象棋人工智慧之實作

指導教授：林順喜教授
學生：劉宇翔

壹、摘要

人工智慧（Artificial Intelligence, AI）亦稱機器智慧，主要是指由人工製造出來的系統所表現出來的智慧。AI的核心問題包括推理、知識、規劃、學習、交流、感知、移動和操作物體的能力等等。人工智慧的定義可以分為兩部分，即「人工」和「智慧」。「人工」比較好理解，即為人類自製的東西。關於什麼是「智慧」，就難以定義。因為涉及到其它諸如意識、自我、心靈，等等問題。[註一]
而本研究主要是程式的地方從零開始，由根基慢慢打造，目標希望能開發一個象棋的人工智慧程式，符合人類的思維模式，且達到能和人互相對下的實力，同時製作一使用者圖型界面(GUI)。

貳、研究動機與背景

象棋是個歷史悠久的棋類，小時候就時常接觸，看見強者擁有自己的套路，就覺得很厲害，自從電子裝備普及後，裡面通常都會有象棋小遊戲，分為初階高階，當無聊時就會玩一下，覺得很有趣。現在希望能透過電腦的智慧，來呈現自己的想法，和對於象棋的理解。

參、相關文獻探討

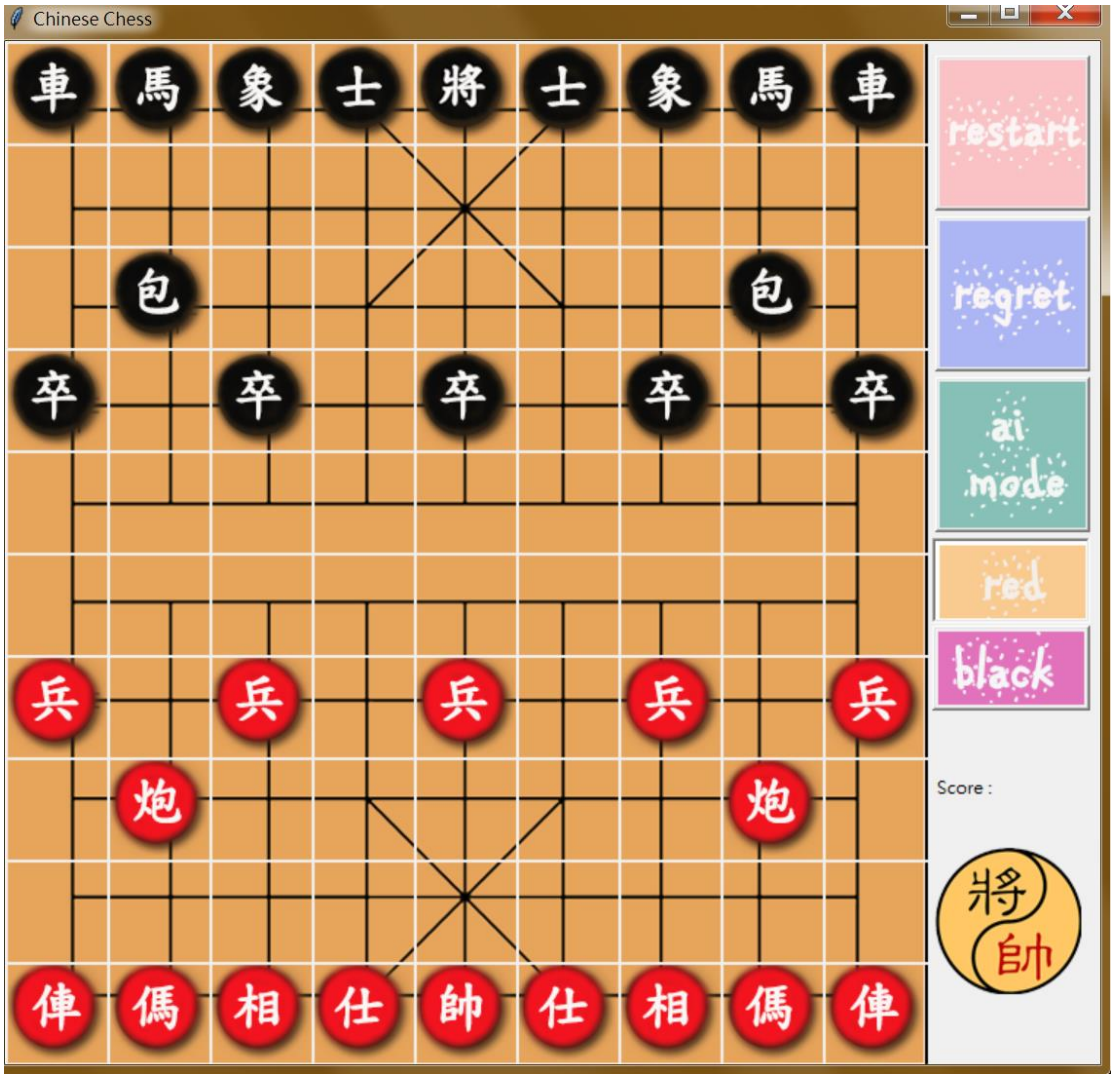
最早的電腦程式的象棋遊戲是1986年的洛斯阿拉莫斯象棋。由於象棋與西洋棋許多相似之處，再加上電腦西洋棋發展較為成熟，電腦象棋軟體設計的架構跟方法大致上都是參考電腦西洋棋。[註二]
實作分為幾個重要部分
1.棋局的資料型態：
如何儲存棋局是一大學問。資料型態的部分也要考慮到象棋的規則，包括走子、吃子等等合法步數。
以象棋來說，直覺的儲存方式是使用二維陣列來儲存，也就是將90格拆成10乘以9，可以不經思考的馬上得知棋子的位置，雖然非常的直覺，但是有一個缺點，即難以快速的產生合法步數。以車為例子，使用二維陣列來儲存的話，電腦必須使用for迴圈一一將每個位置做檢查，對於電腦的運算，要花費相當多的時間和資源。另一種儲存方式是使用一維陣列，將原本10乘以9改為256。另外也可使用BitBoard的方式儲存，當要產生合法步數時，能夠利用邏輯運算，e.g. AND和OR，電腦對於邏輯運算比循序計算更加的快速，產生合法步數在整個AI的實作中，佔了極重要的位置，因為在搜索的過程中，需要大量的窮舉每一子的合法步數，如果能夠減短產生所花費的時間，將會大大縮短總時間，在相同的時間裡，即能搜索到更深層，不過BitBoard適合64個格子的棋盤，e.g.

西洋棋，若要在象棋中使用BitBoard，還需要多加設計。
2.審局函數：
審局函數是整個AI的核心，強的審局函數幾乎是等於強的AI程式，會讀取盤面後給於盤面一個分數。一般會參考
(1)子的分數，給予每一顆子分數，只要存在於版面，就能獲得分數
(2)子的靈活性，表示這個子的活動範圍，因為能到10個地方的車一定比只能到2個地方的車還要強。
(3)子的位置，當子佔據某些點時，會特別強，例如：河頭馬、臥槽馬等等，這些都會有相對的分數加成
(3)子的攻擊，當可以吃到對方的子時，且對方的子沒有被「保護」，也能夠獲得額外的分數。
(4)子的保護，當子力被保護時，會獲得分數，且當對方攻擊此子時，沒有辦法獲得攻擊的分數。
3.搜索演算法：
Min-max的搜索演算法，是根據分數來選擇，奇數層會選擇最高分，偶數層會選擇最低分，因為分數主要都採用同一方的。
主要的概念在於展開所有的盤面，也就是去模擬之後都走步，再從這些走步中去挑選一個最佳走步。
Alpha-beta cut，理念和Min-max相同，也是在搜索最佳走步，只是改良其方法，因為當展開所有盤面時，如果層數很多，即搜索深度很深，會花法大量的時間，所以Alpha-beta cut是設定上下限，如果超過或低於，就不會在繼續搜索下去，而它所搜索到得最佳走步，必定和min-max 相同。
水平線效應：因為象棋屬於吃子類型的遊戲，子會從盤面上消失，假如設定搜索深度為4，有可能再葉節點出線車吃馬，但下一步車反被包吃，因為程式只看見車吃馬的步，認為這是最佳的走步，因而沒考慮到後面的走步。
寧靜搜索：解決水平線效應，當到達要搜索的深度時，會多判斷盤面是否寧靜，即無吃子步，若寧靜則停止搜索，反之，再繼續搜索。
審局函數與搜索演算法之間的關係：
兩者其實是相輔相成，一般的審局函數難以達到完美，因此要靠搜索演算法去彌補審局函數不準的因素，也可以換句話說，搜索很搜索到底層，要靠審局函數來判定何為最佳走步。

肆、開發工具與軟/硬體需求
開發工具：Python
軟體需求：IDLE(Python 編譯器)
硬體需求：記憶體越多，處理器越好，則越佳

伍、系統結構與操作流程

1.新棋局
2.讀取當前盤面
3.AI 模式(審局函數、搜索演算法)
4.下棋
GUI的設計：視窗是使用python 中的tkinter
棋盤設計為產生90個canvas，再載入圖片
盤面上點擊要移動的子，再點選要選擇的位置，移動後會有痕跡，方便觀看。



操作程式：
右邊的三個button分別是restart（重新開始）、regret(悔棋)、ai mode(AI步)
而下方則是誰先手(預設為紅方)，點選再按restart即可
關於悔棋實做的是採用stack來存放每一走步，當按下button後，會從stack中pop出來。
最下方的score代表紅方和黑方的差距，正數代表紅領先，負數代表黑方領先。

陸、系統功能之分項說明

棋盤的資料型態

在期中時，資料結構是採取二維陣列[10][9]，用陣列存放的值來得知盤面資訊，

0	1	2	3	4	5	6	7
空格	將	士	象	馬	車	包	卒

-1	-2	-3	-4	-5	-6	-7
帥	士	相	馬	車	包	兵

改良後，是採用陣列大小為90的一維陣列，儲存的值不變。

再產生走步時更加快速

目前的版本則是採用陣列大小為256。使用256而不用90是因為新增了牆壁，牆壁的優點在於產生走步時少了一個判斷，不需要判斷棋子是否跑到別的地方，牆壁的值設定為8。

```
b.boardinfo = [ 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8,
                8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8,
                8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8,
                8, 8, 8, 5, 4, 3, 2, 1, 2, 3, 4 ,5, 8, 8, 8, 8,
                8, 8, 8, 0, 0, 0, 0, 0, 0, 0, 0, 0, 8, 8, 8, 8,
                8, 8, 8, 0, 6, 0, 0, 0, 0, 0, 6, 0, 8, 8, 8, 8,
                8, 8, 8, 7, 0, 7, 0, 7, 0, 7, 0, 7, 8, 8, 8, 8,
                8, 8, 8, 0, 0, 0, 0, 0, 0, 0, 0, 0, 8, 8, 8, 8,
                8, 8, 8, 0, 0, 0, 0, 0, 0, 0, 0, 0, 8, 8, 8, 8,
                8, 8, 8,-7, 0,-7, 0,-7, 0,-7, 0,-7, 8, 8, 8, 8,
                8, 8, 8, 0,-6, 0, 0, 0, 0, 0,-6, 0, 8, 8, 8, 8,
                8, 8, 8, 0, 0, 0, 0, 0, 0, 0, 0, 0, 8, 8, 8, 8,
                8, 8, 8,-5,-4,-3,-2,-1,-2,-3,-4 ,-5, 8, 8, 8, 8,
                8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8,
                8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8,
                8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8]
```

走步產生：

每一種棋子都會有自己的走步法陣列，例如：馬能走8個方位，馬的走步陣列為

```
m_INC = [-18, 14, -33, -31, -14, 18, 31, 33]
```

所以只需要將馬的原位置各別加上這些值，再去判斷該位置是否是空格，或敵方的子，來確定其為合法步。

審局函數

在期中時，有考量的部分，包括子的分數、靈活度、攻擊與保護。

目前則是加入位置分後，再和子的分數合併。

位置分是7個大小為256的陣列，在每個地方儲存該位置的分數加上子力分。

現階段不考慮靈活度，是因為位置太重要，若說每個位置的差為1，那麼靈活度相當於0.1，又花費成本高，要對子再做一次展開，不過未來還是會納入考量。

另外攻擊與保護，因為有寧靜搜索的關係，所以不太需要。

以帥為例子：

```
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0,0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0,8888,8888,8888, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0,8888,8888,8888, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0,8888,8888,8888, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
```

搜索演算法：

採用alpha-beta-cut，搭配寧靜搜索，目前的搜索深度為4，寧靜搜索為8，另外還有搭配兩個提升效能的方法

(1)incremental update

原本都是在葉節點計算盤面分數，現在改為每動一步就將其變量傳到下一層，這樣和最後一次算相比，能縮短時間。

(2)大子優先考量

因為象棋時常移動大子，即車、馬、包，相對的將、士、象，較少移動，所以如果大子能夠在樹的左邊，就能夠發生更多的剪枝。原本是從左上開始展開，目前使採用先記錄大子位置，再進行展開。

柒、各組員之貢獻

全

捌、後續研究/實作方向

人工智慧智慧主要包含搜索演算法和審局函數，若搜索速度夠快，可以搜索到最底層，那就不需要審局函數，對於人類的思維，就代表是能洞察未來到棋局結束，相反的，若審局函數相當準確，也不需要搜索演算法，就能對對每盤棋的優劣給予評價。
以目前來說，要增加搜索的深度，可能要改變資料結構，另外還有MTD、PVS等加速演算法。
而審局函數的部分，如同上面所說，靈活度要再加入，但權重很低。之後想使用類神經網路來訓練，透過大量資料的訓練，來自動調整。最後還有攻擊模式和防禦模式，差別在於防禦是能換子就換子
GUI的部分，希望能讓動畫更加的流暢。

玖、參考文獻

註一
https://zh.wikipedia.org/wiki/%E7%94%B5%E8%84%91%E8%B1%A1%E6%A3%8B
註二
https://zh.wikipedia.org/wiki/%E4%BA%BA%E5%B7%A5%E6%99%BA%E8%83%BD
人工智慧-上課講義