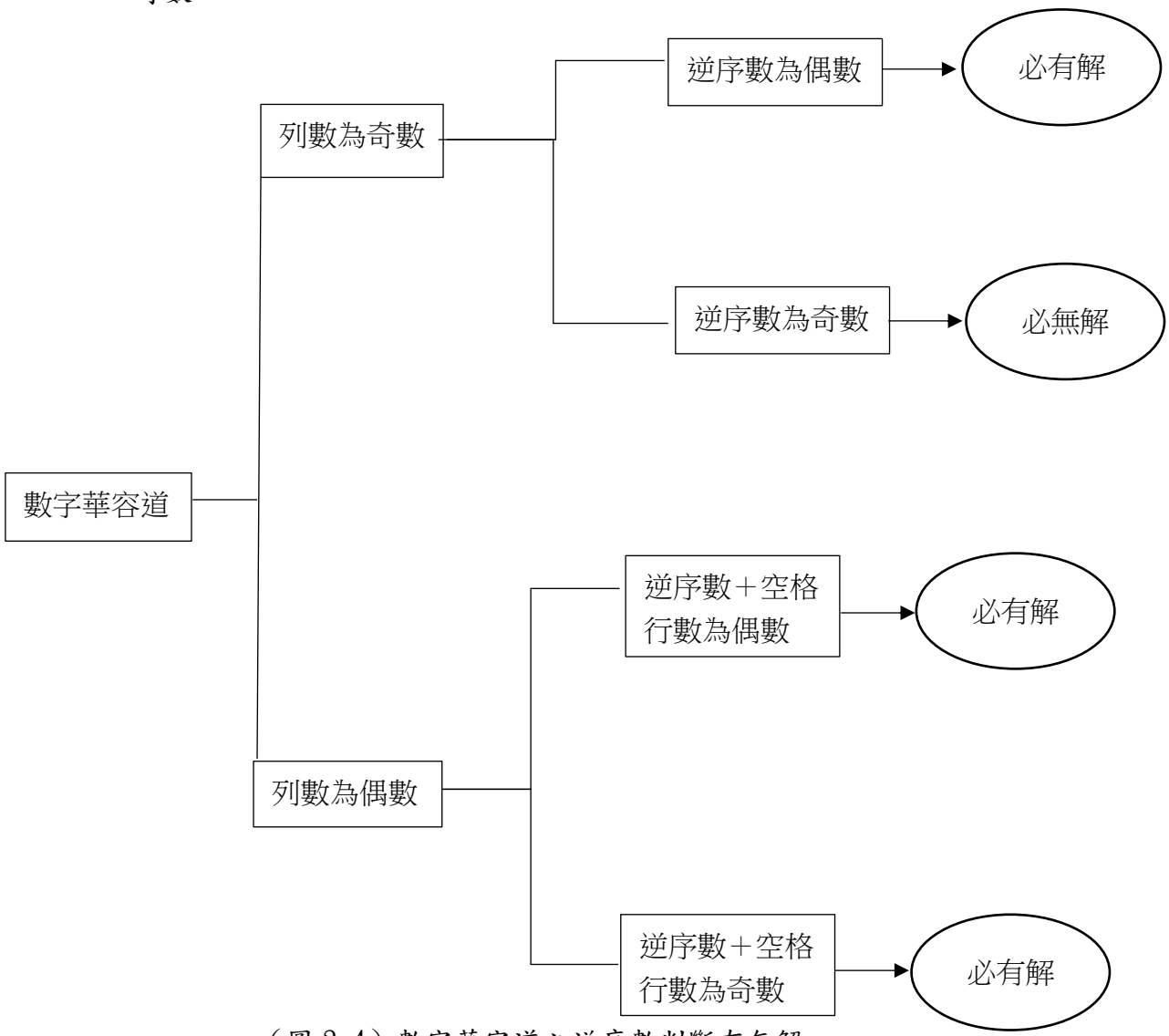
小結：最後我們選擇數字華容道作為研究主題。

2.復原數字華容道適合方法探討

(1)調查數字華容道的解決方式

A. 判斷有無解

數字華容道判斷是否有解與逆序對數有關。而排列必然有解有三種情況，如果格子列數為奇數，則逆序數必須為偶數；若格子列數為偶數，且逆序數為偶數，則當前空格所在行數與初始空格所在行數的差為偶數；若格子列數為偶數，且逆序數為奇數，則當前空格所在行數與初始空格所在行數的差為奇數。

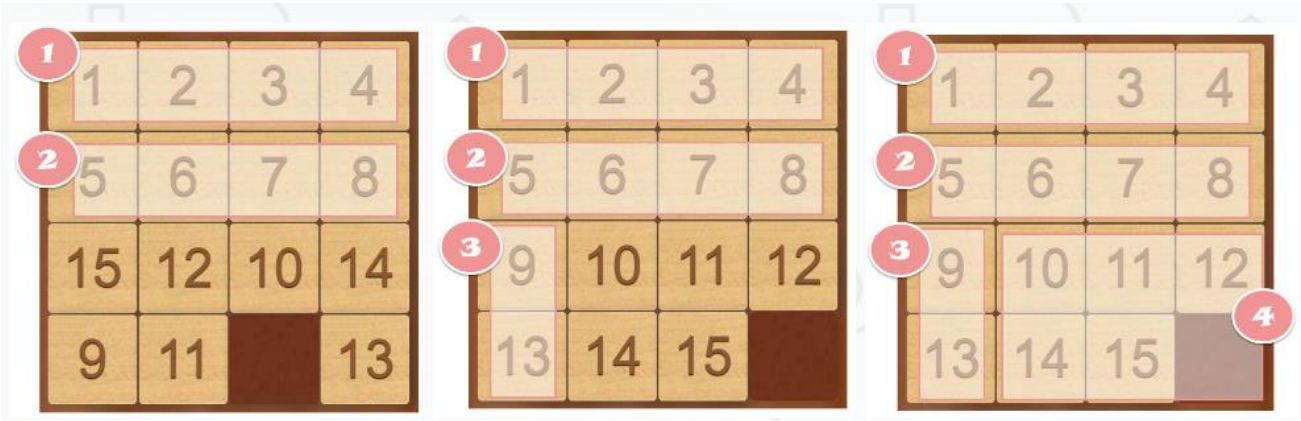


（圖 2-4）數字華容道之逆序數判斷有無解

B.解法

a.層解

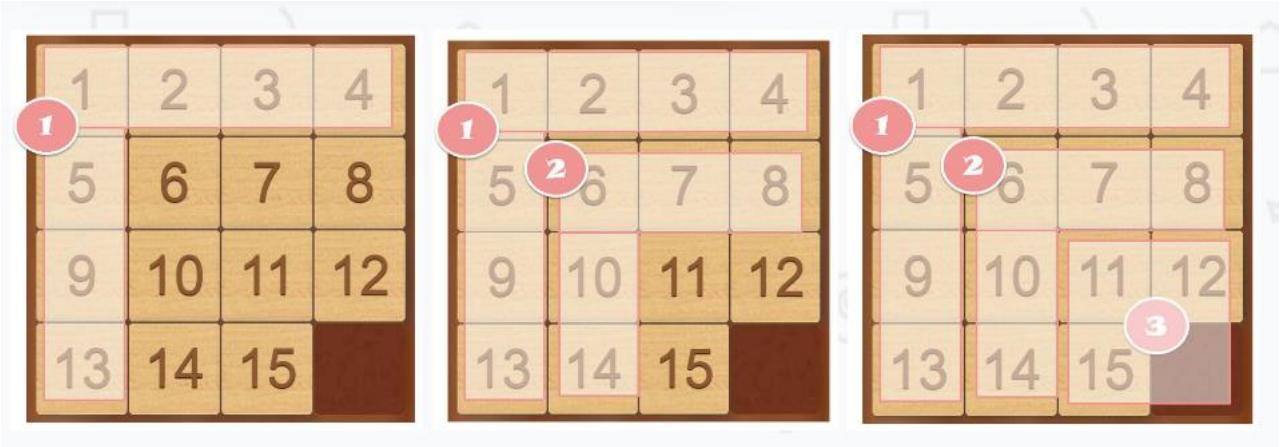
是一層接著一層復原，先完成第一列，再完成下一列，如果解到了最後兩列，再換成解最後兩列的數字，先解第一行數字，再解下一行的數字，最後解到剩下最後兩列的最後兩行的數字（3 個數字+1 空白），再用順時針或逆時針方向轉復原。



（圖 2-5）4x4 數字華容道之層解步驟

b.降階

是用一階接著一接完成，而解的每一階都像 L 字型，一開始先完成第一列與第一行，接著完成下一列與下一行，最後解到了最後兩列與最後兩行，再用順時針或逆時針方向復原完成。



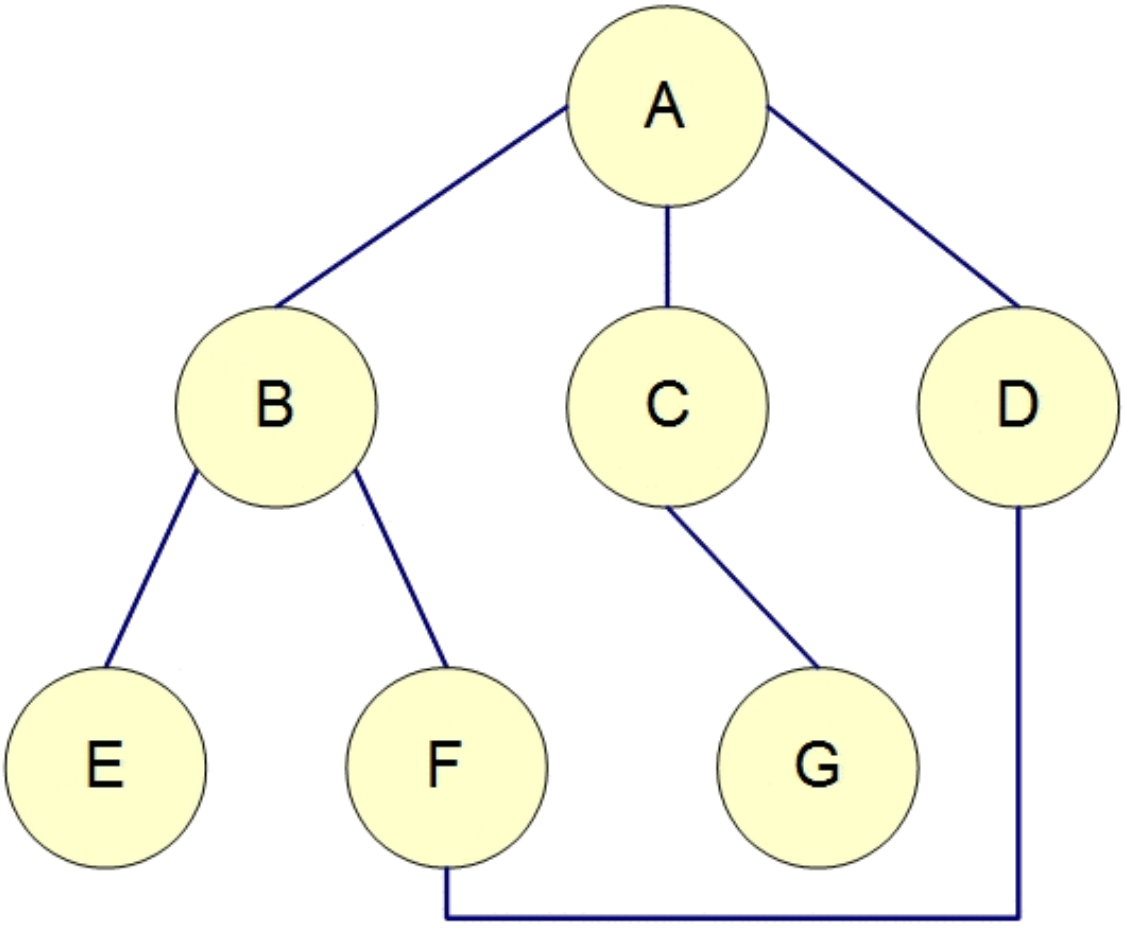
（圖 2-6）4x4 數字華容道之降階解法步驟

C.演算法

a. BFS 廣度優先搜尋

廣度優先搜尋演算法（英語：Breadth-First Search，縮寫為 BFS），又翻譯為廣度優先搜尋，或橫向優先搜尋，是一種圖形搜尋演算法。簡單的說，BFS 是從根節點開始，沿著樹的寬度遍歷樹的節點。如果所有節點都被存取，則演算法中止。廣度優先搜尋的實現一般採用 open-closed 表。BFS 是一種暴力搜尋演算法，目的是系統地展開並檢查圖中的所有節點，以找尋結果。換句話說，它並不考慮結果的可能位址，徹底地搜尋整張圖，直到找到結果為止。BFS 並不是使用經驗法則演算法。

從演算法的觀點，所有因為展開節點而得到的子節點，都會被加進一個先進先出的佇列中。在一般的實作裡，其鄰居節點尚未被檢驗過的節點會被放置在一個被稱為 open 的容器中（例如佇列或是連結串列），而被檢驗過的節點則被放置在被稱為 closed 的容器中。（open-closed 表）



（圖 2-7）BFS 的流程示例圖

b. AI 增強學習的 Q-Learning 演算法

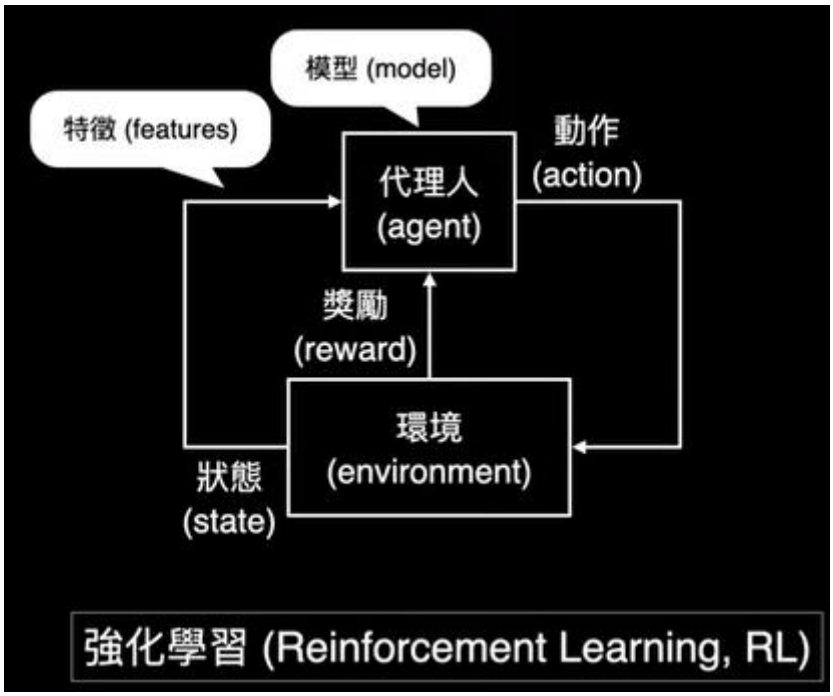
Q-Learning 是強化學習的一種方法。Q-Learning 就是要記錄下學習過的策略，因而告訴電腦什麼情況下該採取什麼行動會有最大獎勵值。Q-Learning 不需對環境進行建模，即使對帶有隨機因素的轉移函數或者獎勵函數也不需要進行特別的

改動就可進行。

對任何有限的馬可夫決策過程（MDP），Q-Learning 可找到一個可最大化所有步驟的獎勵期望的策略。在給定一個部分隨機的策略和無限的探索時間，Q-Learning 可以給出一個最佳的動作選擇策略。Q-Learning 演算法，主要內容為計算狀態與行為對應的最大期望獎勵函式 Q。Q-Learning 最簡單的實現方式就是將獎勵值存儲在一個表格（Q-Table）中，但這種方式受限於狀態和動作空間的數目。Q-Learning 為傳統 RL 演算法，在算法中，有一個稱為 Q-Function 的函數，用於根據狀態估計獎勵。稱之為 Q（state, action），其中 Q 為一個函數，它從狀態（state）和動作（action）計算預期的未來值。

術語	說明	漂流到無人島的人的例子
代理人	對環境採取動作的主體	漂流到無人島的人
環境	代理人所處的世界	無人島
動作	代理人在某個狀態下可採取的動作	移動及休息等
狀態	隨著代理人的動作產生變化的環境元素	此人目前的所在位置等
回饋值	環境根據代理人的動作給予的評價	會隨著生存機率增加而上升的評價
策略	代理人決定動作的原則	此人決策用的方針
回報值	每個動作後的回饋值做加總	一連串的決策與動作預期會取得 3 天份的食物
價值	衡量代理人做這個動作（或處在這個狀態）的指標	很餓的狀態任何食物都很有價值；不餓的狀態愛吃的食物才有價值

（圖 2-8）增強學習的學術用語



（圖 2-9）增強學習（強化學習）的流程

經由上述的研究資料，我們討論出使用降階解法並搭配 AI 增強學習的 Q-Learning 去做程式編程，並用逆序對來判斷有無解，去復原數字容華道。如下圖介紹~

強化學習 : Q learning

(1,3)	(2,3)	(3,3)
(1,2)	(2,2)	(3,2)
(1,1)	(2,1)	(3,1)

目標 :找到金幣寶藏

遊戲假設
- 走到骷髏頭遊戲結束
- 走到金幣寶藏遊戲結束
- 不可走回原路
  (走過的點不能再走)

s_t: 在t時間點的狀態
a_t: 在t時間點的行動
$\alpha = 0.9$
$\gamma = 1$

在開始範例之前，需要知道的幾個定義

- 如何決定下一步該往哪走呢? ⟶ 透過$\underset{a_t}{\text{Max}}\ Q(s_t, a_t)$

  $Q(s_t, a_t)$表示在狀態s_t下，採取某個行為a_t，期望所能得到的最大獎勵 (初始狀態下皆為0，詳見下頁Q table)

- 如何更新$Q(s_t, a_t)$值 $Q^*(s_t, a_t)$:代表更新後的$Q(s_t, a_t)$值

  $$Q^*(s_t, a_t) = Q(s_t, a_t) + \alpha[R(s_t, a_t) + \gamma \underset{a_{t+1}}{\text{Max}}\, Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]$$

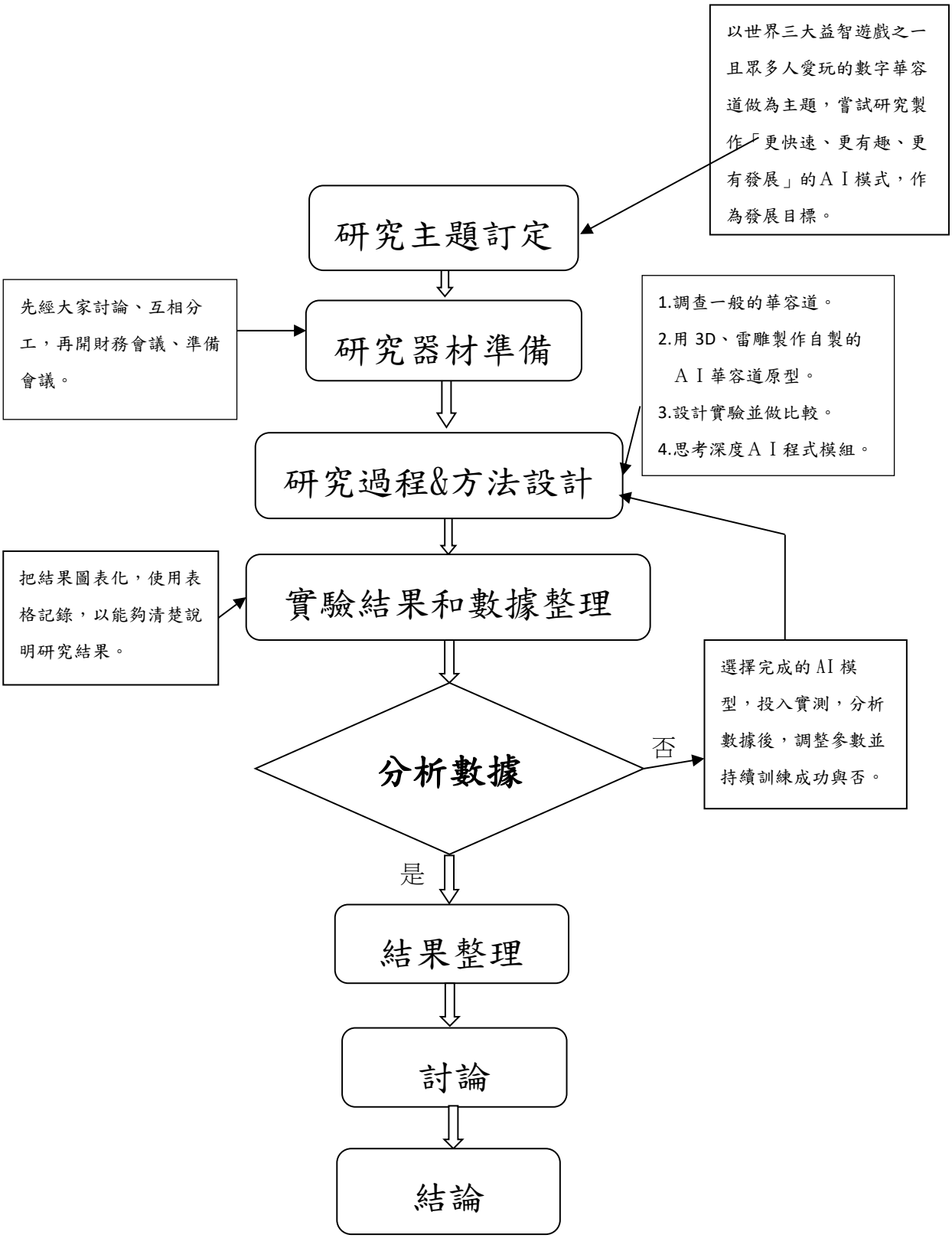
  α : 學習率
  γ : discount rate
  $R(s_t, a_t)$: 給定(s_t, a_t)狀態下reward

- 如何計算(s_t, a_t)狀態下reward

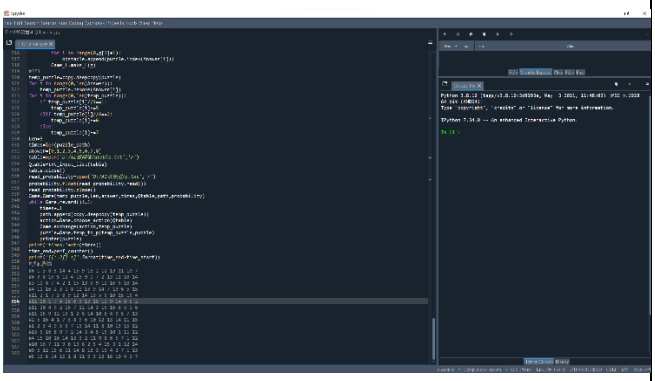
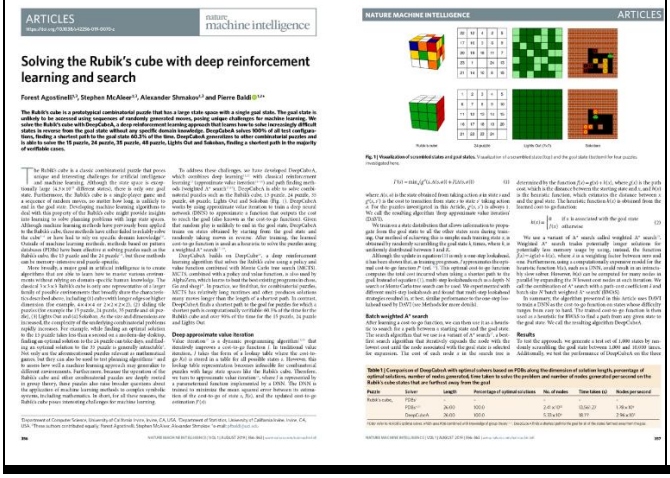
$$R(s_t, a_t) = \begin{cases} 1, & s_{t+1} = (3,2) \\ -1, & s_{t+1} = (3,3) \\ 0, & others \end{cases}$$

（圖 2-10）Q-Learning 模式舉例

（二）研究架構部分


以世界三大益智遊戲之一且眾多人愛玩的數字華容道做為主題，嘗試研究製作「更快速、更有趣、更有發展」的ＡＩ模式，作為發展目標。
研究主題訂定
先經大家討論、互相分工，再開財務會議、準備會議。
研究器材準備
1.調查一般的華容道。
2.用 3D、雷雕製作自製的ＡＩ華容道原型。
3.設計實驗並做比較。
4.思考深度ＡＩ程式模組。
研究過程&方法設計
把結果圖表化，使用表格記錄，以能夠清楚說明研究結果。
實驗結果和數據整理
分析數據
否
選擇完成的AI模型，投入實測，分析數據後，調整參數並持續訓練成功與否。
是
結果整理
討論
結論

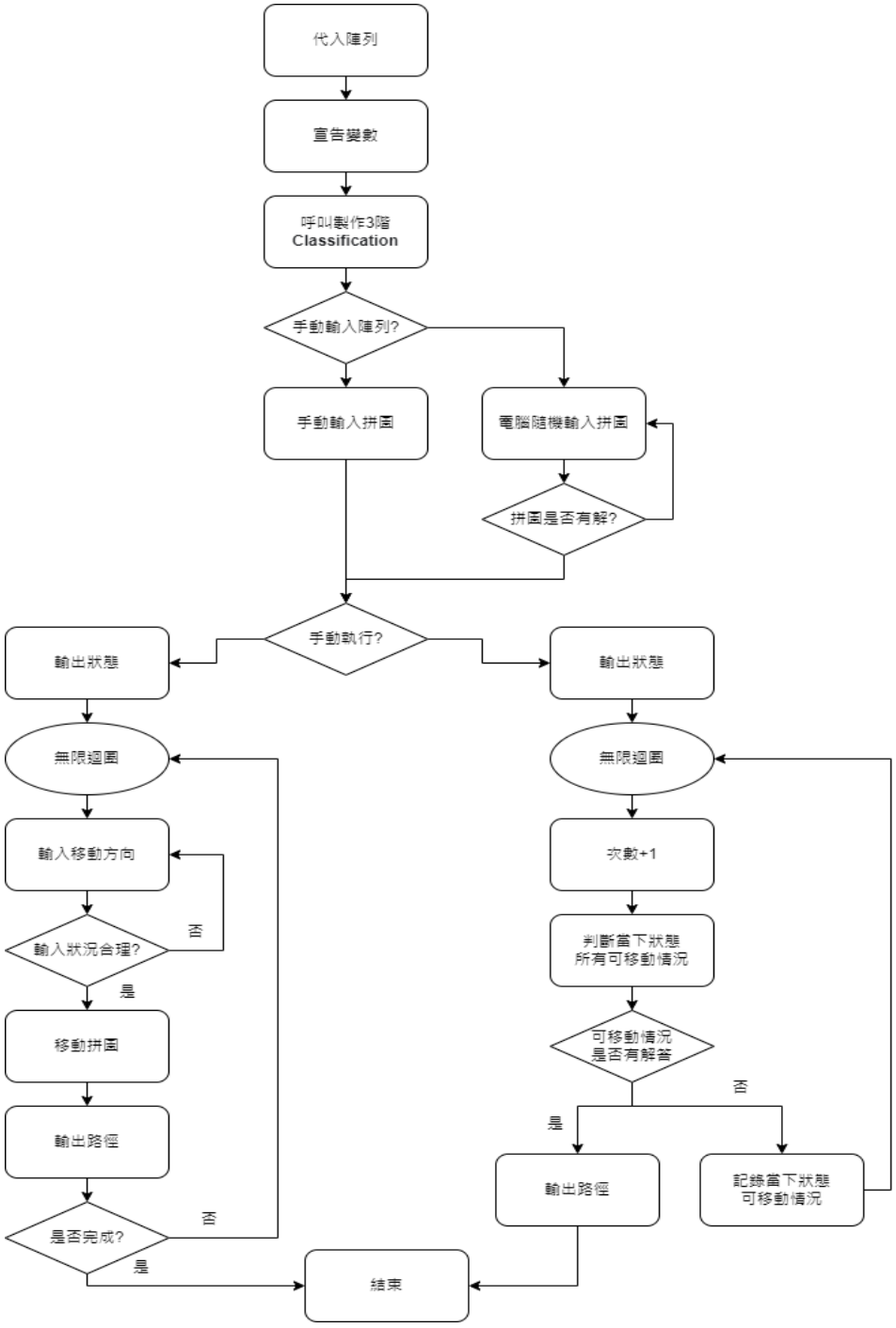
（三）研究設備與器材

<table>
<tr><td>自製華容道作為研究器材(3*3 數字壓克力版)</td><td>學校多台研究電腦()</td></tr>
<tr><td>  </td><td>  </td></tr>
<tr><td>讓我們了解規則及策略的方式</td><td>執行程式專案、統計工具</td></tr>
<tr><td>Python 編譯器(Spyder 5.3.3)</td><td>雷射雕割機與 3D 打印機</td></tr>
<tr><td>  </td><td>  </td></tr>
<tr><td>製作 Python 程式專案和修改程序的工具</td><td>製作自製華容道的工具</td></tr>
<tr><td>Nature 期刊(Solving the Rubik’s cube with deep reinforcement learning and search)</td><td>研究日誌(Research Log)
(耗時三年書寫十本)</td></tr>
<tr><td>  </td><td>  </td></tr>
</table>

(四)配合研究目的二之二的過程

1. 使用 BFS 廣域搜尋演算法解決 3*3 華容道之程式

(1) 根據先前文獻畫出程式流程圖



(圖 2-11) BFS 解決華容道之程式流程圖

(2) 根據流程圖製作程式

A. 製作 3*3 之 Classification

在 3*3 之 Classification 裡有 14 個 function，以下是各個 function 的功能

```
  3   class Game:
  4       def __init__(self,gameBoard,times,blank,path,Len):
  9       def Printer(self,Len,path):
 19       def exchange(self,blank,datum,change):
 23       def bfsXlocation(self,temporarily,Len):
 27       def gameBoardReady(self,answer,a,Len):
 42       def finish(self,b,answer,TIME_s):
 55       def xLocation(self,answer,Len):
 60       def boundary(self,datum,player,times,Len,path):
103       def bfsExchange(self,temporarily,blank,datum,change):
108       def Sort(self,find,All):
118       def bfsnums(self,answer,Len):
131       def bfsnume(self,answer,Len):
140       def binary_search(self,answer,Len,BFS,All):
160       def bfsfinish(self,b,temporarily,TIME_s):
```

（圖 2-12）BFS 解決華容道之程式節錄

a. __init__ ： 初始化變數
b. Printer ： 輸出正確的拼圖格式
c. exchange ： 手動執行時交換位置
d. bfsXlocation ： 電腦執行時尋找空白之位置
e. gameBoardReady ： 判斷拼圖是否無解
f. finish ： 手動執行時判斷是否完成
g. xLocation ： 手動執行時尋找空白之位置
h. boundary ： 判斷移動後是否會超出邊界
i. bfsExchange ： 電腦執行時交換位置
j. Sort ： 將路徑陣列排大小
k. bfsnums ： 將拼圖陣列轉換成編號
l. bfsnume ： 將拼圖編號轉換成陣列
m. binary_search ： 二元搜尋法
n. bfsfinish ： 電腦執行時判斷是否完成

B. 製作 3*3 之變數呼叫及製作架構

在 3*3 除手動執行及電腦執行外寫出大致上之架構

```
174   Len=3
175   start=[]
176   answer=[]
177   path=[]
178   for i in range(0,Len**2):
179     if i!=Len**2-1:
180       start.append(i+1)
181     else:
182       start.append('x')
183   for i in range(0,Len**2):
184     answer.append(0)
185   a=0
186   b=0
187   times=0
188   blank=0
189   Game=Game(answer,times,blank,path,Len)
190   a=int(input('1:手動代入圖片,2:電腦代入圖片\n'))
191   mode=int(input('1:手動執行,2:電腦執行\n'))
192   if a!=1:
193     Game.gameBoardReady(answer,a,Len)
194   else:
195     for i in range(0,9):
196       answer.append(0)
197       answer[i]=int(input('輸入第'+str(i+1)+'個的元素'))
198       if answer[i]==9:
199         answer[i]='x'
200   TIME_s=time.time()
201   if mode==1:
202       #手動執行
219   else:
220       #電腦執行
```

（圖 2-13）BFS 解決華容道之程式節錄

a. 174 ~ 188 ： 呼叫變數
b. 189 ： 呼叫 3*3 之 Classification
c. 190、191 ： 詢問輸入方式以及執行方式
d. 192 ~ 199 ： 代入拼圖
e. 200 ： 紀錄開始時間
f. 201 ~ ： 手動執行或電腦執行

C. 製作 3*3 之手動執行及電腦執行

在 202 ~ 217 為手動執行之程式，而 219 ~ 240 則為電腦用 BFS 執行之程式

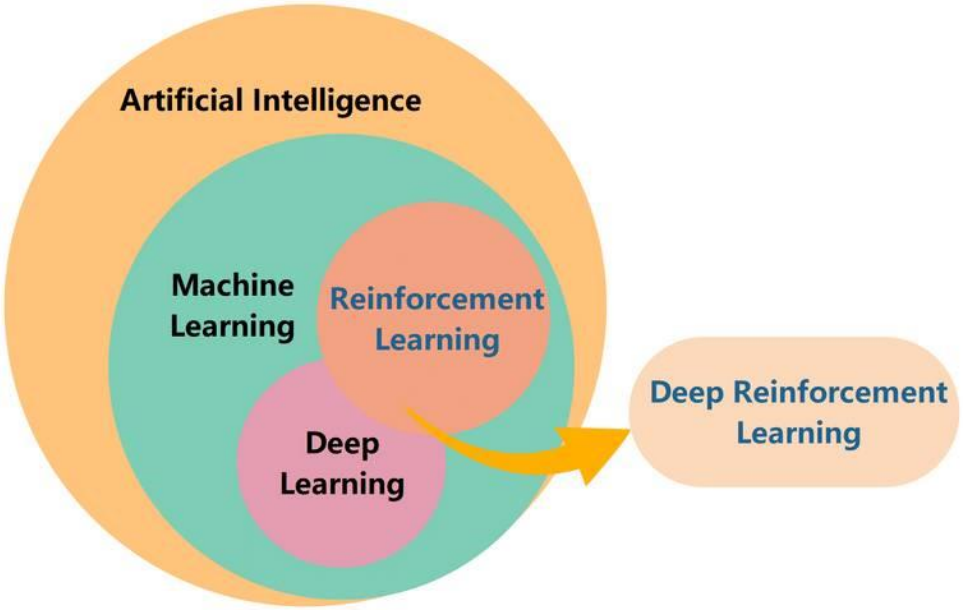
```
201     if mode==1:
202         Game.Printer(Len,path)
203         Game.finish(b,answer,TIME_s)
204         while True:
205           player=str(input())
206           path.append(player)
207           times=times+1
208           datum=Game.xLocation(answer,Len)
209           change=Game.boundary(datum,player,times,Len,path)
210           if change==0:
211             print('error')
212             times=times-1
213             continue
214           else:
215             Game.exchange(blank,datum,change)
216           Game.Printer(Len,path)
217           Game.finish(b,answer,TIME_s)
218     else:
219         Game.Printer(Len,path)
220         Game.finish(b,answer,TIME_s)
221         wasd=['w','a','s','d']
222         bfs=[answer]
223         BFS=[]
224         All=[]
225         while True:
226           times=times+1
227           for i in range(0,len(bfs)):
228             t=copy.deepcopy(bfs[i])
229             for j in range(0,4):
230               temporarily=copy.deepcopy(t)
231               path.append(wasd[j])
232               change=Game.boundary(Game.bfsXlocation(temporarily,Len),wasd[j],times,Len,path)
233               if change!=0:
234                 o=copy.deepcopy(Game.bfsExchange(temporarily,blank,Game.bfsXlocation(temporarily,Len),change))
235                 Game.binary_search(o,Len,BFS,All)
236           print('Level_'+str(times)+':'+str(len(BFS)))
237           for k in range(0,len(BFS)):
238             Game.bfsfinish(b,BFS[k],TIME_s)
239           bfs=BFS
240           BFS=[]
```

（圖 2-14）BFS 解決華容道之程式節錄

a. 202、203 ： 輸出當下狀態並判斷是否為正確答案
b. 205 ~ 207 ： 手動輸入交換方式並記錄次數
c. 208 ~ 215 ： 再輸入合理的情況下移動拼圖
d. 216、217 ： 輸出當下狀態並判斷是否為正確答案
e. 219、220 ： 輸出當下狀態並判斷是否為正確答案
f. 221 ~ 224 ： 代入 BFS 所需要之記錄陣列
g. 225 ~ 240 ： 執行 BFS

2. 使用 AI 增強學習之 Q-Learning 方法解決 3*3 華容道之程式

(1) 參考相關文獻進而分析人工智慧(Artificial Intelligence)以及其各種類型實現方式



(圖 2-15) 人工智慧分類圖

A. 人工智慧(Artificial Intelligence, AI)

為用於研究模擬延伸人類智能的理論、方法、技術及應用系統的一門新的技術科學。是一個範圍極大的科學領域，由機器學習、深度學習、增強學習、深度增強學習等等組成。

B. 機器學習(Machine Learning, ML)

是實現人工智能的一種手段，廣泛應用於自然語言處理、(NLP)、計算機視覺 (CV) 及推薦系統等等方面。能應用機器學習的場景很多，如人臉識別、圖像處理和車牌識別等等，這些在我們生活中以及很常見了。由監督式學習(Supervised Learning)、非監督式學習(Unsupervised Learning)、增強學習、深度增強學習等等組成。

C. 深度學習(Deep Learning, DL)

神經網絡，是深度學習方向的主要工具，雖然在機器學習中也有神經網絡，但深度學習中，神經網絡更複雜，涉及的層數更多。我們了解的 CNN (卷積神經網絡)、RNN (循環神經網絡)、DNN (深度神經網絡) 都是隸屬於深度學習的範疇。

D.增強學習(Reinforcement, RL)

是機器學習的技術之一，又稱強化學習。增強學習就是從環境到行為對應出的學習，強化學習系統必須依靠自身的經歷進行自我學習。強化學習的原理,藉由定義動作(Actions)、狀態(States)、獎勵(Rewards)的方式,不斷訓練機器循序漸進,學會執行某項任務的演算法,常用於動態系統及機器人控制等。

E.深度增強學習(Deep Reinforcement Learning, DRL)

深度強化學習是深度學習與強化學習的結合,具體來說是結合了深度學習的結構和強化學習的思想，但其側重點在深度學習上，解決的依然是決策類問題，不過需要藉助深度神經網絡來解決擬合。深度強化學習是一種與環境交互的訓練神經網絡方法，無現成數據，數據在與環境交互中產生。深度強化學習採用深度神經網絡作為函數擬合器。我們最後參考了在 2019 年 7 月 15 日發表在《nature machine intelligence》雜誌上，名為《Solving the Rubik's cube with deep reinforcement learning and search》裡的 DeepcubeA 而希望做出類似方面的演算法，但因尚未研究透徹 DRL 所以最後跟碩博士討論後想要以 Q-Learning 做出可以解決華容道的程式。

ARTICLES
https://doi.org/10.1038/s42256-019-0070-z
nature machine intelligence

Solving the Rubik's cube with deep reinforcement learning and search

Forest Agostinelli[1,3], Stephen McAleer[2,3], Alexander Shmakov[1,3] and Pierre Baldi[1,2*]

The Rubik's cube is a prototypical combinatorial puzzle that has a large state space with a single goal state. The goal state is unlikely to be accessed using sequences of randomly generated moves, posing unique challenges for machine learning. We solve the Rubik's cube with DeepCubeA, a deep reinforcement learning approach that learns how to solve increasingly difficult states in reverse from the goal state without any specific domain knowledge. DeepCubeA solves 100% of all test configurations, finding a shortest path to the goal state 60.3% of the time. DeepCubeA generalizes to other combinatorial puzzles and is able to solve the 15 puzzle, 24 puzzle, 35 puzzle, 48 puzzle, Lights Out and Sokoban, finding a shortest path in the majority of verifiable cases.

The Rubik's cube is a classic combinatorial puzzle that poses unique and interesting challenges for artificial intelligence and machine learning. Although the state space is exceptionally large (4.3×10^{19} different states), there is only one goal state. Furthermore, the Rubik's cube is a single-player game and a sequence of random moves, no matter how long, is unlikely to end in the goal state. Developing machine learning algorithms to deal with this property of the Rubik's cube might provide insights into learning to solve planning problems with large state spaces. Although machine learning methods have previously been applied to the Rubik's cube, these methods have either failed to reliably solve the cube[1–4] or have had to rely on specific domain knowledge[5,6]. Outside of machine learning methods, methods based on pattern databases (PDBs) have been effective at solving puzzles such as the Rubik's cube, the 15 puzzle and the 24 puzzle[7–9], but these methods can be memory-intensive and puzzle-specific.

More broadly, a major goal in artificial intelligence is to create algorithms that are able to learn how to master various environments without relying on domain-specific human knowledge. The classical $3\times3\times3$ Rubik's cube is only one representative of a larger family of possible environments that broadly share the characteristics described above, including (1) cubes with longer edges or higher dimension (for example, $4\times4\times4$ or $2\times2\times2\times2$), (2) sliding tile puzzles (for example the 15 puzzle, 24 puzzle, 35 puzzle and 48 puzzle), (3) Lights Out and (4) Sokoban. As the size and dimensions are increased, the complexity of the underlying combinatorial problems rapidly increases. For example, while finding an optimal solution to the 15 puzzle takes less than a second on a modern-day desktop, finding an optimal solution to the 24 puzzle can take days, and finding an optimal solution to the 35 puzzle is generally intractable[9]. Not only are the aforementioned puzzles relevant as mathematical games, but they can also be used to test planning algorithms[10] and to assess how well a machine learning approach may generalize to different environments. Furthermore, because the operation of the Rubik's cube and other combinatorial puzzles are deeply rooted in group theory, these puzzles also raise broader questions about the application of machine learning methods to complex symbolic systems, including mathematics. In short, for all these reasons, the Rubik's cube poses interesting challenges for machine learning.

To address these challenges, we have developed DeepCubeA, which combines deep learning[11,12] with classical reinforcement learning[13] (approximate value iteration[14–16]) and path finding methods (weighted A* search[17,18]). DeepCubeA is able to solve combinatorial puzzles such as the Rubik's cube, 15 puzzle, 24 puzzle, 35 puzzle, 48 puzzle, Lights Out and Sokoban (Fig. 1). DeepCubeA works by using approximate value iteration to train a deep neural network (DNN) to approximate a function that outputs the cost to reach the goal (also known as the cost-to-go function). Given that random play is unlikely to end in the goal state, DeepCubeA trains on states obtained by starting from the goal state and randomly taking moves in reverse. After training, the learned cost-to-go function is used as a heuristic to solve the puzzles using a weighted A* search[17–19].

DeepCubeA builds on DeepCube[20], a deep reinforcement learning algorithm that solves the Rubik's cube using a policy and value function combined with Monte Carlo tree search (MCTS). MCTS, combined with a policy and value function, is also used by AlphaZero, which learns to beat the best existing programs in chess, Go and shogi[21]. In practice, we find that, for combinatorial puzzles, MCTS has relatively long runtimes and often produces solutions many moves longer than the length of a shortest path. In contrast, DeepCubeA finds a shortest path to the goal for puzzles for which a shortest path is computationally verifiable: 60.3% of the time for the Rubik's cube and over 90% of the time for the 15 puzzle, 24 puzzle and Lights Out.

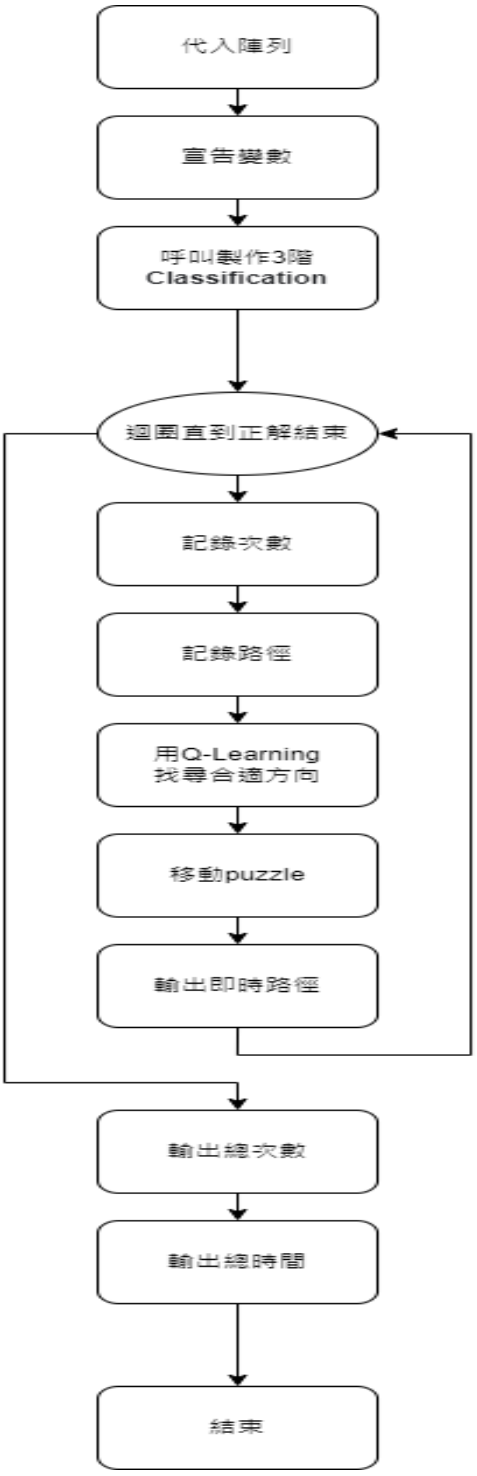
Deep approximate value iteration

Value iteration[15] is a dynamic programming algorithm[14,16] that iteratively improves a cost-to-go function J. In traditional value iteration, J takes the form of a lookup table where the cost-to-go $J(s)$ is stored in a table for all possible states s. However, this lookup table representation becomes infeasible for combinatorial puzzles with large state spaces like the Rubik's cube. Therefore, we turn to approximate value iteration[16], where J is represented by a parameterized function implemented by a DNN. The DNN is trained to minimize the mean squared error between its estimation of the cost-to-go of state s, $J(s)$, and the updated cost-to-go estimation $J'(s)$:

[1]Department of Computer Science, University of California Irvine, Irvine, CA, USA. [2]Department of Statistics, University of California Irvine, Irvine, CA, USA. [3]These authors contributed equally: Forest Agostinelli, Stephen McAleer, Alexander Shmakov. *e-mail: pfbaldi@uci.edu

（圖 2-16）Solving the Rubik's cube with deep reinforcement learning and search 節錄

（2）根據先前文獻畫出 Q-Learning 程式流程圖



（圖 2-17）Q-Learning 解決數字華容道之程式流程圖

(3) 根據流程圖製作程式

A. 製作 3*3 之 Classification

在 3*3 之 Classification 裡有 8 個 function，以下是各 function 的功能

```
 4   class Game:
 5       def __init__(self,temp_puzzle,Len,answer,times,Qtable,path,probability):
13       def exchange(self,action,p):
27       def reward(self):
31       def nums(self,temp_puzzle):
36       def Class(self,num):
41       def boundary(self,p):
52       def state(self,temp_puzzle):
62       def choose_action(self,Qtable):
```

（圖 2-18）Q-Learning 解決華容道之程式節錄

a. __init__ ： 初始化變數
b. exchange ：交換位置
c. reward ： 計算當下拼圖分數
d. nums ： 將拼圖陣列轉換為數字編號
e. Class ： 配合 nums 將數字編號轉為陣列編號
f. boundary ： 判斷移動後是否會超出邊界
g. state ： 尋找陣列在 Q-table 裡的正確位置
h. choose_action ： 找尋最終移動方案

B. 製作代入 Q-table 之 function 以及宣告變數

```
 96   def txt_input_list(txt):
104   def printer(puzzle):
115   puzzle=input('puzzle:').split()
116   time_start=perf_counter()
117   for i in range(0,len(puzzle)):
118       puzzle[i]=int(puzzle[i])-1
119   temp_puzzle=copy.deepcopy(puzzle)
120   Len=3
121   times=0
122   answer=[0,1,2,3,4,5,6,7,8]
123   path=[]
124   table=open('D:/AI華榮道/qtable.txt','r')
125   Qtable=txt_input_list(table)
126   table.close()
127   read_probability=open('D:/AI華榮道/p.txt','r')
128   probability=float(read_probability.read())
129   read_probability.close()
130   Game=Game(temp_puzzle,Len,answer,times,Qtable,path,probability)
```

（圖 2-19）Q-Learning 解決華容道之程式節錄

a. txt_input_list ： 將文字檔案轉換為陣列格式
b. printer ： 輸出正確的拼圖格式
c. 115 ~ 129 ： 宣告變數
d. 130 ： 呼叫解決 3*3 華容道之 Classification

C. 製作處理 3*3 華容道之迴圈

```
131
132 while Game.reward()!=1:
133     times+=1
134     path.append(copy.deepcopy(temp_puzzle))
135     action=Game.choose_action(Qtable)
136     Game.exchange(action,temp_puzzle)
137     printer(puzzle)
138 print('times:'+str(times))
139 time_end=perf_counter()
    print('[{:.2f} s]'.format(time_end-time_start))
```

（圖 2-20）Q-Learning 解決華容道之程式節錄

a. 131 ： 執行迴圈直到變成答案

b. 132、133 ： 紀錄次數、路徑

c. 134 ： 選擇移動方向

d. 135 ： 移動華容道

e. 136 :輸出正確格式

f. 137 ~ 139 ： 當完成華容道後記錄次數與時間

(4) training 3*3 華容道之 Qtable

Qtable，為 Q-Learning 中類似計分板的陣列，用來紀錄 Q-Learning 在選擇狀態(state)與決定行為(action)後與原本決策的差異分數。我們希望將 3*3 華容道的每種狀態 training 10 次，已達成執行 Q-Learning 需要大量資料的需求。已知 3*3 華容道所有狀態為 181440 種，我們製作了一個可 training 數據的程式，並用文字檔案紀錄結果。(見研究結果與附錄一)

(5) 比較 3*3 華容道用 Q-Learning 與 BFS 之差異

(1)隨機選取 10 種 3*3 華容道的狀態並進行「時間」比較

選取 10 種有解狀態後，每一種狀態執行 10 次並計算平均值後比較。(見研究結果與附錄二)

(2)隨機選取 10 種 3*3 華容道的狀態並進行「次數」比較

選取 10 種有解狀態後，每一種狀態執行 10 次並計算平均值後比較。(見研究結果與附錄三)

3. 分析 AI 增強學習的 Q-Learning 方法，對 4*4 數字華容道作可行性分析

（1）初期構思

我們在已完成的 3*3 華容道基礎下，希望可以做出 4*4 的數字華容道，而我們有想了三種辦法，希望其中有實施可能性較大的方法可以嘗試。

A.使用層解方式完成

我們使用層解的方式先達成拼圖的上半部分再使用 Q-Learning 解決剩下 4*2 的問題。

B.使用降階方式完成

我們使用降階的方式先達成拼圖的左上角 L 行部分再使用 Q-Learning 解決剩下 3*3 的問題。

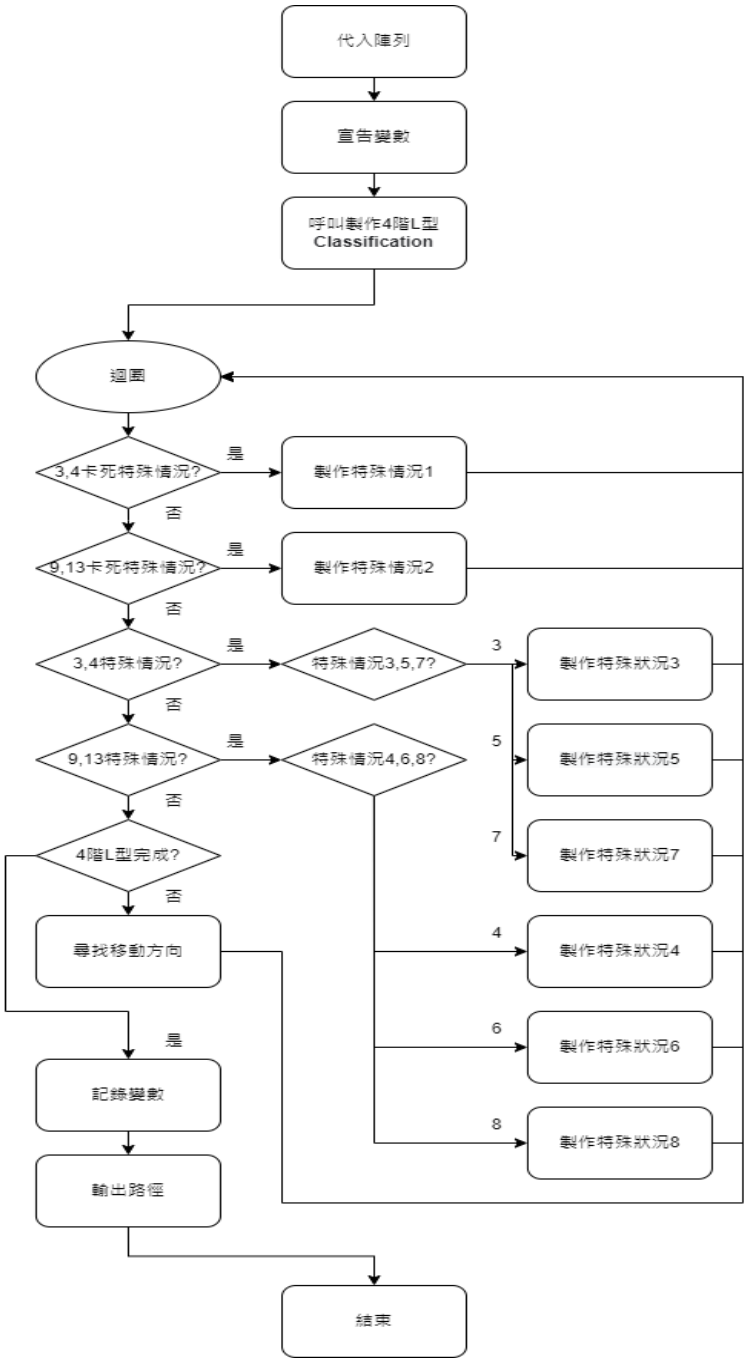
C.不切割直接完成

我們直接使用 Q-Learning 解決 4*4 的問題，不去使用其他切割方式。

最後經由討論我們決定先嘗試直接完成的方式，若無法完成再使用降階或層解的方式，不過實在由於組合太過龐大（約二十兆種），試完後決定採用第二種模式（降階方式）加以處理。

(2) 根據先前文獻畫出程式流程圖（第二種模式-降階方式）

我們一開始嘗試不分割 4*4 華容道直接完成，但因計算排列組合次數後發現執行所有 4*4 華容道的 Q-table 狀態會高達 20 兆種。電腦測試難以負荷，因此將不切割 4*4 華容道的方式作為廢案，而嘗試使用降階的方式，解決 4*4 華容道的難題，並獲成功！



（圖 2-21）解決 4*4 降階之程式流程圖

(3) 根據流程圖製作程式

A.配合流程圖製作 4*4 降階之 Classification

在 4*4 降階之 Classification 裡有 13 個 function，以下是各 function 的

功能

```
4   class Game_L:
5       def __init__(self,Len,puzzle,answer,path,obstacle):
11      def goal(self):
22      def exchange(self,a,b):
26      def b_Obst(self,obstacle,value):
34      def g_Obst(self,obstacle,value):
40      def distance(self,a,b):
42      def b_wasd(self,b_goal,b,obstacle):
61      def choose_direction(self,wasd):
79      def goal_wasd(self,g,b,answer,Wasd):
106     def moving_path(self,g_direction,g,obstacle):
118     def swap(self,state):
135     def make_L(self,g):
145     def shortcut(self,path):
```

（圖 2-22）解決 4*4 降階之程式節錄

a. __init__ ： 初始化變數
b. goal ： 判斷目標位置
c. exchange ：交換位置
d. b_Obst ： 以空白位置為基準判斷障礙物之位置
e. g_Obst ： 以目標位置為基準判斷障礙物之位置
f. distance ： 判斷點 a 到點 b 之距離
g. b_wasd ： 以空白位置為基準判斷可合理移動之位置
h. choose_direction ： 判斷移動方向
i. goal_wasd ： 以目標位置為基準判斷可合理移動之位置
j. moving_path ： 移動拼圖路徑
k. swap ： 處裡特殊情況
l. make_L ： 統整上方 function
m. shortcut: 判斷是否走過相同的路徑

B. 製作輸出正確格式之 function 以及宣告變數

```
160 def printer(puzzle):
171     Len=4
172     puzzle=input('puzzle:').split()
173     time_start=perf_counter()
174     for i in range(0,len(puzzle)):
175         puzzle[i]=int(puzzle[i])-1
176     answer=[0,1,3,7,4,12,13]
177     Answer=[0,1,2,3,4,8,12]
178     copy_puzzle=copy.deepcopy(puzzle)
179     path=[copy_puzzle]
180     obstacle=[]
181     Game_L=Game_L(Len,puzzle,answer,path,obstacle)
```

（圖 2-23）解決 4*4 降階之程式節錄

a. printer ： 輸出正確的拼圖格式
b. 171 ~ 180 ： 宣告變數
c. 181 ： 呼叫解決 4*4 降階華容道之 Classification

C. 製作處理 4*4 降階之迴圈

```
182  while True:
183      if puzzle[2]==15 and puzzle[3]==2 and puzzle[6]==3:
184          Game_L.swap(1)
185          answer=[0,1,2,3,4,12,13]
186      elif puzzle[8]==15 and puzzle[9]==12 and puzzle[12]==8:
187          Game_L.swap(2)
188          answer=[0,1,2,3,4,8,12]
189      elif puzzle[3]==2 and puzzle[7]==3:
190          if puzzle[6]==15:
191              Game_L.swap(3)
192              answer=[0,1,2,3,4,12,13]
193          elif puzzle[11]==15:
194              Game_L.swap(5)
195              answer=[0,1,2,3,4,12,13]
196          else:
197              Game_L.swap(7)
198              answer=[0,1,2,3,4,12,13]
199      elif puzzle[12]==8 and puzzle[13]==12:
200          if puzzle[9]==15:
201              Game_L.swap(4)
202              answer=[0,1,2,3,4,8,12]
203          elif puzzle[14]==15:
204              Game_L.swap(6)
205              answer=[0,1,2,3,4,8,12]
206          else:
207              Game_L.swap(8)
208              answer=[0,1,2,3,4,8,12]
209      else:
210          g=Game_L.goal()
211          if g=='complete!':
212              puzzle_path=Game_L.shortcut(path)
213              for i in range(0,len(puzzle_path)):
214                  printer(puzzle_path[i])
215              break
216          b=puzzle.index(15)
217          obstacle=[]
218          for i in range(0,g[1]+1):
219              obstacle.append(puzzle.index(Answer[i]))
220          Game_L.make_L(g)
```

（圖 2-24）解決 4*4 降階之程式節錄

a. 183 ~ 185 ： 解決第一種特殊情況(當特殊狀況出現)
b. 186 ~ 188 ： 解決第二種特殊情況(當特殊狀況出現)
c. 189 ~ 198 ： 整合三、五、七特殊情況判斷式
   190 ~ 192 ： 解決第三種特殊情況(當特殊狀況出現)
   193 ~ 195 ： 解決第五種特殊情況(當特殊狀況出現)
   196 ~ 198 ： 解決第七種特殊情況(當特殊狀況出現)
d. 199 ~ 208 ： 整合四、六、八特殊情況判斷式
   200 ~ 202 ： 解決第四種特殊情況(當特殊狀況出現)
   203 ~ 205 ： 解決第六種特殊情況(當特殊狀況出現)
   206 ~ 208 ： 解決第八種特殊情況(當特殊狀況出現)
e. 210 ~ 215 ： 非特殊狀況，解決是否為答案狀況
f. 216 ~ 219 ： 為普通狀況，宣告變數
g. 220 ： 執行 4*4 華容道 L 型 Classifition

4. 整合 4*4 降階以及 3*3 華容道程式

我們整合程式讓電腦可以執行 4*4 之華容道，隨機選取 10 個有解 4*4 華容道之狀態，每個狀態執行 100 次並用折線圖記錄完成時間以及平均時間和次數（見研究結果與附錄四）。

三、 研究結果與討論

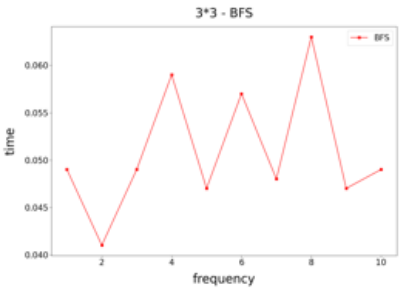
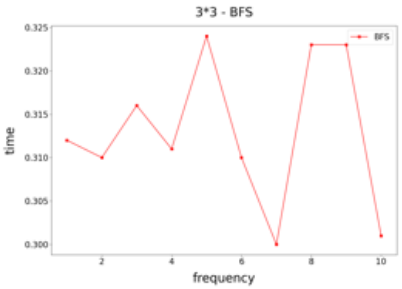
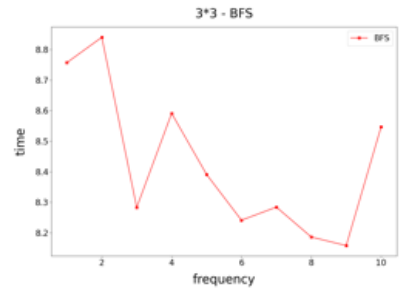
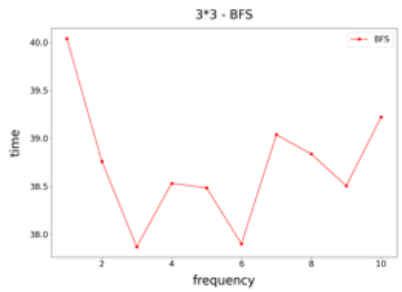
(一)研究結果

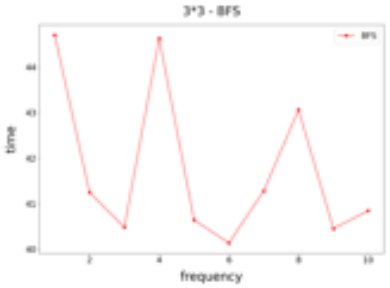
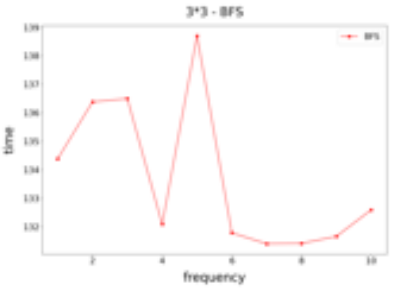
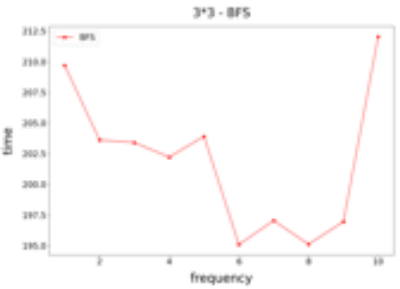
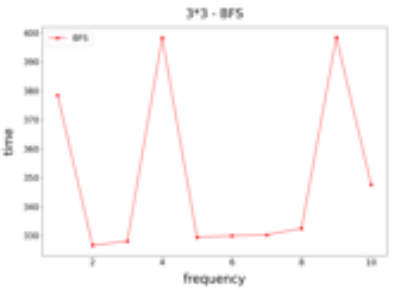
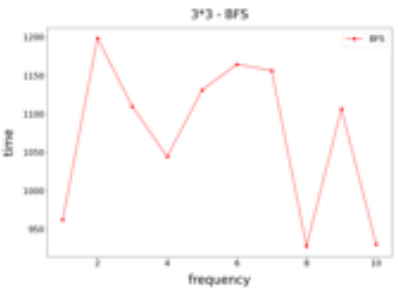
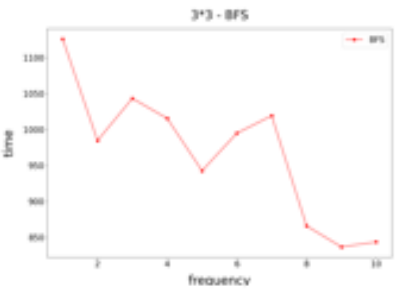
1. 實驗部分

我們隨機選取 3*3 華容道之十種狀態，分別採用 BFS 模式，藉以比較次數與時間的差異。

(1)BFS 3*3 華容道之次數及時間之統計

我們先選取 10 種狀態之拼圖並用程式執行 10 後記錄完成的時間以及次數進行分析。

圖片	state	圖表
2 5 3 4 6 7 1 8	253496718	
1 2 7 5 6 8 3 4	192756834	
2 3 6 7 4 8 5 1	236749851	
7 2 6 5 1 8 3 4	726519834	

圖片	state	圖表
2 6 4 7 1 8 3 5	264719835	
4 7 5 8 1 2 6 3	475981263	
8 6 2 5 3 1 7 4	869253174	
7 2 6 5 4 8 1 3	792654813	
3 4 7 5 1 2 6 8	347519268	
8 6 7 2 5 4 3 1	867254391	

State	1	2	3	4	5	6	7	8	9	10	平均
253496718	10	10	10	10	10	10	10	10	10	10	10
192756834	13	13	13	13	13	13	13	13	13	13	13
236749851	17	17	17	17	17	17	17	17	17	17	17
726519834	19	19	19	19	19	19	19	19	19	19	19
264719835	20	20	20	20	20	20	20	20	20	20	20
475981263	21	21	21	21	21	21	21	21	21	21	21
869253174	22	22	22	22	22	22	22	22	22	22	22
792654813	23	23	23	23	23	23	23	23	23	23	23
347519268	27	27	27	27	27	27	27	27	27	27	27
867254391	31	31	31	31	31	31	31	31	31	31	31

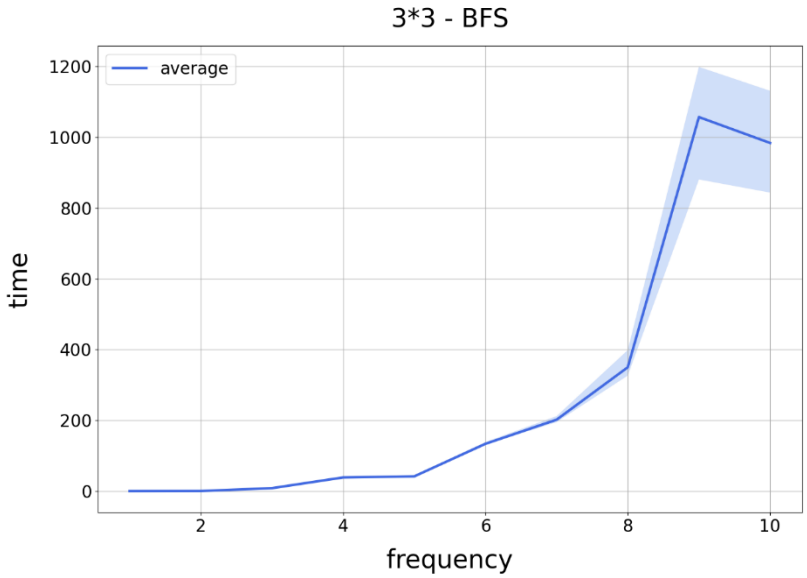
（圖 3-1）BFS 解決 3 階之次數記錄表

我們發現在使用 BFS 解決時因為是用展開樹狀圖去進行搜尋，所以次數上並沒有區別，且必為最短解。

State	1	2	3	4	5	6	7	8	9	10	平均
253496718	0.049	0.041	0.049	0.059	0.047	0.057	0.048	0.063	0.047	0.049	0.0509
192756834	0.312	0.31	0.316	0.311	0.324	0.31	0.3	0.323	0.323	0.301	0.313
236749851	8.757	8.84	8.283	8.591	8.391	8.241	8.284	8.186	8.158	8.547	8.4278
726519834	40.041	38.76	37.869	38.531	38.483	37.899	39.038	38.836	38.504	39.222	38.7183
264719835	44.703	41.251	40.484	44.631	40.646	40.142	41.278	43.064	40.455	40.848	41.7502
475981263	134.383	136.383	136.5	132.088	138.703	131.785	131.401	131.414	131.66	132.591	133.691
869253174	209.708	203.626	203.411	202.228	203.914	195.095	196.948	195.121	196.958	212.041	201.905
792654813	378.474	326.614	328.051	398.363	329.435	329.881	330.253	332.474	398.439	347.495	349.948
347519268	962.618	1198.535	1109.602	1044.422	1131.031	1164.678	1156.545	928.034	1106.583	930.477	1073.25
867254391	1126.216	984.847	1043.49	1015.601	942.794	995.002	1019.367	866.023	836.73	843.61	967.368

（圖 3-2）BFS 解決 3 階之時間記錄表

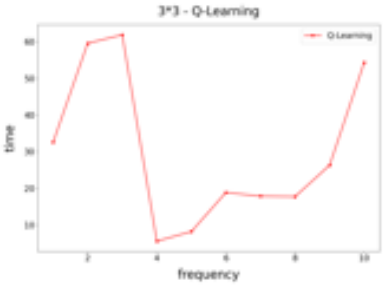
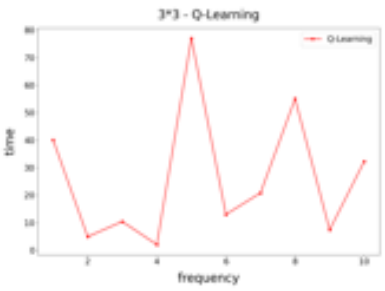
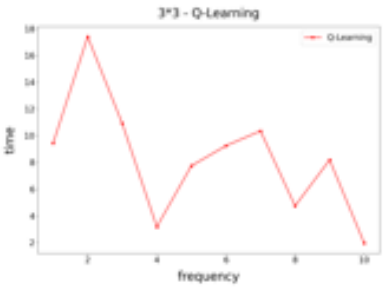
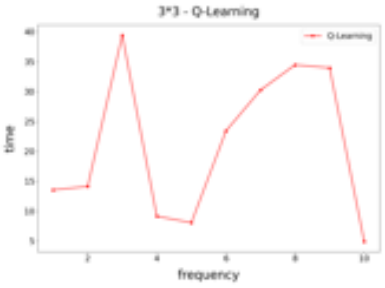
在時間上我們也發現時間絕大部分會隨著次數呈現正相關成長，但因為搜尋狀態不同導致次數會隨之不同，而造成時間上的改變。

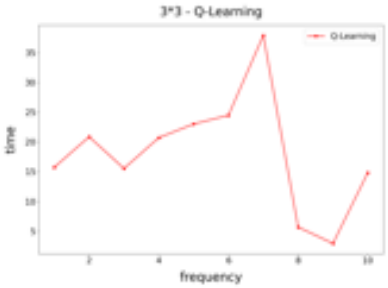
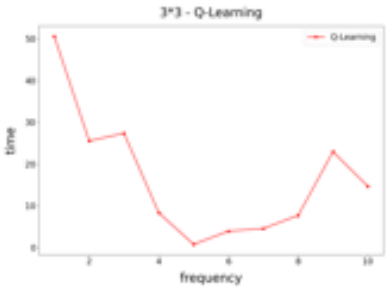
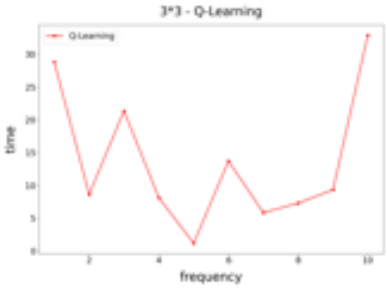
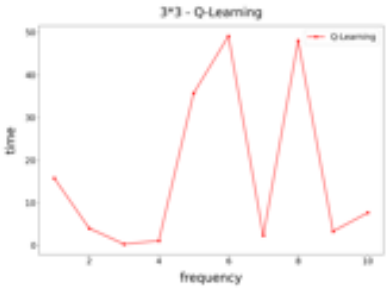
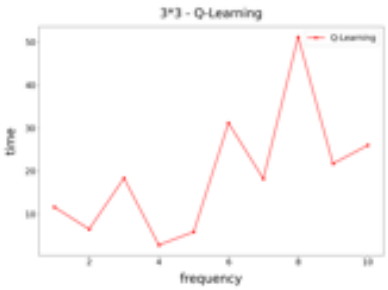
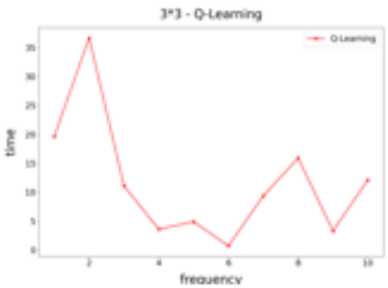


（圖 3-3）BFS 解決 3 階之時間折線分布圖

(2) Q-Learning 3*3 華容道之次數及時間之統計

我們先選取 10 種狀態之拼圖並用程式執行 10 後記錄完成的時間以及次數進行分析。

圖片	state	圖表
2 5 3 4 6 7 1 8	253496718	
1 2 7 5 6 8 3 4	192756834	
2 3 6 7 4 8 5 1	236749851	
7 2 6 5 1 8 3 4	726519834	

圖片	state	圖表
2 6 4 7 1 8 3 5	264719835	
4 7 5 8 1 2 6 3	475981263	
8 6 2 5 3 1 7 4	869253174	
7 2 6 5 4 8 1 3	792654813	
3 4 7 5 1 2 6 8	347519268	
8 6 7 2 5 4 3 1	867254391	

State	1	2	3	4	5	6	7	8	9	10	平均
253496718	454822	840530	1128676	89382	101208	250924	330214	245338	492552	747070	468072
192756834	54025	65285	145123	26355	907895	181325	296693	718879	98481	454539	294860
236749851	192895	384193	246219	61793	54247	168153	226059	98553	171621	36187	163992
726519834	294123	297585	870147	207859	158759	513403	649821	737163	718193	90377	453743
264719835	326769	440587	349367	453477	528957	536383	849729	124291	46773	320103	397644
475981263	1099869	463757	493713	145111	9673	70059	80159	140307	430967	275903	320952
869253174	535340	151364	399002	147324	7770	246934	101280	136640	167992	615386	250903
792654813	267851	77367	1993	17767	666811	452177	20755	456287	85693	200001	224670
347519268	203987	164437	494601	74381	127783	804353	469101	1304041	545609	648747	483704
867254391	355933	962423	273977	94447	101759	11327	235345	400575	85117	310523	283143

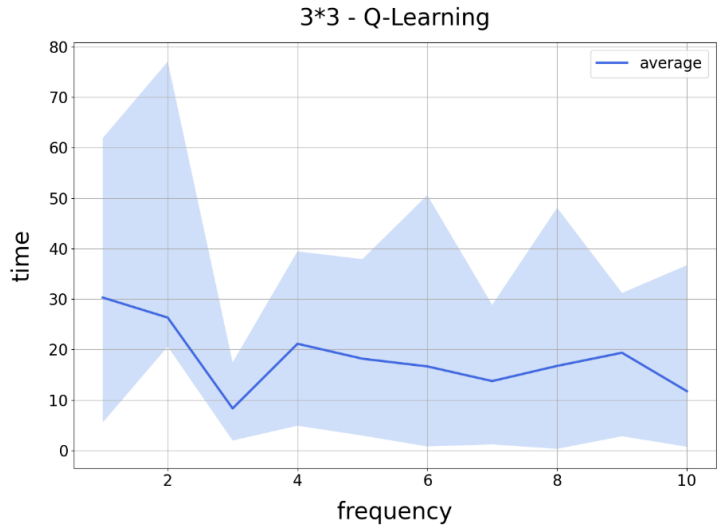
（圖 3-4）Q-Learning 解決 3 階之次數記錄表

我們發現在使用 Q-Learning 解決時因為不是是用展開樹狀圖去進行搜尋，所以次數上會因機率等等問題而有所區別，也就不會是最短步數解。

State	1	2	3	4	5	6	7	8	9	10	平均
253496718	32.58	59.58	61.88	5.57	8.19	18.86	17.84	17.72	26.31	54.31	30.284
192756834	40.1	4.92	10.45	2.06	77.02	13.02	20.85	55.03	7.41	32.27	26.313
236749851	9.44	17.4	10.91	3.18	7.74	9.24	10.35	4.74	8.18	1.96	8.314
726519834	13.54	14.12	39.43	9.13	8.06	23.4	30.25	34.47	33.97	4.9	21.127
264719835	15.74	20.86	15.54	20.7	23.01	24.48	37.89	5.68	2.94	14.83	18.167
475981263	50.56	25.54	27.37	8.32	0.79	3.95	4.58	7.7	22.94	14.67	16.642
869253174	28.82	8.64	21.38	8.13	1.19	13.73	5.86	7.3	9.35	32.92	13.732
792654813	15.76	4.04	0.31	1.1	35.72	49.12	2.3	48.06	3.31	7.67	16.739
347519268	11.64	6.49	18.35	2.81	5.85	31.18	18.24	51.18	21.71	26.01	19.346
867254391	19.64	36.68	11.09	3.6	4.9	0.72	9.37	15.91	3.31	12.12	11.734

（圖 3-5）Q-Learning 解決 3 階之時間記錄表

在時間上我們發現時間並不會隨著次數呈現正相關成長，除了搜尋狀態不同導致次數會隨之不同，進而造成時間上的改變之外，Q-Learning 在每一種狀態表現出的次數也因回饋的獎賞及機率而有所差別。



（圖 3-6）Q-Learning 解決 3 階之時間折線分布圖

(3) 對上述 BFS 及 Q-Learning 進行對比分析

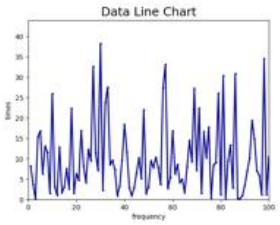
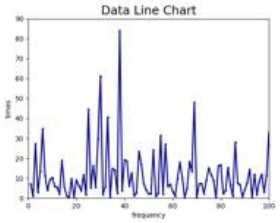
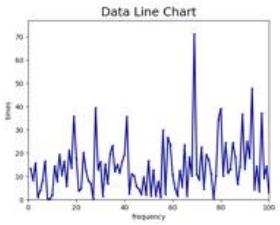
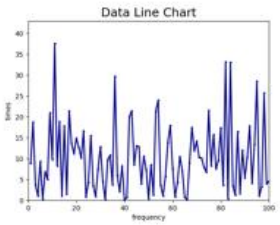
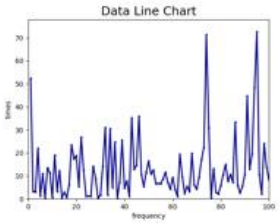
A. BFS 若在次數較短的情況下時間是可以少於 Q-Learning 的，但在次數增加後會因紀錄的狀態越來越大導致需要的時間越來越久。

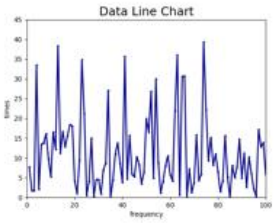
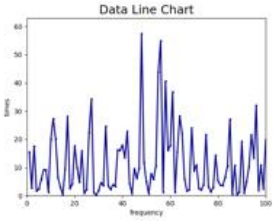
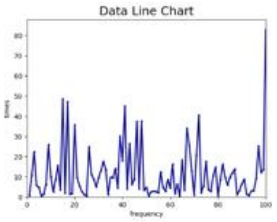
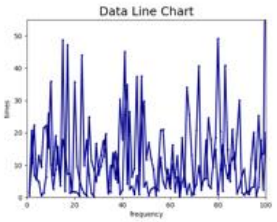
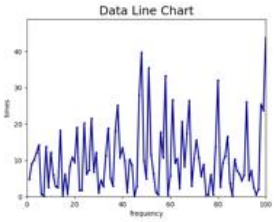
B. Q-Learning 所記計算出的時間不會因為狀態最短步數而有所改變。

C. 若要縮短 Q-Learning 執行的時間，可以增加訓練的次數達到更快的效果

(4) Q-Learning 4*4 華容道之次數及時間之統計

我們先選取 10 種狀態之拼圖並用程式執行 100 次後記錄完成的時間以及次數進行分析。

圖片	state	圖表
	6 1 3 8 5 14 4 15 9 16 2 12 13 11 10 7	
	6 3 8 15 5 12 4 16 9 1 7 2 13 11 10 14	
	3 11 8 7 4 2 1 15 13 5 9 12 16 6 10 14	
	4 11 16 2 3 1 8 12 13 9 14 7 10 6 5 15	
	11 2 1 7 3 8 9 12 14 13 5 6 10 15 16 4	

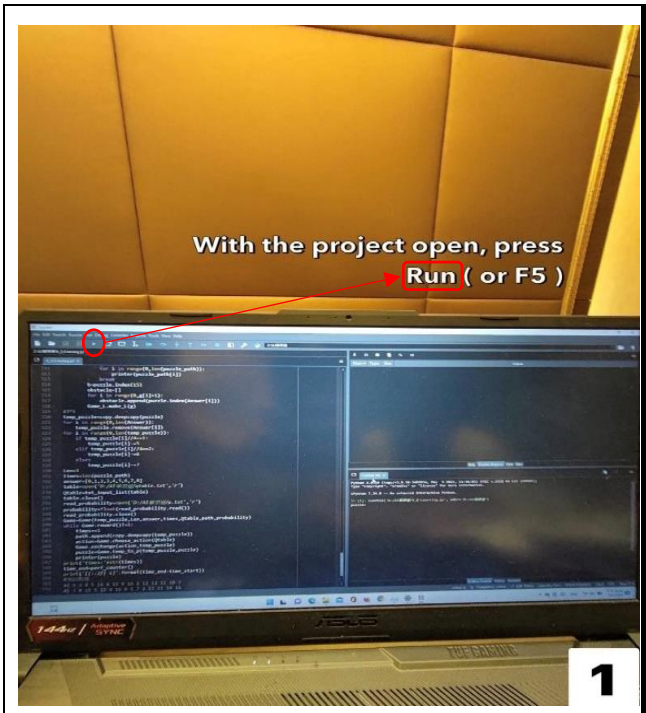
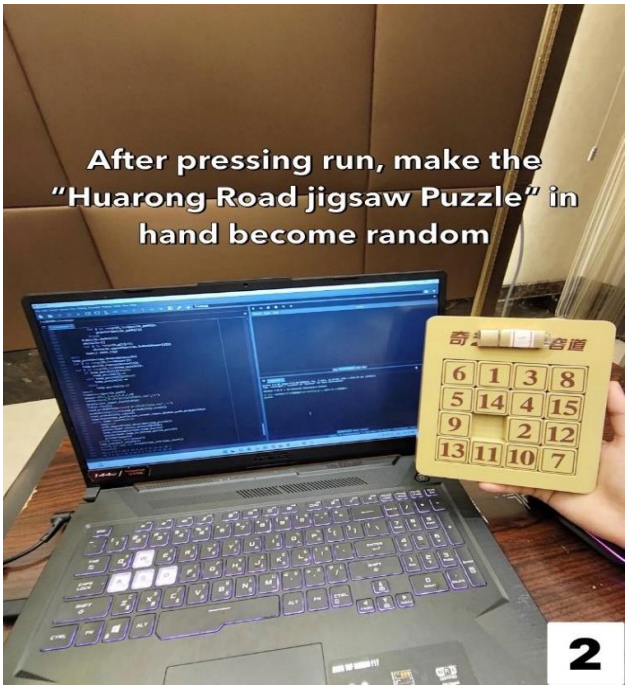
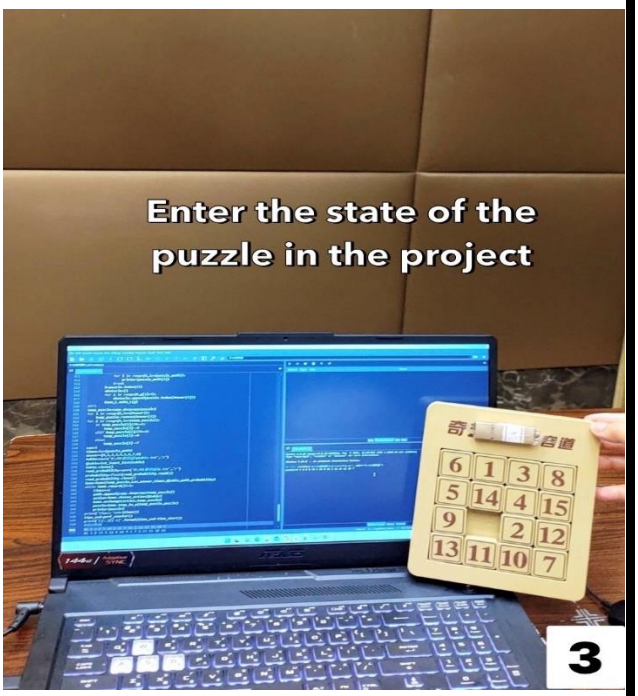
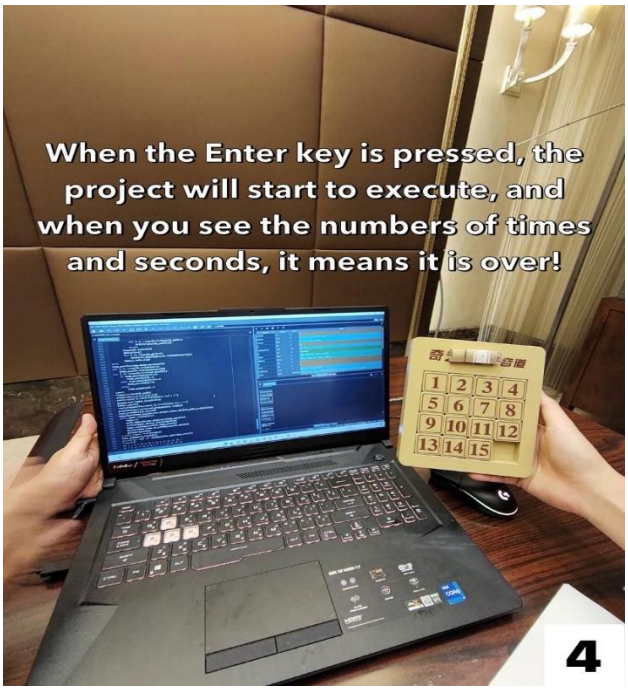
圖片	state	圖表
	11 10 1 7 4 16 6 3 13 15 12 9 14 8 5 2	
	11 10 4 9 2 15 7 12 14 3 13 16 8 5 1 6	
	11 15 9 12 16 1 2 6 14 10 8 4 3 5 7 13	
	2 5 16 4 1 7 3 8 9 6 10 12 13 14 11 15	
	1 2 3 4 9 5 6 7 16 14 11 8 10 13 15 12	

（圖 3-7）Q-Learning 綜合解法，解決 4 階隨機狀態之處理圖

我們藉由上方圖表發現隨機選取的華榮道狀態大部分完成時間可在 50~60 秒內完成

2. 實際落地實施部分

（1）實際落地實施程序

<table>
<tr>
<td> 
</td>
<td> 
</td>
</tr>
<tr>
<td>打開電腦，進入 Spider，
按下箭頭所指的『RUN』鍵</td>
<td>接著『手動』輸入數字華容道
要解的順序</td>
</tr>
<tr>
<td> 
</td>
<td> 
</td>
</tr>
<tr>
<td>按『Enter』鍵，就開始跑啦~</td>
<td>當程式跑完會看到『次數』與『時間』~</td>
</tr>
</table>

(2)小朋友跟電腦比賽圖

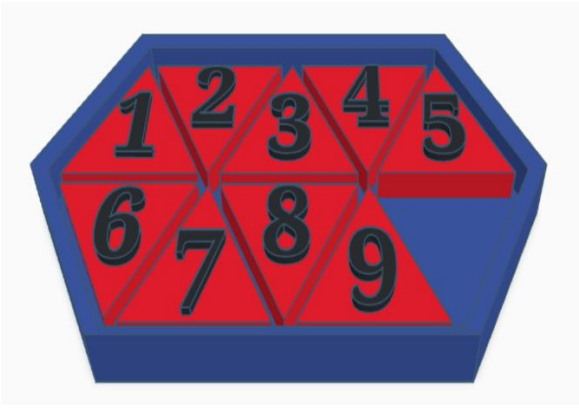
（圖 3-8）小朋友用 4*4 華容道跟電腦比賽

（圖 3-9）大家相互討論問題

(二) 討論

事情只有討論才會更加完整，我們在研究期間固定與指導老師會議討論，藉以明確研究的方向和重點，整理幾項重要討論要點：

1. 數字華容道如果經由排列組合隨機選取狀態，會有一半無解的情況，我們可以利用反證法確定程式是正確的，只要無解的狀況無法解出，有解的卻解的出答案就能確定程式正確。
2. 經由文獻得之 4*4 數字華容道最複雜情況的最短解僅為 80 步，若需要讓次數時間有質的提升，需要利用深度增強學習（DRL）的技術，這需要相當複雜的數學知識如微積分、線性代數、統計等等…，與人工智慧更高階的演算法配合才行。
3. 若要解決 5*5 數字華容道用現有之方式會特別困難，4*4 的數字華容道就已經是 16 階層約 20 兆種的組合了，若到 5*5 的數字華容道，那將是 25 階層的組合，從我們找到 Nature 期刊中發現，可以用 DeepCubeA 的深度增強學習模式，目前還在學習研究中。
4. 前面所提的馬可夫決策過程是一種無記憶的決策過程，所進行的未來狀態判斷只與現有狀態相關，並不會與過去相關，在執行時會有與 Q-Learning 相似的狀態矩陣，因都是進行決策時可使用的方法，所以都能解決華容道的問題。
5. 若要解中國傳統華容道的拼圖，其邊框為 4*5，也就是說橫有四格、縱有五格，而且每一個方塊的大小也不一樣！要移動起來限制更多，但用我們程式來解，理論上是必有解的，只是跑出來的次數可能會多得嚇人，現在應該還不能找到最短步數解，若要找到最短步數解，該要使用深度增強學習（DRL）的方式才行。
6. 我們在實際落地操作，給小朋友玩的過程中，如果要給小朋友弄亂，再一個一個手動輸入至電腦才可以玩，會出現輸入過慢的問題，我們接下來希望做機電整合與無線傳輸或 AI 電腦視覺處理，讓它直接到電腦顯現，不僅可以節省時間，還能達到多人連線競賽的趣味性。
7. 我們還設計了六角形邊框的三角華容道，也就是每一個華容道可移動的單元不再是正方形了，而是三角形，且在六邊形的邊框內移動，共九個可移動的三角形與一個三角形空間，這樣不僅可以測試我們 AI 程式的功能性，還可增加小朋友試玩的趣味性，達到寓教於樂的目的。



（圖 3-10）六邊形創新華榮道構想圖

四、結論與應用

(一) 結論

本研究透過自建的增強學習之 Q-Learning 模式，建立 4*4 數字華容道的完整組合全解。依據實驗結果與比較分析可以發現，本研究所自建的 AI 模式對於傳統 BFS 的解法有更快的速度與延伸性。本研究透過訓練完成的增強學習模式，在 180 萬組以上的次數資料上，進行實際訓練，研究其成果能夠完全解決約十兆多種排列組合的任何一種可能。且在真實場合實際進行真人比較，本研究的 AI 模式下，具有較快的速度。

1. 依據 180 多萬種的資料訓練，4*4 的數字華容道中，建立起平均 30-60 秒的解題速度，並且能更運用於實際的真人競賽中。
2. 運用 Q-Learning 之增強學習技術，將原先無法解決的 4*4 數字華容道拼圖，實際解出。並在 3*3 的數字華容道中進行比對分析，發現平均解題速度上將能極大幅度提升。
3. 本研究所建立增強學習模式，針對 4*4 的拼圖進行 180 萬餘次訓練後，可提升解題速度，且可推廣至真人競賽，提升學生學習資訊程式、與運算思維的趣味性與教育性，提升本研究的優異效能與附加價值。

(二)未來展望與應用

本研究自建的增強學習系統之 AI 模式，在益智遊戲的領域中，具有可延伸與發展空間，並且在增加訓練階段後具有更快的速度，惟在面對更高階與更複雜的益智遊戲時。需研究完『深度增強學習（DRL)』後，才易寫出程式解決，因此對於未來研究的建議，有幾項發展的方向。

1. 本研究主要集中在 4*4 數字華容道上，若欲解中國傳統華容道，該應用本研究發展之技術，寫出可解的程式。
2. 4*4 的組合資料多達 20 多兆種，扣掉無解的情況也有 10 兆多種，但 4*4 的最複雜組合，其最精簡步數卻只有 80 步，如何找到最短步數解呢？希望能夠開發『深度增強學習系統（DRL)』的程式，達到秒殺解出的目的。

3. 本研究所建立的增強學習模式，對於最後組合排列的結果，是必須直接用手動輸入的，然後才能進行人機對奕，對於一般的新手玩家或小朋友，會覺得太慢也不夠方便，故開發出快速的輸入方式，也是後續發展的方向。因此本研究建議未來可以透過人工智慧的方式，對 4*4 的華容道，進行全圖影像的切割，切割分成 16 區塊，並且依據每個區塊的數字圖形，進行不同的電腦視覺(CV)判斷，整合匯入程式中，能在人類感知的範圍內提升輸入速度。
4. 本研究主要樣本多為 3*3 或 4*4 的正方型模式，為提升系統對於不同框架的測試，特設計六角形的邊框，對其內三角形數字格進行訓練，朝更多圖形的數字華容道進行分析解決。
5. 為提升 AI 學習與資訊能力，未來應發展數萬人的連線遊戲，透過更多人數進行估算、訓練，達到人工智慧普及的目的。

五、參考文獻

(一)中文部分

【書籍】

池内孝啓, 鈴木たかのり, 石本敦夫, 小坂健二郎, 真嘉比愛。(2016)。Python 函式庫語法範例字典。旗標出版社

湯曉鷗、陳玉琨。(2018)。人工智能基礎(高中版)。華東師範大學出版社

李開復、陳楸帆。(2021)。AI 2041: 預見 10 個未來新世界。遠見天下文化出版股份有限公司

【期刊/科展文章】

MSDN。人工智慧-增強式學習的簡介(2018)。2018/10。取自 https://learn.microsoft.com/zh-tw/archive/msdn-magazine/2018/october/artificially-intelligent-introduction-to-reinforcement-learning

網路部分：

【百科全書】

Wikipedia (2020)。廣度優先搜尋。取自 https://zh.wikipedia.org/zh-tw/%E5%B9%BF%E5%BA%A6%E4%BC%98%E5%85%88%E6%90%9C%E7%B4%A2。

Wikipedia (2020)。Q-Learning。取自 https://zh.wikipedia.org/zh-tw/Q%E5%AD%A6%E4%B9%A0。

【其他相關網頁】

Ithelp。[魔法陣系列] Deep Q Network (DQN) 之術式解析(2018)。2018/11。取自 https://ithelp.ithome.com.tw/articles/10208668

莫凡 PYTHON。什麼是 Q Leaning(2018)。2018/01。取自 https://mofanpy.com/tutorials/machine-learning/reinforcement-learning/intro-q-learning

(二)英文部分

【期刊文章】

Forest Agostinelli, Stephen McAleer, Alexander Shmakov, Pierre Baldi. (2019). Solving the Rubik's cube with deep reinforcement learning and search.《Nature machine intelligence》

Agichtein, Eugene & Lawrence, Steve & Gravano, Luis. (2004). Learning to find answers to questions on the Web.《ACM Trans》.

附錄

附錄一：4x4 Q-Learning 華容道執行程式

```
from random import randint,random
import copy
from time import perf_counter
class Game:
    def __init__(self,temp_puzzle,Len,answer,times,Qtable,path,probability):
        self.temp_puzzle=temp_puzzle
        self.Len=Len
        self.answer=answer
        self.times=times
        self.Qtable=Qtable
        self.path=path
        self.probability=probability
    def exchange(self,action,p):
        if action==0:
            Difference=-3
        elif action==1:
            Difference=-1
        elif action==2:
            Difference=3
        else:
            Difference=1
        x=p.index(8)
        blank=p[x]
        p[x]=p[x+Difference]
        p[x+Difference]=blank
        return p
    def temp_to_p(self,temp_puzzle,puzzle):
        for i in range(0,len(temp_puzzle)):
            p=i+5
            tp=5
            if i//3==1:
                p+=1
            elif i//3==2:
                p+=2
            if temp_puzzle[i]//3==1:
                tp+=1
            elif temp_puzzle[i]//3==2:
                tp+=2
            puzzle[p]=temp_puzzle[i]+tp
        return puzzle
    def reward(self):
        if temp_puzzle==answer:
            return 1
        return 0
    def nums(self,temp_puzzle):
        num=str(temp_puzzle[0])
        for i in range(1,len(temp_puzzle)):
            num+=str(temp_puzzle[i])
        return num
```

```
    def Class(self,num):
        ans=num
        for i in range(1,num):
            ans*=i
        return ans
    def boundary(self,p):
        wasd=[]
        if p.index(8)-3>=0:
            wasd.append(0)
        if p.index(8)%3!=0:
            wasd.append(1)
        if p.index(8)+3<=8:
            wasd.append(2)
        if p.index(8)%3!=2:
            wasd.append(3)
        return wasd
    def state(self,temp_puzzle):#找編號
        ans=[0,1,2,3,4,5,6,7,8]
        Ans=0
        for i in range(0,9):
            for j in range(0,len(ans)):
                if int(temp_puzzle[i])==ans[j]:
                    Ans+=j*Game.Class(8-i)
                    del(ans[j])
                    break
        return Ans//2
    def choose_action(self,Qtable):
        wasd=Game.boundary(temp_puzzle)
        mp=[]
        for i in range(0,len(wasd)):
            mp.append(Qtable[Game.state(temp_puzzle)][wasd[i]])
        Rand=random()
        Max=0
        if Rand>float(probability):
            for i in range(1,len(wasd)):
                if mp[Max]<mp[i]:
                    Max=i
                elif mp[Max]==mp[i]:
                    if randint(0,1):
                        Max=i
        else:
            if len(wasd)>=1:
                Max=randint(0,len(wasd)-1)
        return wasd[Max]
class Game_L:
    def __init__(self,Len,puzzle,answer,path,obstacle):
        self.Len=Len
        self.puzzle=puzzle
        self.answer=answer
        self.path=path
        self.obstacle=obstacle
```

```
def goal(self):
    Answer=[0,1,2,3,4,8,12]
    for i in range(0,len(Answer)):
        if puzzle.index(Answer[i])==Answer[i] and i==len(Answer)-1:
            return 'complete!'
        else:
            break
    for i in range(0,len(answer)):
        if puzzle.index(Answer[i])!=answer[i]:
            return [puzzle.index(Answer[i]),i]
    return 'complete!'
def exchange(self,a,b):
    c=puzzle[a]
    puzzle[a]=puzzle[b]
    puzzle[b]=c
def b_Obst(self,obstacle,value):
    if value not in obstacle:
        if value//4==obstacle[-1]//4:
            return 1
        if value%4 == obstacle[-1]%4:
            return 2
        return 3
    return 0
def g_Obst(self,obstacle,value):
    if value//4==obstacle//4:
        return 1
    if value%4 == obstacle%4:
        return 2
    return 0
def distance(self,a,b):
    return abs(a//4-b//4)+abs(a%4-b%4)
def b_wasd(self,b_goal,b,obstacle):
    wasd=[-1]*4
    if b-4>=0:
        b_w=Game_L.b_Obst(obstacle,b-4)
        if b_w%2:
            wasd[0]=Game_L.distance(b-4,b_goal)
    if b%4>0:
        b_a=Game_L.b_Obst(obstacle,b-1)
        if b_a>1:
            wasd[1]=Game_L.distance(b-1,b_goal)
    if b+4<=15:
        b_s=Game_L.b_Obst(obstacle,b+4)
        if b_s%2:
            wasd[2]=Game_L.distance(b+4,b_goal)
    if b%4<3:
        b_d=Game_L.b_Obst(obstacle,b+1)
        if b_d>1:
            wasd[3]=Game_L.distance(b+1,b_goal)
    return wasd
```

```
def choose_direction(self,wasd):
    min_=[]
    min_position=[]
    for i in range(0,4):
        if wasd[i]!=-1:
            if len(min_)!=0:
                if min_[0]>wasd[i]:
                    min_=[wasd[i]]
                    min_position=[i]
                elif min_[0]<wasd[i]:
                    continue
                else:
                    min_.append(wasd[i])
                    min_position.append(i)
            else:
                min_.append(wasd[i])
                min_position.append(i)
    return min_position
def goal_wasd(self,g,b,answer,Wasd):
    wasd=[-1]*4
    if answer//4<g//4:
        wasd[0]=Game_L.distance(g-4,b)
    if answer%4<g%4:
        wasd[1]=Game_L.distance(g-1,b)
    if answer//4>g//4:
        wasd[2]=Game_L.distance(g+4,b)
    if answer%4>g%4:
        wasd[3]=Game_L.distance(g+1,b)
    g_wasd=Game_L.choose_direction(wasd)
    if len(g_wasd)==1:
        return Wasd[g_wasd[0]]
    else:
        obst=Game_L.g_Obst(g,b)
        if obst:
            if obst==1:
                wasd[1]=-1
                wasd[3]=-1
            else:
                wasd[0]=-1
                wasd[2]=-1
            g_wasd=Game_L.choose_direction(wasd)
            if len(g_wasd)==1:
                return Wasd[g_wasd[0]]
    r=randint(0,len(g_wasd)-1)
    return Wasd[g_wasd[r]]
```

```
def moving_path(self,g_direction,g,obstacle):
    b=puzzle.index(15)
    wasd=[b-4,b-1,b+4,b+1]
    blank_wasd=Game_L.b_wasd(g_direction,b,obstacle)
    choose=Game_L.choose_direction(blank_wasd)
    r=randint(0,len(choose)-1)
    choose=choose[r]
    blank_direction=wasd[choose]
    Game_L.exchange(blank_direction,b)
    copy_puzzle=copy.deepcopy(puzzle)
    path.append(copy_puzzle)
    return blank_wasd[choose]
def swap(self,state):
    s=0
    if state%2==1:
        swap_path=[1,4,-1,4,1,-4,-4,-1,4,1,4,-1,-4,-4,1,4]
    else:
        swap_path=[4,1,-4,1,4,-1,-1,-4,1,4,1,-4,-1,-1,4,1]
    if 7>state>4:
        s=11
    elif 5>state>2:
        s=13
    elif state>6:
        s=14
    for i in range(s,len(swap_path)):
        b=puzzle.index(15)
        Game_L.exchange(b,b+swap_path[i])
        copy_puzzle=copy.deepcopy(puzzle)
        path.append(copy_puzzle)
def make_L(self,g):
    wasd=[g[0]-4,g[0]-1,g[0]+4,g[0]+1]
    g_direction=Game_L.goal_wasd(g[0],b,answer[g[1]],wasd)
    move=Game_L.moving_path(g_direction,g,obstacle)
    if move:
        pass
    else:
        Game_L.exchange(g[0],puzzle.index(15))
        copy_puzzle=copy.deepcopy(puzzle)
        path.append(copy_puzzle)
def shortcut(self,path):
    puzzle_path=[]
    while True:
        if len(path):
            path_0=path[0]
            puzzle_path.append(path_0)
            del(path[0])
            if path_0 in path:
                while True:
                    del(path[0])
                    if path_0 not in path:
                        break
        else:
            break
    return puzzle_path
```

```
def txt_input_list(txt):
    array=[]
    for line in txt.readlines():
        line = line.strip()
        line=line.split(' ')
        array.append(line)
    txt.close()
    return array
def printer(puzzle):
    print('-------------')
    for i in range(0,4):
        print('|',end='')
        for j in range(0,4):
            if puzzle[i*4+j]==15:
                print('  ',end='|')
            else:
                print('%02d'%(puzzle[i*4+j]+1),end='|')
        print('\n-------------')
    print()
Len=4
puzzle=input('puzzle:').split()
time_start=perf_counter()
for i in range(0,len(puzzle)):
    puzzle[i]=int(puzzle[i])-1
answer=[0,1,3,7,4,12,13]
Answer=[0,1,2,3,4,8,12]
copy_puzzle=copy.deepcopy(puzzle)
path=[copy_puzzle]
obstacle=[]
Game_L=Game_L(Len,puzzle,answer,path,obstacle)
while True:
    if puzzle[2]==15 and puzzle[3]==2 and puzzle[6]==3:
        Game_L.swap(1)
        answer=[0,1,2,3,4,12,13]
    elif puzzle[8]==15 and puzzle[9]==12 and puzzle[12]==8:
        Game_L.swap(2)
        answer=[0,1,2,3,4,8,12]
    elif puzzle[3]==2 and puzzle[7]==3:
        if puzzle[6]==15:
            Game_L.swap(3)
            answer=[0,1,2,3,4,12,13]
        elif puzzle[11]==15:
            Game_L.swap(5)
            answer=[0,1,2,3,4,12,13]
        else:
            Game_L.swap(7)
            answer=[0,1,2,3,4,12,13]
```

```
    elif puzzle[12]==8 and puzzle[13]==12:
        if puzzle[9]==15:
            Game_L.swap(4)
            answer=[0,1,2,3,4,8,12]
        elif puzzle[14]==15:
            Game_L.swap(6)
            answer=[0,1,2,3,4,8,12]
        else:
            Game_L.swap(8)
            answer=[0,1,2,3,4,8,12]
    else:
        g=Game_L.goal()
        if g=='complete!':
            puzzle_path=Game_L.shortcut(path)
            for i in range(0,len(puzzle_path)):
                printer(puzzle_path[i])
            break
        b=puzzle.index(15)
        obstacle=[]
        for i in range(0,g[1]+1):
            obstacle.append(puzzle.index(Answer[i]))
        Game_L.make_L(g)
#3*3
temp_puzzle=copy.deepcopy(puzzle)
for i in range(0,len(Answer)):
    temp_puzzle.remove(Answer[i])
for i in range(0,len(temp_puzzle)):
    if temp_puzzle[i]//4==1:
        temp_puzzle[i]-=5
    elif temp_puzzle[i]//4==2:
        temp_puzzle[i]-=6
    else:
        temp_puzzle[i]-=7
Len=3
times=len(puzzle_path)
answer=[0,1,2,3,4,5,6,7,8]
table=open('D:/AI華榮道/qtable.txt','r')
Qtable=txt_input_list(table)
table.close()
read_probability=open('D:/AI華榮道/p.txt','r')
probability=float(read_probability.read())
read_probability.close()
Game=Game(temp_puzzle,Len,answer,times,Qtable,path,probability)
while Game.reward()!=1:
    times+=1
    path.append(copy.deepcopy(temp_puzzle))
    action=Game.choose_action(Qtable)
    Game.exchange(action,temp_puzzle)
    puzzle=Game.temp_to_p(temp_puzzle,puzzle)
    printer(puzzle)
print('times:'+str(times))
time_end=perf_counter()
print('[{:.2f} s]'.format(time_end-time_start))
```

附錄二：4x4 Q-Learning 華容道訓練程式

```
from random import randint,random
import copy
class Game:
    def __init__(self,Len,All,answer,times,Qtable,path,probability):
        self.Len=Len
        self.All=All
        self.answer=answer
        self.times=times
        self.Qtable=Qtable
        self.path=path
        self.probability=probability
    def exchange(self,action,p):
        if action==0:
            Difference=-3
        elif action==1:
            Difference=-1
        elif action==2:
            Difference=3
        else:
            Difference=1
        x=p.index('8')
        blank=p[x]
        p[x]=p[x+Difference]
        p[x+Difference]=blank
        return p
    def find(self,p):
        if p in path:
            return 1
        return 0
    def boundary(self,p):
        wasd=[]
        if p.index('8')-3>=0:
            wasd.append(0)
        if p.index('8')%3!=0:
            wasd.append(1)
        if p.index('8')+3<=8:
            wasd.append(2)
        if p.index('8')%3!=2:
            wasd.append(3)
        i=0
        while i<len(wasd):
            temp=copy.deepcopy(p)
            temp=Game.exchange(wasd[i],temp)
            f=Game.find(temp)
            if f:
```

```
            if len(wasd)>1:
                wasd.remove(wasd[i])
                continue
        i+=1
    return wasd
def reward(self,puzzle):
    r={0:0.5,1:0.6,2:0.65,3:0.7,4:0.85,5:0.9,6:0.95,7:1}
    C=[[[0],[2],[6]],
       [[0,1],[0,3],[1,2],[3,6]],
       [[0,1,2],[0,3,6]],
       [[0,1,2,3],[0,1,3,6]],
       [[0,1,2,3,6]],
       [[0,1,2,3,4,5],[0,1,3,4,6,7],[0,1,2,3,6,7],[0,1,2,3,5,6]],
       [[0,1,2,3,4,5,6],[0,1,2,3,4,6,7]],
       [[0,1,2,3,4,5,6,7]]]
    for i in range(len(C)-1,-1,-1):
        for j in range(0,len(C[i])):
            for k in range(0,len(C[i][j])):
                if int(puzzle[C[i][j][k]])==C[i][j][k]:
                    if k==len(C[i][j])-1:
                        return r[i]
                else:
                    break
    return 0
def nums(self,temp_puzzle):
    num=str(temp_puzzle[0])
    for i in range(1,len(temp_puzzle)):
        num+=str(temp_puzzle[i])
    return num
def Class(self,num):
    ans=num
    if num>1:
        for i in range(2,num):
            ans*=i
    return ans//2
def state(self,temp_puzzle):
    L=len(temp_puzzle)-1
    Ans=0
    ans=[0,1,2,3,4,5,6,7,8]
    for i in range(0,9):
        s=ans.index(int(temp_puzzle[i]))
        Ans+=s*Game.Class(L-i)
        del(ans[s])
    return Ans
def choose_action(self,Qtable,M):
```

```
        wasd=Game.boundary(temp_puzzle)
        mp=[]
        for i in range(0,len(wasd)):
            mp.append(Qtable[Game.state(temp_puzzle)][wasd[i]])
        Rand=random()
        Max=0
        if Rand<float(probability) and M=='MQ':
            for i in range(1,len(wasd)):
                if mp[Max]<mp[i]:
                    Max=i
                elif mp[Max]==mp[i]:
                    r=randint(0,1)
                    if r:
                        Max=i
        else:
            if len(wasd)>=1:
                Max=randint(0,len(wasd)-1)
        return wasd[Max]
def txt_input_list(txt,condition):
    array=[]
    for line in txt.readlines():
        line = line.strip()
        if condition:
            line=line.split(' ')
        array.append(line)
    txt.close()
    return array
def list_input_txt(Qtable):
    qtable=open('D:/AI華榮道/qtable.txt','w')
    i=0
    while True:
        if i<len(Qtable):
            qtable.write(str(' '.join(Qtable[i]))+'\n')
            i+=1
        else:
            break
    qtable.close()
def read_arp(num):
    if num==1:
        read_a=open('D:/AI華榮道/a.txt','r')
        a=float(read_a.read())
        read_a.close()
        return a
    elif num==2:
        read_r=open('D:/AI華榮道/r.txt','r')
```

```
        r=float(read_r.read())
        read_r.close()
        return r
    else:
        read_probability=open('D:/AI華榮道/p.txt','r')
        probability=read_probability.read()
        read_probability.close()
        return probability
Time=int(input('times:'))
All=open('D:/AI華榮道/all.txt','r')
All=txt_input_list(All,0)
table=open('D:/AI華榮道/qtable.txt','r')
Qtable=txt_input_list(table,1)
table.close()
answer=[0,1,2,3,4,5,6,7,8]
a=read_arp(1)
r=read_arp(2)
probability=read_arp(3)
path=[]
Game=Game(3,All,answer,0,Qtable,path,probability)
for i in range(0,181440):
    print(i)
    puzzle=[]
    for j in range(0,9):
        puzzle.append(All[i][j])
    path=[]
    temp_puzzle=copy.deepcopy(puzzle)
    Qstate=Game.state(temp_puzzle)
    Qaction=Game.choose_action(Qtable,'Q')
    times=0
    while times<Time and Game.reward(puzzle)!=1:
        times+=1
        path.append(copy.deepcopy(temp_puzzle))
        Game.exchange(Qaction,temp_puzzle)
        MQstate=Game.state(temp_puzzle)#MaxQ[s]
        MQaction=Game.choose_action(Qtable,'MQ')#MaxQ[a]
        R=Game.reward(puzzle)
        Qtable[Qstate][Qaction]=str(float(Qtable[Qstate][Qaction])+a*(R+r*float(Qtable[MQstate][MQaction])-float(Qtable[Qstate][Qaction])))
        Qstate=MQstate
        Qaction=MQaction
```

附錄三：3x3 BFS 華容道執行程式

```
from random import random,sample
import sys,copy,math,time
class Game:
  def __init__(self,gameBoard,times,blank,path,Len):
    self.gameBoard=gameBoard
    self.times=times
    self.blank=blank
    self.Len=Len
  def Printer(self,Len,path):
    for i in range(0,Len):
      print('|',end='')
      for j in range(0,Len):
        if answer[i*Len+j]!='x':
          print('%02d'%answer[i*Len+j]+'|',end='')
        else:
          print(' '+str(answer[i*Len+j])+'|',end='')
      print()
    print('times:'+str(times))
  def exchange(self,blank,datum,change):
      blank=Game.gameBoard[datum+change]
      Game.gameBoard[datum+change]=Game.gameBoard[datum]
      Game.gameBoard[datum]=blank
  def bfsXlocation(self,temporarily,Len):
    for datum in range(0,Len**2):
      if temporarily[datum]=='x':
        return datum
  def gameBoardReady(self,answer,a,Len):
    while True:
      temporarily=sample(start,Len**2)
      for i0 in range(0,Len**2):
        for i1 in range(0,Len**2):
          if temporarily[i0] !='x' and temporarily[i1] !='x':
            if i0<i1 and temporarily[i0]>temporarily[i1]:
              a=a+1
      if a%2==0:
        for i2 in range(0,Len**2):
          answer[i2]=temporarily[i2]
        break
      else:
        a=0
        continue
  def finish(self,b,answer,TIME_s):
    TIME_e=time.time()
    for i in range(0,Len**2):
      if i!=Len**2-1:
        if Game.gameBoard[i]==i+1:
          b=b+1
      else:
        if Game.gameBoard[i]=='x':
          b-b+1
```

```
  if b==Len**2:
    print('試了'+str(times)+'次')
    print('算了'+str(TIME_e-TIME_s)+'秒')
    sys.exit(0)
def xLocation(self,answer,Len):
  for i in range(0,Len**2):
    if Game.gameBoard[i]=='x':
      datum=i
  return datum
def boundary(self,datum,player,times,Len,path):
  if player=='w':
    if datum-Len<0:
      path.remove(path[-1])
      return 0
    else:
      return Len-(Len*2)
  elif player=='a':
    if datum%Len==0:
      path.remove(path[-1])
      return 0
    else:
      return -1
  elif player=='s':
    if datum+Len>Len**2-1:
      path.remove(path[-1])
      return 0
    else:
      return Len
  elif player=='d':
    if datum%Len==Len-1:
      path.remove(path[-1])
      return 0
    else:
      return 1
  elif player=='p':
    path.remove(path[-1])
    if path[-1]=='w':
      path.remove(path[-1])
      return Len
    elif path[-1]=='a':
      path.remove(path[-1])
      return 1
    elif path[-1]=='s':
      path.remove(path[-1])
      return Len-(Len*2)
    elif path[-1]=='d':
      path.remove(path[-1])
      return -1
  else:
    print('errormessage')
```

```
    path.remove(path[-1])
    return 0
def bfsExchange(self,temporarily,blank,datum,change):
  blank=temporarily[datum+change]
  temporarily[datum+change]=temporarily[datum]
  temporarily[datum]=blank
  return temporarily
def Sort(self,find,All):
  i=0
  while i<len(All):
    if find<All[i]:
      break
    else:
      i=i+1
      continue
  All.insert(i,find)
  return All
def bfsnums(self,answer,Len):
  num=[]
  for i in range(0,Len**2):
    num.append(9)
  S=0
  for j in range(0,Len**2):
    if answer[j]=='x':
      num[j]=num[j]*(10**(Len**2-j-1))
    else:
      num[j]=answer[j]*(10**(Len**2-j-1))
  for k in range(0,Len**2):
    S=S+num[k]
  return S
def bfsnume(self,answer,Len):
  t=[]
  for e in range(Len**2-1,-1,-1):
    if answer//(10**e)==9:
      t.append('x')
    else:
      t.append(answer//(10**e))
    answer=answer%(10**e)
  return t
def binary_search(self,answer,Len,BFS,All):
  find=Game.bfsnums(answer,Len)
  c=1
  Lmax=len(All)-1
  Lmin=0
  l=Lmax//2
  while Lmin<l<Lmax:
    if All[l]==find:
      c=0
      break
    elif All[l]<find:
```

```
        Lmin=l
        l=(Lmax+l)//2+1
      elif All[l]>find:
        Lmax=l
        l=(l+Lmin)//2
    if c==1:
      Game.Sort(find,All)
      e=Game.bfsnume(find,Len)
      BFS.append(e)
  def bfsfinish(self,b,temporarily,TIME_s):
    TIME_e=time.time()
    for i in range(0,Len**2):
      if i!=Len**2-1:
        if temporarily[i]==i+1:
          b=b+1
      else:
        if temporarily[i]=='x':
          b=b+1
    if b==Len**2:
      print('試了'+str(times)+'次')
      print('算了'+str(TIME_e-TIME_s)+'秒')
      sys.exit(0)
Len=3
start=[]
answer=[]
path=[]
for i in range(0,Len**2):
  if i!=Len**2-1:
    start.append(i+1)
  else:
    start.append('x')
for i in range(0,Len**2):
  answer.append(0)
a=0
b=0
times=0
blank=0
Game=Game(answer,times,blank,path,Len)
a=int(input('1:手動代入圖片,2:電腦代入圖片\n'))
mode=int(input('1:手動執行,2:電腦執行\n'))
if a!=1:
  Game.gameBoardReady(answer,a,Len)
else:
  for i in range(0,9):
    answer.append(0)
    answer[i]=int(input('輸入第'+str(i+1)+'個的元素'))
    if answer[i]==9:
      answer[i]='x'
TIME_s=time.time()
```

```
if mode==1:
    Game.Printer(Len,path)
    Game.finish(b,answer,TIME_s)
    while True:
      player=str(input())
      path.append(player)
      times=times+1
      datum=Game.xLocation(answer,Len)
      change=Game.boundary(datum,player,times,Len,path)
      if change==0:
        print('error')
        times=times-1
        continue
      else:
        Game.exchange(blank,datum,change)
      Game.Printer(Len,path)
      Game.finish(b,answer,TIME_s)
else:
    Game.Printer(Len,path)
    Game.finish(b,answer,TIME_s)
    wasd=['w','a','s','d']
    bfs=[answer]
    BFS=[]
    All=[]
    while True:
      times=times+1
      for i in range(0,len(bfs)):
        t=copy.deepcopy(bfs[i])
        for j in range(0,4):
          temporarily=copy.deepcopy(t)
          path.append(wasd[j])
          change=Game.boundary(Game.bfsXlocation(temporarily,Len),wasd[j],times,Len,path)
          if change!=0:
            o=copy.deepcopy(Game.bfsExchange(temporarily,blank,Game.bfsXlocation(temporarily,Len),change))
            Game.binary_search(o,Len,BFS,All)
      print('level_'+str(times)+':'+str(len(BFS)))
      for k in range(0,len(BFS)):
        Game.bfsfinish(b,BFS[k],TIME_s)
      bfs=BFS
      BFS=[]
```

附錄四：用深度增強學習與搜尋方法去解魔術方塊

Solving the Rubik's cube with deep reinforcement learning and search

The **Rubik's** cube is a prototypical **combinatorial** puzzle that has a large state space with a single goal state. 2022.09.21	魔術方塊是一個典型組合的解謎遊戲，魔術方塊有龐大的狀態位置，與有著單一的目標狀態。 2022.09.21
The goal state is unlikely to be accessed using sequences of **randomly** generated moves, posing unique challengers machine learning. We solve the Rubik's cube with DeepCubeA, a deep reinforcement learning approach that learns how to solve increasingly difficult states in reverse from the goal state without any specific domain knowledge. 2022.09.22	訪問目標狀態是不太可能去用一連串隨機產生的移動，這造成機器學習獨特的挑戰者。 我們解魔術方塊用 DeepCubeA ，一個深度增強學習方法，在沒有任何特定領域知識下，從目標狀態反向去學習如何去解不斷增加困難的狀態。 2022.09.22
DeepCubeA solves 100% of all test configurations, finding a shortest path to the goal state 60.3% of the time. DeepCubeA generalizes to other combinatorial puzzles and is able to solve the 15 puzzle, 24 puzzle, 35 puzzle, 48 puzzle, Lights Out and Sokoban, finding a shortest path in the majority of verifiable cases. 2022.09.23	DeepCubeA 解決 100% 的全部測試佈局，找到最短路徑到目標狀態 60.3%的時間 ，DeepCubeA 概括其它的解謎且有能力去解 15 滑塊（4 x 4 數字華容道）、24 滑塊（5 x 5 數字華容道）、35 滑塊（6 x 6 數字華容道）、48 滑塊（7 x 7 數字華容道）、關燈遊戲和倉庫番遊戲，找到最短路徑在大部分可證實的實例。 2022.09.23
The Rubik's cube is a classic combinatorial puzzle that poses unique and interesting challenges for artificial intelligence and machine learning. 2022.09.24	魔術方塊是經典的組合數學解謎遊戲,而它給予人工智慧與機器學習造成獨特且有趣的挑戰。 2022.09.24
Although the state space is exceptionally large (4.3x 10^{19}different states), there is only one goal state. 2022.09.25	雖然魔術方塊狀態位置是異常龐大（4.3x 10^{19} 不同狀態），但它只有一個目標狀態。 2022.09.25
Furthermore, the Rubik's cube is a single-player game and a sequence of random moves, no matter how long, is unlikely to end in the goal state.	此外，魔術方塊是一個單一玩家遊戲，魔術方塊經一系列的隨機移動，不管時間長度多久，是難以結束在目標狀態。 2022.09.26

2022.09.26	
Developing machine learning algorithms to deal with this property of the Rubik's cube might provide insights into learning to solve planning problems with large state spaces. 2022.09.27	開發機器學習演算法去分出這個的特性，在龐大的狀態位置，可能提供洞察魔術方塊去解規劃的問題。 2022.09.27
Although machine learning methods have previously been applied to the Rubik's cube, these methods have either failed to reliably solve the cube or have had to rely on specific domain knowledges. 2022.09.28	雖然機器學習方法有先前地運用對於魔術方塊，但這些方法不是未能做到能可靠的解魔術方塊，就是依賴特定領域的知識。 2022.09.28
Outside of machine learning methods, methods based on pattern databases (PDBs) have been effective at solving puzzles such as the Rubik's cube, the 15 puzzle and the 24 puzzle", but these methods can be memory intensive and puzzle specific. 2022.09.29	在機器學習方法之外，方法基於在樣本數據庫（PDBs）有有效的在解解謎遊戲，如魔術方塊、15 滑塊（4 x 4 數字華容道）和 24 滑塊（5 x 5 數字華容道），但這些方法會在密集的記憶和特定的解謎遊戲。 2022.09.29
More broadly, a major goal in artificial intelligence is to create algorithms that are able to learn how to master various environments without relying on domain specific human knowledge. The classical 3x3x3 Rubik's cube is only one representative of a larger family of possible environments that broadly share the characteristics described above, including (1) cubes with longer edges or higher dimension (for example, 4x4x4 or 2x2x2x2), (2) sliding tile puzzles (for example the 15 puzzle, 24 puzzle, 35 puzzle and 48 puzzle), (3) Lights Out and (4) Sokoban. As the size and dimensions are increased, the complexity of the underlying combinatorial problems rapidly increases. For example, while finding an optimal solution to the 15 puzzle takes less than a second on a modern-day desktop, finding an optimal solution to the 24 puzzle can take days, and finding an optimal	更多概括地，一個重大的目標在人工智慧是去創建演算法，它能去學習如何去掌控不同樣式的環境，在沒有依賴領域特定的人類知識。 經典的 3x3x3 魔術方塊只有一個代理人，在一個大的家族可能的環境，大體的分費特徵描繪在上面，包含(1)正立方體長的邊，或更高維度（範例：4x4x4 或 2x2x2），(2)滑動圖塊解謎遊戲（範例：15 滑塊，24 滑塊，35 滑塊，48 滑塊），(3)關燈遊戲，(4)倉庫番。像尺寸和維度增大，而複雜性在深層的組合數學問題訊速增大。舉例來說，在找到最佳的解對於 15 滑塊，花費至少超過一秒，在現代日子的電腦，找到最佳的解對於 24 滑塊可以花費一天，而找到最佳解對於 35 滑塊通常是難以解決。不只前面提到的解謎遊戲相關的數學遊戲，還可能去運用去測試規劃的演算法，以及評估如何是一個好的機器學習，接近可能概括不同的環境。此外，因為操作的魔術方塊和其它組合的解謎

solution to the 35 puzzle is generally intractable". Not only are the aforementioned puzzles relevant as mathematical games, but they can also be used to test planning algorithms and to assess how well a machine learning approach may generalize to different environments. Furthermore, because the operation of the Rubik's cube and other combinatorial puzzles are deeply rooted in group theory, these puzzles also raise broader questions about the application of machine learning methods to complex symbolic systems, including mathematics. In short, for all these reasons, the Rubik's cube poses interesting challenges for machine learning. 2022.09.30	遊戲是非常的在組合理論根深蒂固，而這些解謎遊戲總是導致廣泛的問題，在於應用的機器學習方法，對複雜象徵性的系統，包括數學。在短時暫的，對於全部這些原因，魔術方塊提出有趣的挑戰給機器學習。 2022.09.30
To address these challenges, we have developed DeepCubeA,which combines deep learning with classical reinforcement learning (approximate value iteration) and path finding methods (weighted A* search7.). DeepCubeA is able to solve combinatorial puzzles such as the Rubiks cube, 15 puzzle, 24 puzzle, 35 puzzle, 48 puzzle, Lights Out and Sokoban (Fig. 1). DeepCubeA works by using approximate value iteration to train a deep neural network (DNN) to approximate a function that outputs the cost to reach the goal (also known as the cost-to-go function). Given that random play is unlikely to end in the goal state, DeepCubeA trains on states obtained by starting from the goal state and randomly taking moves in reverse. After training, the learned cost-to-go function is used as a heuristic to solve the puzzles using a weighted A* search. 2022.10.01	為應付這些挑戰，我們已經開發 DeepCubeA，他兼具深度學習和傳統增強學習一起（大致的值迭帶)一起，和路徑找到的方法（加權重 A * search），DeepCube 是可以去解組合解謎遊戲，像是魔術方塊，15 滑塊，24 滑塊，35 滑塊，48 滑塊，關燈遊戲和倉庫番遊戲。DeepCubeA 運作是利用大概的值迭帶,去訓練一個深度神經網路（DNN）去接近一個函數，產出一個價值去達到目標（此外被稱為 cost-to-go function ）。做隨機的活動，是不太可能去結束目標狀態。DeepCubeA 行列在狀態存在，從目標狀態和做隨機移動移動返向被開始。在訓練之後，這個學習 cost-to-go function 在使用像一個搜索式的，去解解謎遊戲，利用一個有利的 A*search。

DeepCubeA builds on DeepCube, a deep reinforcement learning algorithm that solves the Rubiks cube using a policy and value function combined with Monte Carlo tree search (MCTS).MCTS, combined with a policy and value function, is also used by AlphaZero, which learns to beat the best existing programs in chess, Go and shogi. In practice, we find that, for combinatorial puzzles, MCTS has relatively long runtimes and often produces solutions many moves longer than the length of a shortest path. In contrast, DeepCubeA finds a shortest path to the goal for puzzles for which a shortest path is computationally verifiable: 60.3% of the time for the Rubik's cube and over 90% of the time for the 15 puzzle, 24 puzzle and Lights Out. 2022.10.02	DeepCubeA 建立在 DeepCube，一個深度增強演算法，它可以解魔術方塊利用一個策略和價值函數結合與蒙特卡洛樹搜尋法（MCTS）一起用。MCTS，結合與一個策略與價值函數一起，此外被 AIphaZero 運用，它學習那，對於組合解謎遊戲，MCTS 有相對較長的執行時間和常產生解許多移動較長的時間，在一個最短路徑。在對比，DeepCubeA 找到一個最短路徑到目標狀態對於解謎遊戲，它對於最短路徑是數學組合可證實的：60.3%的時間對於魔術方塊和超過 90% 的時間對於 15 滑塊, 24 滑塊 and 關燈遊戲。 2022.10.02
Deep approximate value iteration Value iteration" is a dynamic programming algorithm that iteratively improves a cost-to-go function *J*. In traditional value iteration, J takes the form of a lookup table where the cost-to-go J(s) is stored in a table for all possible states s. However, this lookup table representation becomes infeasible for combinatorial puzzles with large state spaces like the Rubik's cube. Therefore we turn to approximate value iteration'", where J is represented by a parameterized function implemented by a DNN. The DNN is trained to minimize the mean squared error between its estimation of the cost-to-go of state s, J(s), and the updated cost-to-go estimation J'(s): $J'(s)= \min_a (g^a (s, A(s, a)) + J(A(s, a)))$ (1) 2022.10.03	深度大概值迭代 價值迭帶是一個不斷變化的程式演算法，它迭帶改善一個值函數 J。在傳統的值函數迭帶，J 形成表的查閱表格，值函數 J(s)儲存在一個表，對所有可能的狀態 s。然而，這個查閱表展現變得可行性，對於組合數學的解謎遊戲，在龐大的狀態空間，向魔術方塊。此外，我們轉向近似值的迭帶，而 J 是代表被一個參數函式實施，在一個深度神經網路下。深度神經網路是訓練去減到最小平均的方塊圖案差錯，在它估計的值函數狀態 s、J(s)，以即使升級價函數評估 J'(s): $J'(s)= \min_a (g^a (s, A(s, a)) + J(A(s, a)))$ (1) 2022.10.03

【評語】190017

1. 本作品以Q-Learning建立4x4數字華容道的完整組合全解，並比較分析與傳統 BFS 的解法在速度上與延伸性上的差異。

2. Q-Learning 是提出很就的技術，建議未來可嘗試不同的深度學習技術，例如 Deep Q-Learning，來增進演算過程的效能。