五子棋人機對戰實作
–評估函數、Minimax對局分析與Alpha–Beta剪枝

製作人：劉蕃熙
時間：112年2月~3月

source code of project：https://github.com/FelicityTomato/gomoku
我的學習歷程檔案：https://felicitytomato.github.io/MLP/

摘要

五子棋是經典易學的棋類遊戲。在這份報告中，我利用python來撰寫程式。從雙人對戰的評判程式著手，擴展至人機對戰，並以增加電腦的棋力與減少運算時間為目標。

在最開始，我使用「評估函數」來評判哪個位置是最佳選擇。而後我考慮到評估函數中的參數是否有最佳值。因此我讓不同評估函數參數的程式互相比賽，找出最佳解，並討論結果的意義和當前算法會出現的問題。同時我發現，程式運行的速度不盡理想，因此有了第二版的評估函數，減少運算量以增進速度，並分析其不同評估函數優缺與差異。

而後，我使用Minimax對決搜尋與Alpha–Beta剪枝以增加電腦棋力。在實作後發生正確性下降與運算速度過慢等問題。我在思考與搜尋資料後，發現評估函數增加特判「死四」、「活三」後，會大幅增加正確性和優化運算速度，到平均5秒內可搜尋完四層。

我統整查找資料的結果後，發現了其他優化方法，像是Zobrist hash 算法、啟發式評估函數和迫著搜尋法。在未來也會嘗試並分析其優缺與效率。

目錄

壹、動機

利用程式完成一個人機對戰的遊戲一直是我的目標之一。在之前社團的經驗中，我也帶其他社員們寫了不少像1A2B、OOXX、wordle的遊戲。然而，都只停留在人人對戰並利用程式判斷輸贏的程度。此外，我在高中這幾年雖然一直在學習演算法，但大多數都是跟程式競賽相關的。因此，我在查找資料後，發現要利用「評估函數」和Minimax對局搜尋來使電腦可以與人對戰，這是我之前沒接觸過的領域，但也同時是我喜歡的演算法。所以我對其很感興趣，在研究後也嘗試自己實作出來。此外，我為了能增加電腦的棋力，想到可以random不同的參數，讓程式自己對打，藉此優化評估函數，找出最佳值。而這份報告就記錄下了，這段時間的想法、過程、和遭遇的困難。

貳、目的

一. 用python實作五子棋遊戲，讓雙人對戰並評判輸贏
二. 利用評估函數使電腦能與人對戰
三. 最佳化評估函數
四. Minimax對局搜尋+Alpha–Beta剪枝

參、研究過程及方法

一. 用python實作五子棋遊戲，讓雙人對戰並評判輸贏

5. 想法：

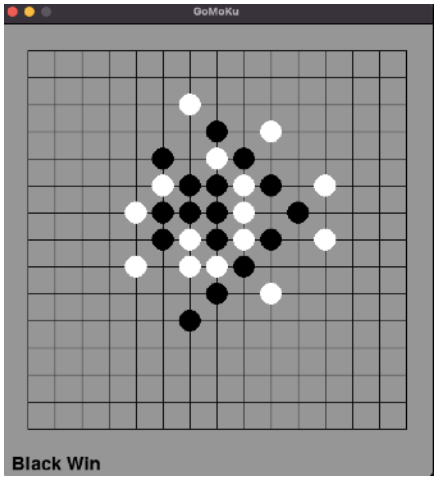
在任一方下完一個棋子後，判斷此盤面是否有人贏
條件：在四個方向（橫、直、左斜、右斜）中有任一方向有連續五個相同顏色的棋子即為勝利

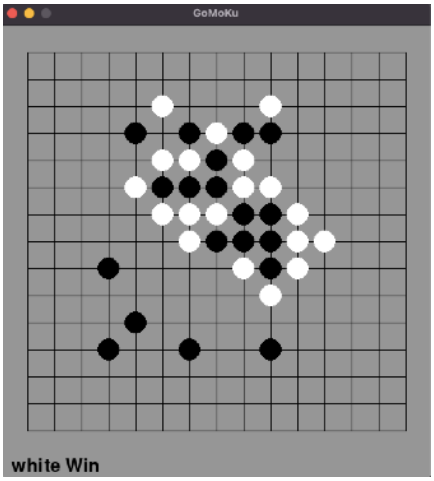
6. 實作：

(1) 最初版本：
枚舉每個棋盤位置，對每個位置分別找每個方向是否勝利，並且注意邊界條件
(2) 改進：
建立一個tuple p=(1,0,0,1,1,1,1,–1)，兩兩分別代表四個方向，並在每次都判斷所有邊界條件，這樣就可以包在一個迴圈內完成，不需要寫四次相次的東西。

```python
def judge(grid): # 0:draw 1:black 2:white
    p = (1,0,0,1,1,1,1,-1)
    for i in range(ROWS):
        for j in range(COLS):
            for t in range(4):
                for k in range(5):
                    if grid[i][j]!=0 \
                        and i+k*p[t*2]<ROWS and i+k*p[t*2]>=0 and j+k*p[t*2+1]<COLS and j+k*p[t*2+1]>=0 \
                        and grid[i+k*p[t*2]][j+k*p[t*2+1]]==grid[i][j]:
                        if k==4:
                            return grid[i][j]
                    else:
                        break
```

3. 結果：





二. 人機對戰–評估函數

1. 想法：

對於每一個棋盤狀態，都可以算出其分數，我將其一子、兩子、三子、四子、五子、連線分別給一個分數，並分成「進攻」和「防守」分。每次輪到電腦下時，就枚舉每個空白點下黑棋，找分數最高的一組下棋。

2. 實作：

將進攻設為plus = (0,1,3,10,30,100)，防守設為minus = (0,2,5,9,40,99)，分別代表零至五個棋子相連所要的加減分。

每次輪到電腦下時，就枚舉每個空白點下黑棋，並計算出其分數。選擇分數最大值的位置。若最後有超過一組位置是最大值，則random決定下的位置。

```
def evaluation(grid,turn):
    plus = (0,1,3,10,30,100)
    minus = (0,2,5,9,40,99)
    p = (1,0,0,1,1,1,1,-1)
    value = 0
    for i in range(ROWS):
        for j in range(COLS):
            for t in range(4):
                if i-p[t*2]<0 or j-p[t*2+1]<0 \
                    or i-p[t*2]>=ROWS or j-p[t*2+1]>=COLS \
                    or grid[i-p[t*2]][j-p[t*2+1]]!=grid[i][j]:
                    for k in range(5):
                        if grid[i][j]!=0 \
                            and i+k*p[t*2]<ROWS and i+k*p[t*2]>=0 and j+k*p[t*2+1]<COLS and j+k*p[t*2+1]>=0 \
                            and grid[i+k*p[t*2]][j+k*p[t*2+1]]==grid[i][j]:
                            if k==4:
                                if grid[i][j]==turn: value+=plus[k+1]
                                else: value-=minus[k+1]
                                break
                        else:
                            if grid[i][j]==turn: value+=plus[k]
                            else: value-=minus[k]
                            break
```

3. 改進：

我發現這樣寫出來的程式只會不斷進攻，不會防守，因為他只下對自己最好的位置，而不會防守，也就是下對對方最好的位置，來避免對方下一步下這個位置。因此我決定改為枚舉完黑棋後再枚舉白棋，若白棋得分高於黑棋，則決定此步要防守，否則選擇進攻。

4. 困難與討論：

(1) 如何不重複計算每一組連續棋子？

我在每次沒舉到每個棋子時，都考慮他是不是這一組的「第一個」。也就是說，我會往反方向看前一個位置下的是什麼，如果是同色棋子，則代表這組算過了，就跳過；反之計算共有幾個棋子。

(2) 怎麼確定評估函數的數值是最佳的？

我一開始參考「程式人雜誌」中評估函數的數值進行，結果也符合預期。但我嘗試更動裡面的數值後，就有不一樣的結果。這讓我開始思考，是否有一組數值是最好的，而這也使我開始做下一個主題：最佳化評估函數

5. 結果：

使用評估函數進行五子棋人機對戰實戰影片：
https://www.youtube.com/watch?v=3RVu02ASMC8

三. 最佳化評估函數

1. 想法：

評估函數內的數值會影響程式運行的結果，而現在的版本中的數值是參考「程式人雜誌」中的評估函數，為何它是最好的？有沒有保證一組最好的數值？

2. 實作：

我的做法是讓使用不同評估函數參數的程式互相比賽，贏得當後手，連續贏一定次數就算贏家。

第一版的程式是雙方對戰，連續贏50次就結束。

第二版則考慮到我的程式是具有隨機性的，為避免有很好的數據因為一次隨機到的位置而被刪除，我將每一組評估函數都對打5次，取贏3次以上對贏方，連續贏50次者算贏家。

此外，為了避免有評估函數重複出現，我將每組計算過的函數都做rolling hash後丟入set，並在每次rand後檢查它在set中是否存在，若存在則重新random到不存在後再繼續。

3. 數據分析：

所有數據：https://github.com/FelicityTomato/gomoku

因為所有數據共有約10^{16}種可能，且一筆平均要跑30秒，故不可能全部跑完，我將有贏超過10次的數據拿出來分析，尋找它們的共通性。

第一版：

攻擊2~5子/防守2~5子

[7, 19, 36, 68] [49, 58, 70, 93]
[3, 8, 11, 51] [32, 44, 86, 96]
[6, 8, 34, 38] [17, 30, 69, 100]
[5, 19, 38, 39] [5, 23, 34, 65]
[10, 43, 57, 62] [12, 24, 44, 58]
[11, 15, 35, 51] [33, 47, 60, 65]（22次）

第二版：

[7, 24, 54, 69] [24, 28, 55, 62]
[1, 4, 15, 51] [54, 56, 69, 74]
[11, 26, 39, 77] [5, 35, 38, 81]
[7, 29, 33, 41] [15, 18, 26, 35]
[7, 22, 39, 54] [7, 37, 45, 95]
[3, 23, 56, 69] [8, 62, 93, 94]
[1, 14, 45, 50] [7, 58, 61, 87]
[7, 12, 33, 50] [40, 43, 70, 71]
[1, 8, 17, 47] [1, 20, 68, 94]（50次）

4. 結果：

從上述幾筆可以發現幾個性質：

1. 數值的絕對大小不重要，重要的是相對比例。
2. 大部分數據中，連續兩個數值間差異會超過兩倍
3. 防守五子多大於攻擊五子
4. 防守五子絕對大於攻擊四子

最後找到最好的一組評估函數為：

攻擊：[1, 1, 8, 17, 47]
防守：[1, 1, 20, 68, 94]

5. 困難與討論：

雖然最後有做出連續贏50次的評估函數值，但我認為這些數據還有一定的問題存在，並分析問題的來由：

(1) 大部分的數據都可以看出主要是以防守為主，像是最後找到最好的一組就可以發現，攻擊5子的分數甚至比防守4子低。也就是說當再下一個棋子電腦就可以贏的時候，它可能會選擇放棄進攻而防守對方的4子。

這不符合常理狀況我認為可能是因爲我都將勝利的組合放到後手，因此對讀防手的分數較高。

(2) 是否不存在「最佳參數」的組合？

因我的程式含有隨機因素，且評估函數的值有些些微差異可能不會對輸贏有影響，這導致有可能評估函數的值會形成「環」，而沒有唯一的最佳組合，而只存在一些較佳的參數。我為此有讓其對打多次，也盡量避免random影響結果，但也無法保證其正確性。

(3) 對戰速度是否可加快？

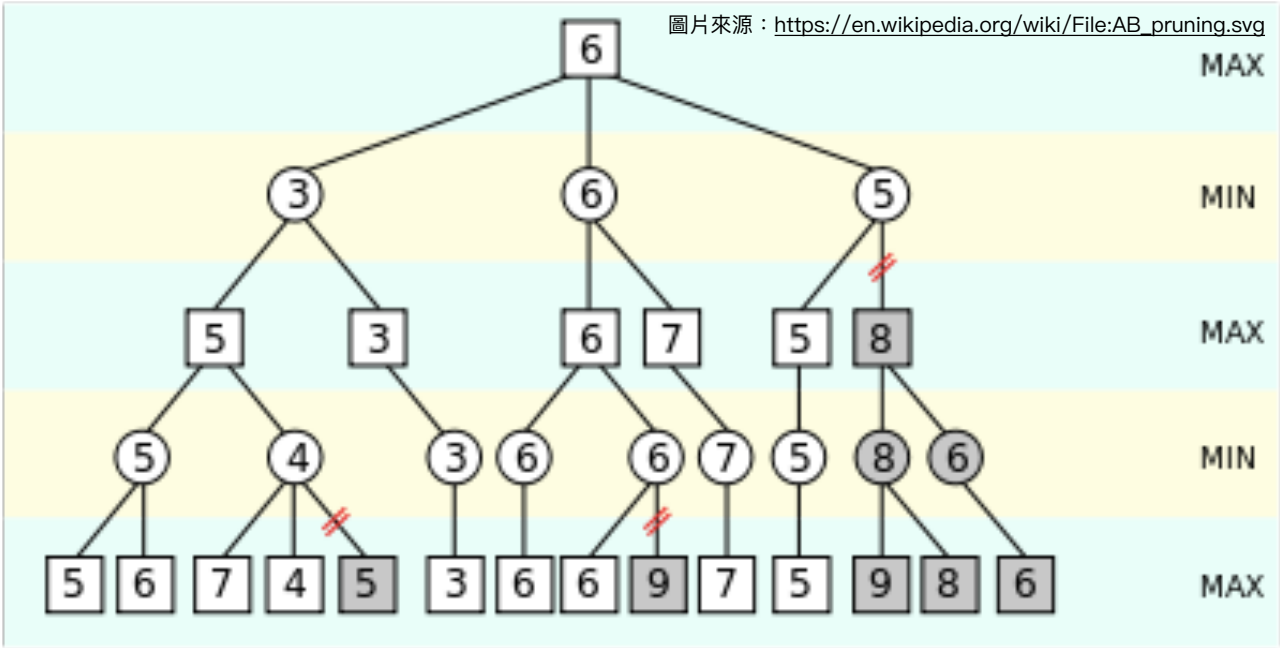
在計算下，平均一次對戰的時間為30秒。我希望可以使時間下降。我想到現在的評估函數式評估整個盤面，但事實上，除了新增的棋子周圍外，其他部分的得分會相同的。這種評估方法的好處是在這種做法下，每次都只針對新增的棋子評估，可以加速到一秒5筆對戰。但我在實作後卻發現這樣的評估方法效果明顯低於原方法，他對「防守」的敏銳度下降了。這讓我開始思考有沒有其他更好的評估方法。

四. Minimax對局搜尋+Alpha–Beta剪枝

1. 想法：

Minimax對局搜尋是用來計算棋類對局遊戲的最佳下法。做法是搜尋n步之後最好的狀態而決定這一步要下什麼。在每一步，都會用評估函數計算出值，並且以雙方都用最佳下法下為前提，因此要使己方分數最大化，對方分數最小化。也就是說，己方會下分數最高的地方，而對方則會下分數最低處。在搜尋樹上，就會是輪流搜尋maximum 和 minimum。而這就是Minimax對局搜尋。

Alpha–Beta剪枝則是針對Minimax對局搜尋的剪枝方法。Minimax對局搜尋的複雜度會隨搜尋層數以階層的方式暴增。以五子棋為例，在搜尋3層時就有225*224*223種可能。這龐大的計算量使電腦無法負荷多層的搜尋。因此就要用Alpha–Beta剪枝來減少不必要的計算。具體做法是當此層要取最小值時，若此子樹的值已經大於上一層的最大值時，因上一層是要找最大值，而此層找最小值，所以它肯定不會是答案，故可以不用繼續搜尋。



2. 實作：

我將搜尋樹分為奇數層和偶數層，分別要找最小值和最大值，並在最下面一層是呼叫評估函數，計算並回傳，找到時同時要記錄下棋的位置，才能決定最後要下哪個棋。剪枝時alpha為最大值，beta為最小值，當現在的beta>=alpha時，代表這是不可能的被取到的值，因此不用在搜尋了。

```
if depth%2==0:
    mn=1e9
    for i in range(ROWS):
        for j in range(COLS):
            if grid[i][j]==0:
                grid[i][j]=2
                score,a,b=dfs(grid,depth+1,alpha,beta)
                if score<mn:
                    mn=score
                    x,y=i,j
                grid[i][j]=0
                beta=min(beta,score)
                if beta<=alpha: break
    return (mn,x,y)
else:
    mx=-1e9
    for i in range(ROWS):
        for j in range(COLS):
            if grid[i][j]==0:
                grid[i][j]=1
```

3. 結果與問題：

我發現這樣寫出來的Minimax有兩個問題：

(1) 就算加了Alpha–Beta剪枝，運算速度還是太慢

(2) 加了Minimax的版本甚至沒有單純只用評估函數好，更常出現下無法理解的位置。像是有活三而不擋，可以直接獲勝卻選擇防守等問題。

4. 問題與討論：

(1) 運算速度太慢

針對運算速度的問題，我結合過去打資訊競賽學習演算法的經驗，發現他無論他下的順序為何，只要現在的盤面相同，則他的評估分數就會相同，當出現超過一次相同盤面時，我現在的做法會重複計算。於是，我想到可以利用動態規劃的想法記錄下已經算過的盤面分數，再出現相同的值時，就可以直接取值，而不用重複計算。

在想到這個做法後，我再度搜尋資料，發現與「Zobrist hash 算法」正是一般對於解決此類問題的有效工具。可將其面評估結果進行hash後，儲存在一個二維陣列之中，以減少重複評估。目前我還沒有實作出這一個版改良，不過確定這是個可行的改良方式。

(2) 加了Minimax的版本反而比只有評估函數的版本弱

在這一部份，我一直無法理解原因，也懷疑過是不是Minimax或Alpha–Beta剪枝寫的是錯誤的。而後，我發現問題是兩個主因導致的：

(i) 有資料顯示，在少於4層的對局搜尋下，效果是非常弱的，有時會比僅有用評估函數的差。至少要考慮到六層，才會有明顯的效果，程度大約是沒有專研過五子棋的玩家。但我現在的版本運算速度太慢，在算四層時就十分吃力，難以擴張到六層以上。而這，正是目前遇到的瓶頸。

(ii) 我的評估函數方法不好。搜尋資料後發現，大部分人做的評估函數都有特判「活三」、「死三」、「死四」。而我當時認為不做特判應該也是可行的，只是精準性或稍嫌下降。但我發現，在加上Minimax時，不做特判會使其發生嚴重的錯誤。我認為是因為原本的小錯誤，在搜尋層數增加後，會使影響擴張，導致做出錯誤的決定。

(iii) 此外，我在一篇碩士論文「五子棋相關棋類人工智慧之研究」中，看到了在搜尋樹中可以用迫著搜尋方法（TSS）來進行。這個搜尋法是由 Allis 在證明五子棋是先手必勝時一同提出。是利用防守方會被攻擊方牽制的方式進行。如果將電腦由後手改為先手的話，可以嘗試此方法。反之則進行防守防禦

(3) 改進：

我將評估函數改為先特判是否有死四，如果有的話就直接擋，不然就在看有沒有活三。如果都沒有再開始跑Minimax。此外，我還將評估函數的評估範圍改進，改為只對現在有下的點周圍進行單點評估，找到最佳位置。在經過這兩個優化後，Minimax的正確性大幅上升，且在搜尋4層的情況下，也可以在平均5秒內跑完，是可以接受的速度。但若搜尋到六層的話，平均一步就需跑超過一分鐘，這時候就需再使用其他方式優化，例如「Zobrist hash 算法」。

肆、結論

在這次報告中，我主要從基礎的五子棋評判程式，擴展至使用Minimax對局搜尋和Alpha–Beta剪枝來進行人機對戰。在這之中，我讓電腦互相對戰而找出最佳的一組評估函數：攻擊[1, 1, 8, 17, 47]、防守[1, 1, 20, 68, 94]，並運用Minimax對局搜尋+Alpha–Beta剪枝進行，發現與預期的結果有很大的差異，藉此討論其原因，並嘗試尋找更快速的優化方式。最後，我討論若使用不同的評估方式，是否對答案有所影響。

伍、心得反思

在製作這個作品時，我原本是抱持著想做人機對戰的想法開始，發現需要使用評估函數和minimax對局搜尋。但在真正開始做之後，才又冒出很多想法，擴展出許多內容。這份報告，主要是記錄了我的發想、對每個問題的想法和解決辦法、中間遇到的困難與挑戰、和因而延伸出的更多方向與可能。對報告本身而言，或許敘述和分類稍嫌雜亂，但這正是因我誠實地記錄下自己想法的發散過程，和對每一個部份發生問題的探討、解決與延伸。

這次的專題製作，是完全由自己發想，在遇到困難時先自己思考，猜測問題的來由與解決方式，再查找資料確認可行性。以往的專題，我都會在開始製作前擬定詳細的計畫。但這次不同，在進行之前我沒有進行詳盡計畫，有的只是大方向。我反而更可以自由地進行，想出比平常更多的想法，也考慮到很多事前不會注意到的事。像是評估函數的最佳化就是在做完評估函數後才想到的問題；後續Minimax遇到問題的解決延伸也是如此。如果沒有這一些「意外」的發想，這次的專題也不會有這麼多驚喜，更不會發現很多可以繼續嘗試的方向。

在完成這個專題的過程中，我結合過去參與資訊競賽經驗和所學的演算法，來解決專題中遇到的問題。像是利用rolling hash來記錄評估函數是否已算過，及吸取動態規劃的經驗想出minimax的優化方式。在其中，我遇到了許多困難。不僅是debug不出來，更多時候是發生預期之外的錯誤。像是程式下的棋不符合預期，卻模擬不出其他的原因；或是運算速度太慢，優化效果不佳等。在嘗試解決和查找資料的過程中，我又發現到很多不一樣的做法和延伸的優化方式，進而吸收到了許多意料之外的收穫。

整體而言，這次的五子棋人機對戰作品對我的而言意義重大。不僅是我十分感興趣且期待已久的內容，更是完全由自己發想，自己規劃和進行的作品。結合了我過去所學的經驗，和不曾接觸的領域碰撞而產生，並嘗試解決所遇的各種困難，一步步推進擴張，造就了這一份作品。

陸、未來展望

在未來，我希望可以在幾個部分使內容更加完整，使其有更強的能力與人對戰。

1. 嘗試不同的評估方式，像是是否特判，全盤評估或單點評估，如何簡化評估數量等。並比較其優缺，包含正確性與運算速度，找出最好的評估方法。
2. 在Minimax對局搜尋上，尋找優化方式。可以嘗試加上「Zobrist 算法」，以減少重複計算，加快搜尋速度。或改以使用「啟發式評估函數」，增加Alpha–Beta剪枝效果。或嘗試使用「迫著搜尋法」來進行優化

七、參考資料

程式人雜誌：
http://programmermagazine.github.io/201407/book/pmag.html#%E9%9B%BB%E8%85%A6%E4%B8%8B%E6%A3%8B%E7%B0%A1%E4%BB%8B

Mr.Opengate：
https://www.mropengate.com/2015/04/ai–ch4–minimax–alpha–beta–pruning.html

五子棋相關棋類人工智慧之研究：
https://ir.nctu.edu.tw/bitstream/11536/73423/1/754301.pdf

New Heuristic Algorithm to improve the Minimax for Gomoku Artificial Intelligence:
https://dr.lib.iastate.edu/server/api/core/bitstreams/39a805d5–8f5b–41e6–b07c–19c07229f813/content

維基百科Alpha-Beta pruning:
https://en.wikipedia.org/wiki/Alpha%E2%80%93beta_pruning

言川的博客：
https://github.com/lihongxun945/myblog

Codepen Gomoku Game by Anton Mudrenok:
https://codepen.io/mudrenok/pen/gpMXgg