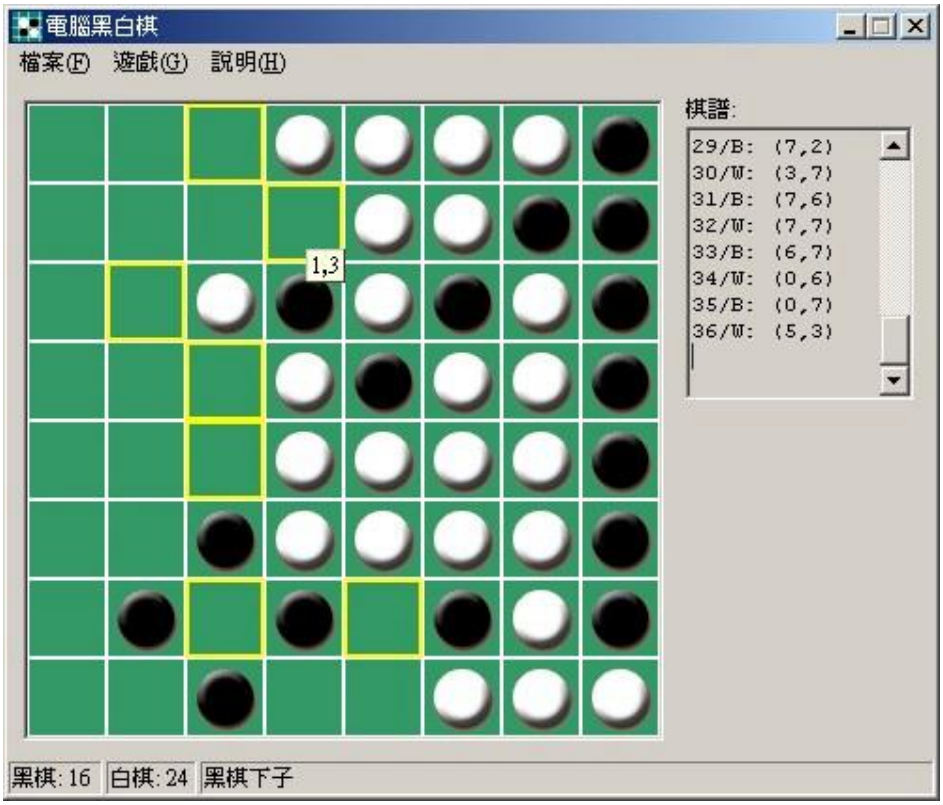
Artificial Intelligence Project

The Othello Game

NTU CSIE
B87506017, Chen-hsiu Huang (黃振修)

執行畫面



本程式使用 Borland C++ Builder 製作，在任何 Windows 平台皆可執行。解開壓縮檔之後位於 "Run\" 子目錄下的 Othello.exe 即是執行檔，直接執行即可。

黑白棋遊戲規則

黑白棋的主要用具是一塊 8x8 空格的 Othello board (多數是綠底)，另外有 64 個圓形棋子(正式尺寸是直徑 3.5 cm)，兩面分別爲一黑一白。香港坊間有一種質素劣的棋盤，棋子是用紅色和綠色的，但兩者完全是沒有分別的。

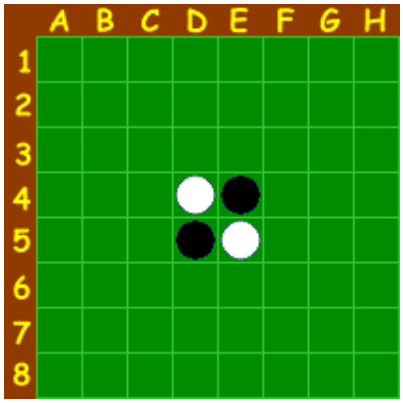


圖 1 初期配置下子方法

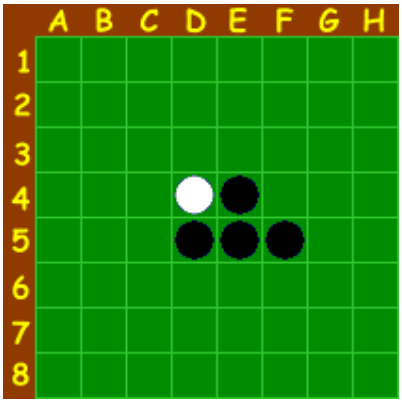


圖 2 黑 F5 之後

盤面的初期配置可參看圖 1，值得留意的是，兩顆黑子都是在 E4 、D5 兩個方格上，請不要弄錯，否則棋盤就看得不舒服了。

放好棋子後，雙方就輪流下子，注意黑棋永遠是先下的一方。下子的方法是：把自己顏色的棋子放在棋盤上的空格上，而當自己放下的棋子在橫、直、斜八個方向內有令一個自己的棋子，則被夾在中間的全部會成爲自己的棋子。例如在圖 1，黑如果下 F5，因爲 D5 的黑子，中間在 E5 原本爲白色的棋子會變成黑色，成了圖 2 的模樣。

當然，黑棋除了下 F5 以外，也可以下 E6、C4、D3 之其中一處，但因爲它們都是對稱的關係，因此爲方便起見，很多時候都會以 F5 作爲第一手棋。

之後，白棋可以有更多下子選擇了。例如，白棋下 D6 便可以將位於 D5 變成白子(圖 3a)，而白棋下不同的地方，就會有不同的效果。 (圖 3b,c)

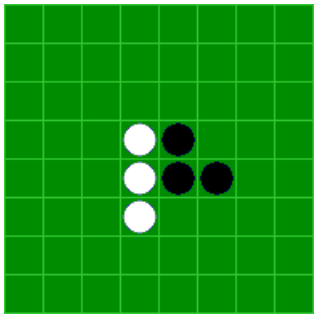


圖 3a 白棋下 D6 後的盤面

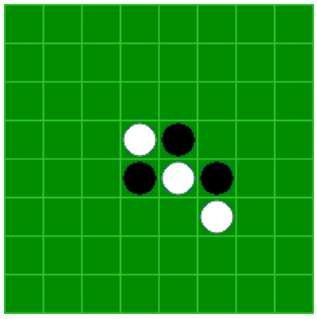


圖 3b 白棋下 F6

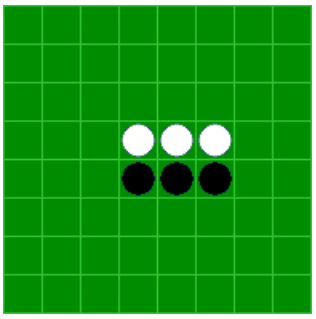


圖 3c 白棋下 F4

隨著棋盤上的子越來越多，你可以在一步內反到的棋子也多了。假如下棋的一方在橫、直、斜同時有自己的棋子，中間的所有棋子都會轉爲我色棋子，例如在圖 4，白棋如下 F5 ，則 D3-E4、E5、D7-E6、F3-F4、F6-F7、G5、G4 和 G6 間的黑子都被‘夾’住了，它們都全部成爲白棋的棋子。成了圖 4b 。正因爲如此，在一局棋最後的數手內，雙方棋子的數目可以有很大的改變。

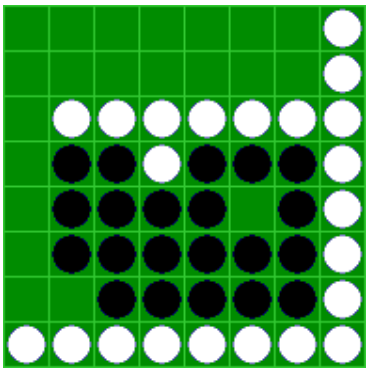


圖 4a 此時如果白棋下 F5 位

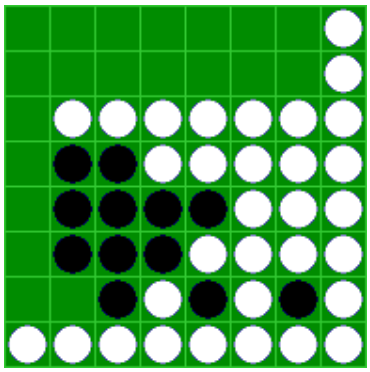


圖 4b 白棋多了十個棋子了

(以上資料取自 "黑白棋中文主題網頁 Hello! Othello")

黑白棋的技巧

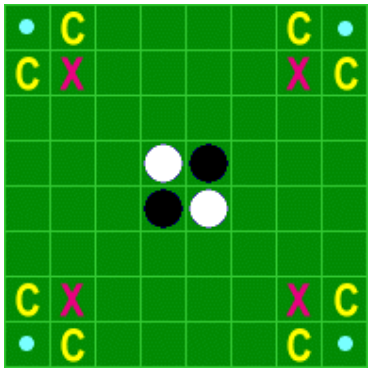


圖 10

角點、C-squares 和 X-squares

在黑白棋中最受注目的地方，都不離開這三種點：角、C-squares 和 X-squares。

角，就正如上節所提，是建立 Stable disc 的好地方，價值很高。但是，黑白棋除了四個角外，還有其他很值得留意的地方，包括 X-square 和 C-squares，它們的位置就如左圖 10 所示。

右圖中的小藍點就是角了。大家可以見到，C-square 和 X-square 都是在它的周圍，因此下這些方格有一定的危險性。現在就解釋一下：

C-square 是在邊上 (edge) 和角相鄰的方格，下這些方格有時並不代表即時的危險，因爲如果可以同時佔到一條邊上的兩個 C-square 且對稱，那多數是邊上的好形 (有關邊上的好形，我們會留待以後章節再討論) 。此外，就算只佔了一個 C-square，角上亦未有即時危險。舉個例子，圖 10 中的白子在一條邊只佔到了一個 C-squqare ，假如黑棋來子(圖 11)，希望在下一步取到角(三角形位)，但下步白子只需要吃掉它就可以了，黑棋以後亦無計可施 (圖 12)。假如黑棋再下 A 位，則白 B 位取角，黑吃大虧。

圖 10

圖 11 黑棋想拿角的時候..

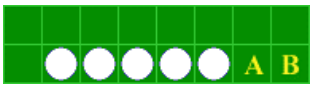


圖 12 白棋的反擊

當然，C-square 始終都在角的左右，下 C-square 雖然不能令對方立刻取得角，可是這樣下會令對方在其他地方有手段(例如換角等)，籍威脅角地在其他地方取利。棋力高的讀者，一定知道甚麼是 Stoner Traps ，這就是利用 C-square 取利的好例子。當然大家繼續看的時後，就會知道一些利用 C-square 的手筋。

至於 X-square 就是指在對角線上和角相鄰的方格，全個棋盤上總共有四個。論危險性，X-square 就比 C-square 高得多。因爲只要你下 X-square，對方就差不多可以肯定得到了一隻角。圖 13 中，黑棋佔了右上 X-squares，雖然白棋不能即時取角 (三角形位) ，但是只要白棋做些準備工夫 (圖 14)，黑棋就沒有辦法阻止對方得角，於是白棋便勝定了。

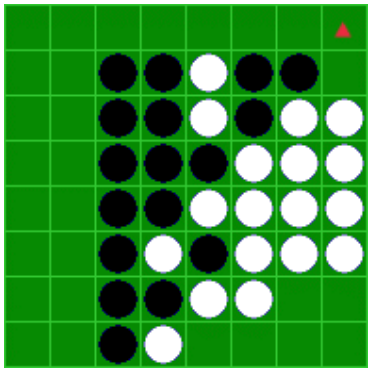


圖 13 黑棋佔了 X-squares

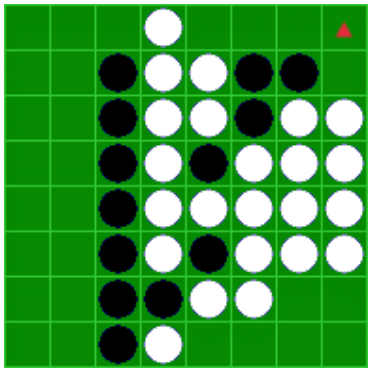


圖 14 右上角必然爲白棋所有

Mobility

Mobility 是指一方可以下子的空格數目，Mobility 越高，自己的形勢就越好。棋局中段如取太多子，Mobility 多數會因比而降低。以下所提的都是黑白棋的好形：

- 棋子不太多
- 自己的棋子凝成一團 (這正好和圍棋相反，是不是？)

相反，有關 Mobility 惡形就是：

- 大量棋子所形成的棋形
- 自己棋子各散四周的棋形
- 形成厚壁的棋形

(以上資料取自 ”黑白棋中文主題網頁 Hello! Othello”)

穩定子 (Stability)

有一些下黑白棋經驗的人都知道，占住棋格的四個角落很重要，因爲下在角落的子不會被別人吃掉，是穩定的。這一點對於最多吃子法和最少吃子法來說都是一樣的。

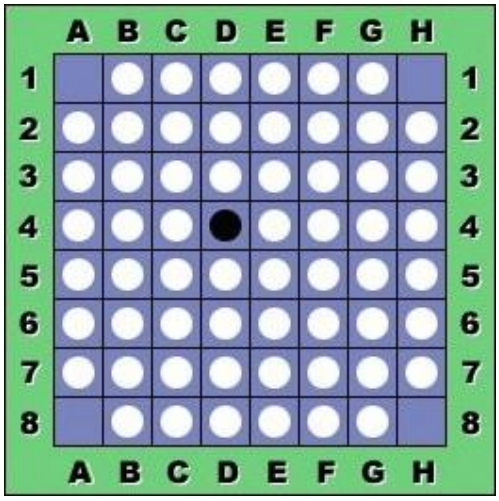
防止對手取得角落的方法是，**不要在靠近角落的三個格子內下子(X-Square, C-Square)**。當然，接近終局時有時會犧牲角落來換得其他地方的優勢。

在邊上的棋子只能被一個方向吃掉，而遊戲初期是很難吃掉對方邊上的子的，所以最多吃子法會儘量佔據棋格的四條邊。而最少吃子法則不會，因爲**在邊上的子如果沒有貼著角落，就是不穩定的棋子**。

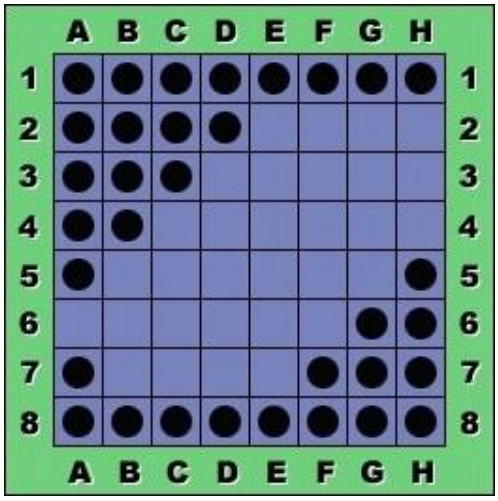
圖一很好的說明了這一點，白棋佔據了四條邊，但最後都被吃光了。

那麼什麼叫穩定子呢？一個簡單的判定方法是，在棋子的橫、豎、撇、捺四條線上，**如果線的一個方向全是自己的棋子或邊界，或是線的兩個方向都填滿了棋子，則稱這條線是穩定的。如果棋的四條線是穩定的，則稱這個棋子是穩定的**。

如圖二示，黑方已經有 33 個穩定子，就是說不會被別人吃掉的子，因此在接下來的比賽中，想輸掉都沒有可能了。（因爲 33>64/2）



圖一



圖二

(以上資料取自 ”黑白棋世界”)

程式演算法

程式主要利用 MinMax Algorithm 來作爲搜尋的演算法。在簡單的模式中我們往下推兩層，普通模式中往下推四層，高難度模式則是往下推六層。

主要的程式寫在 TOthello.cpp 中，以一個 TOthello 物件代表遊戲狀態在每一個回合中，搜尋函式 Search(), 依次序呼叫 alphaSearch() 以及 betaSearch() 函數分別代表白棋以及黑棋所選擇的下法：

```
void TOthello::Search()
{
    int i, j;
   int best_x = -1, best_y = -1;
   int score, maxScore = -99999;
   bool done = false;
   deep = 0;

   TOthello saved_state;
   saved_state = *this;

   for(i = 0; i < 8; i++)
    for(j = 0; j < 8; j++)
          if(checkAllow(j, i)) {
                board[i][j] = OTHELLO_WHITE;
             changeBoard(j, i);

                score = alphaSearch(*this, 0);
                if(score > maxScore) {
                maxScore = score;
                 best_x = j;
                 best_y = i;
             }
             // restore last board
             *this  = saved_state;
          }
    // determine the best decision
   if(best_x != -1 && best_y != -1) {
        PutDown(best_x, best_y);
       done = true;
   }
   if(!done)
        turn *= -1;
}
```

```
int TOthello::alphaSearch(TOthello state, int d)
{
    int i, j, value = 0;
   int score, maxScore = 1000000;

   if(d >= depth)
    return state.evail(OTHELLO_WHITE);
```

```
    for(i = 0; i < 8; i++)
     for(j = 0; j < 8; j++) {
           if(checkAllow(j, i)) {
                 state.board[i][j] = OTHELLO_WHITE;
                 state.changeBoard(j, i);
             value = state.evail(OTHELLO_WHITE);
              score = value + state.alphaSearch(state, d + 1);
                 maxScore = score > maxScore ? score: maxScore;
           }
     }
     return maxScore;
}
```

```
int TOthello::betaSearch(TOthello state, int d)
{
     int i, j, value = 0;
    int score, minScore = 1000000;

    if(d >= depth)
     return state.evail(OTHELLO_BLACK);

    for(i = 0; i < 8; i++)
     for(j = 0; j < 8; j++) {
           if(checkAllow(j, i)) {
                 state.board[i][j] = OTHELLO_BLACK;
                 state.changeBoard(j, i);
             value = state.evail(OTHELLO_BLACK);
              score = state.alphaSearch(state, d + 1) - value;
                 minScore = score > minScore ? score: minScore;
           }
     }
     return minScore;
}
```

估值函數(Evaluation Function)

在設計 AI 的程式中，遊戲的好壞取決於估值函數的好壞。因此我根據網路上找到有關下黑白棋的技巧（如上述）自行設計估值函數。

我的估值函式是以加權方式計算，落子在越好的地方加的分數越高（例如角點），而**一開始盡量不要去下 C-Square 點**（因爲很有可能因此角點被對方搶走），所以如果是落在 C-Square 或是 X-Square 上面的點會以扣分方式處理。

而**邊上的點更是重要**，加的分數也很高。另外就是如果我方佔住角點，該角點的 C-Square 和 X-Square 就應該去下，這裡乘上負號以達到加分效果。

另外穩定線也是很重要的。我們給予棋面上的穩定線額外的加分，讓於是程式就會試著去製造穩定線，並試著去打破對方的穩定線（**這部分的效果很好，但卻是我的黑白棋程式致命的弱點，容後再敘**）。

一旦程式每佔到一個角點，就應該盡可能的促使穩定線的行程，所以每佔到一個角點都會提高穩定線加分。

```
CORNER = 3000;
MOBILITY = 30;
STABLE = 25;
XSTABLE = 50;
C_SQUARE = -1000;
X_SQUARE = -900;
EDGE = 900;
THIRD_SQUARE = 500;
```

以上是我預設的計分方式，部分加權會隨著局勢改變。估值函數爲 TOthello::evail()，每回合的盤面局勢是以己方減去對方的分數爲考量。

程式評估

感覺上這個程式好像會蠻厲害的，其實不然。稍微善用技巧的人通常可以藉由佔邊或是角點而取得優勢。因爲穩定邊及盤面棋數的考量，程式往往會在開局及中局拼命製造穩定邊，所以開局或是中局電腦棋數往往很多，進而忽略對邊和角點的攻佔，等對方佔到邊或是角點之後，往往就被豬羊變色，因而慘敗。

那麼我試著調高對邊和角點的加權分數，但是這裡會遇到一個問題：因爲邊的加分太高，讓程式選擇去下該死的 C-Square 點，因爲這個加分機制，往往可以刻意製造陷阱騙程式去下 C-Square 點，然後程式上當之後對方就很容易佔到角點，於是又是一次豬羊變色。

我發現這裡的加分機制間的差距扮演相當重要的角色，角點比邊點重要，那麼他的分數應該是邊點的幾倍？C-Square 以及 X-Square 點應該扣多少才會讓程式不去下這兩點，確又不會因爲扣分過多讓程式不去搶邊點？

程式寫好之後我爲了調整這些數字花了兩天的時間，總是無法調出一個滿意的數值，往往是顧此失彼，邊角一旦失守，製造再多的穩定邊也沒用，盤面上的棋數更是不準確。我和室友討論一整夜的結論是：這樣的加分方式很難躲過技巧性的欺騙，多下幾次就可以猜出程式在想什麼，會去搶什麼位置，優先次序是怎樣，總是可以找到破解方式。

我們在想也許這時候就需要有棋譜的幫忙吧？累積經驗幫助程式不易受騙。可是這個專題沒有時間讓我去作這樣的大工程。

後來我找了網路上其他的黑白棋程式，發現都比我的程式強，我找了一個程式來看的的原始碼，發現他的估值函式非常簡單，只有判斷角點和邊點，其他的全靠盤面上的棋子數定勝負（而我的棋子數卻是加權最低的），然後剩下的就靠搜尋去找出最佳解。

既然找不到最好的加分方式，就乾脆不要理他，讓搜尋和盤面結果去決定一切，這反而是最直接了當的作法，我過分考慮各種狀況（當然考慮這些狀況都是對的），反而因爲找不出最佳的加分方式而吃大虧（由此可見 Evaluation Function 的重要性）。

後來繼續跟室友討論，即使找到最佳的加分方式，對邊角點有最好的攻防，只要玩家不去裡那些邊角之爭，拼命找穩定邊穩定子，還是有機會贏的。

心得感想

雖然花了好幾天的心力寫出一個笨笨的黑白棋程式，心裡有點沮喪，但總還是知道到底問題出在哪裡，用最直接單純的作法反而最好。不過因爲期末的關係，也沒什麼時間在去重寫，重新測試了。

就把這個笨笨的黑白棋程式當成是個紀念吧！

參考網站

1. 黑白棋評論 Disco Othello World (http://www.disco.com.hk/)
2. 黑白棋中文主題網頁 Hello! Othello (http://web.hku.hk/~h0014282/index.htm)
3. 黑白棋世界 (http://go7.163.com/~blacwet/index.html)