第 13 章　五子棋遊戲

簡介：

五子棋是個規則簡單的棋盤式遊戲，但是已經包含多數棋盤遊戲製作的基本要素。包括：棋盤製作、交替下棋不得重複的控制，以及勝負規則的判斷等等。本單元將使用動態產生物件的程式技巧製作出有大量棋格的棋盤，每個棋格物件還必須能回應下棋的動作，就是具備被滑鼠點擊時的事件副程式！當然還必須有相當聰明，而且不能太冗長的連線成功與否的判斷程式。

13-1 製作棋格與棋子物件

[製作棋格與棋子圖案]

本單元有關棋格與棋子的基本構想是以一個 Label 物件來製作，每個棋格是一個方形的標籤，未下棋時顯示一個十字形，很多棋格並排時看起來就是一個圍棋棋盤的外觀了！當標籤被點擊時就當作有人在此格下棋，如果下白棋就將標籤圖案換成一個空心圓圈，下黑棋就是實心黑圓圈了！所以首先要製作棋格與黑白棋子的基本圖案，相信應該沒有人不會用小畫家畫圓圈與十字架吧？請作出 50x50 像素點的下面三個圖案(最好以 GIF 格式儲存，檔案較小且不失真)。

[製作棋格物件的副程式]

做好之後請依序以 C、W 與 B 的名稱加入新建立專案的資源檔(分別代表 Cross, White & Black)，新專案的名稱可取為 CsGobang2010，Gobang 是五子棋的英文名稱。接下來模仿前一單元製作磚塊的副程式，先建立一個自訂的製作空棋格物件的副程式如下：

```
private Label Cell(int x, int y)
{
    Label Q = new Label();
    Q.Width = 50;
    Q.Height = 50;
    Q.Image = Properties.Resources.C;
    Q.Left = x;
    Q.Top = y;
    return Q;
}
```

標籤長寬皆 50 點，載入十字圖案，不放心的話可以在 Form_Load 事件寫個程式叫用它：this.Controls.Add(Cell(0, 0));，試驗一下(如打磚塊單元)。

13-2 製作空棋盤

[製作填滿棋格的表單]

五子棋一般雖然以圍棋棋盤來下，但基本上並沒有絕對的長寬限制，本單元將先預設將表單放到最大，再自動偵測視窗大小盡量塞進最多的棋格。所以首先將表單的 WindowState 設為 Maximized。再到 Form_Load 事件寫程式如下：

```
int nx, ny;//棋格寬與高數目
int[,] P;//用來記錄其閤下棋狀況之陣列：空格=0；白棋=-1；黑棋=1
private void Form1_Load(object sender, EventArgs e)
{
    nx = this.ClientSize.Width  / 50;//橫向棋格數
    ny = (this.ClientSize.Height  - 24) / 50;//縱向棋格數
    P = new int[nx, ny];//宣告棋格對應陣列
    for (int i = 0; i <= nx - 1; i++)//橫向棋格
    {
        for (int j = 0; j <= ny - 1; j++)//縱向棋格
        {
            this.Controls.Add(Cell(50 * i, 50 * j));//產生棋格物件
        }
    }
}
```

[計算棋格的長寬個數]

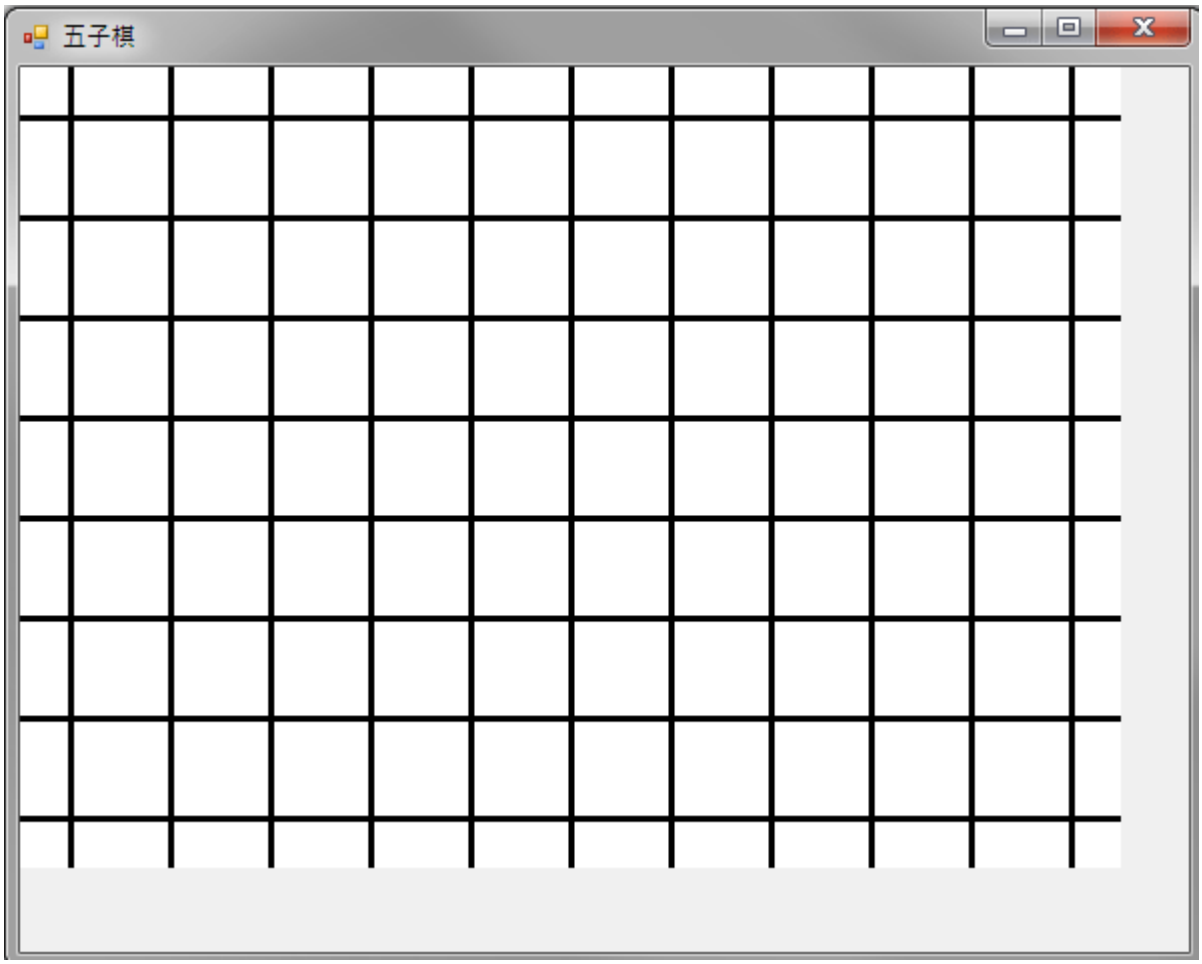
上述程式碼已經盡可能加入註解。nx 與 ny 代表表單內可以置放的棋格寬與高數目，每個棋格 50 點長寬，所以表單可用區域寬(this.ClientSize.Width)除以 50 就是橫向可以擺的格數，縱向也是一樣！因為相除的兩個數字都是整數，按照一般程式語言的規則，正數相除的結果會是無條件捨去小數的一個整數，所以棋格實際佔據面積會小於或等於視窗內部的範圍，不會超出表單。

[紀錄棋盤狀態的二維陣列]

接著宣告 P 陣列，這是一個二維的陣列，架構與一般常見的表格一樣，你可以想像就是填寫每一個棋格目前狀態的表格！0 表示空格，1 表示黑棋，-1 表示白棋。需要注意的是陣列宣告的語法，我們在副程式之外宣告 int[,] P 意義是：P 是一個整數的陣列，這裡的括號是方形的中括號！中間的逗點表示此陣列有兩個維度，就如表格有幾列與幾欄是兩個維度的意義相同。接著在副程式內計算好 nx 與 ny 之後，明確宣告這個陣列的長寬且建置(new)它 P = new int[nx, ny];。因為最先的 P 是在公共區域宣告的，所以各個副程式都可以共用這個陣列。

[用迴圈製作棋盤]

接著就是一個雙層的迴圈，內迴圈依序建立垂直向的棋格，外迴圈則移動水平向座標，與前一單元建立磚塊的程式邏輯相同！此時執行程式畫面概略如下，未能完全貼齊的邊緣就是因為表單長寬未必剛好是 50 的倍數之故，你可以想想辦法用程式動態調整版面。當然實際情況畫面會比較大，看你的螢幕大小而定。



13-3 下棋的程式

[以樣板製作棋格的 Click 事件副程式]

下棋的動作其實是用滑鼠去點擊棋格物件的事件！這就有點尷尬了，在設計階段我們的棋格根本不存在！如何替它們寫事件副程式呢？答案是找替身！先到工具箱叫一個 label1 物件出來，到它的屬性欄選擇事件→Click，建立事件副程式框架，假裝是替棋格寫程式，之後再設法用程式碼連結這個副程式到新增的棋格物件就好了！當然也必須讓 label1 物件看不見(Visible=false)或乾脆殺掉它都可以！(是不是很像電影明星的替身？)

```
Boolean T; //切換黑白棋的指標，True=黑；False=白
private void label1_Click(object sender, EventArgs e)
{
    Label Q = (Label)sender; //取得被點的棋格稱之為Q
    int x = Q.Left / 50; //計算棋格在表格的X位置
    int y = Q.Top / 50; //計算棋格在表格的Y位置
    if (P[x, y] == 0) //棋格是空的！
    {
        if (T)
        { //下黑棋
            Q.Image = Properties.Resources.B;
            P[x, y] = 1;
        }
        else
        { //下白棋
            Q.Image = Properties.Resources.W;
            P[x, y] = -1;
        }
        T = !T;
    }
}
```

[T 是下棋序的控制變數]

這段程式碼好長！必須好好說明。首先 T 是個 Boolean 變數，就是只有 True 與 False 兩種值的變數。下棋時不是輪到黑棋就是白棋，我們用此變數記住現在該誰下了？預設值是 False，就是白棋會先下。等到白棋落子之後此變數應該變成 True，稍後就是該黑棋下了！

[sender 是被點的棋格]

進入事件副程式一開始是將 sender 物件，就是觸發事件的棋格物件，也就是被點的格子(可能是棋盤上幾百個格子的任何一個)，在此用一個代名詞 Q 來表示。因為 C# 要求明確宣告物件型別，所以(Label)sender 這個動作是必要的，雖然我們知道棋格一定是 Label 但是直接用 sender 代表棋格物件 C#還是不接受的！一定要寫：Label Q = (Label)sender; 就對了！

[計算棋格對應的陣列位置]

接著是計算棋格對應到二維陣列記錄的哪一格？因為一個棋格長寬 50 點，所以用物件位置座標(Left & Top)除以 50 就算出來了！而且陣列的宣告雖然用資料個數 nx & ny，但是呼叫資料時的索引卻是從 0 開始，所以 P 陣列是從 P[0,0]開始而不是 P[1,1]！這樣 Q.Left / 50 的結果剛好是索引值範圍的 0 ~ nx-1。反正被點的格子 Q 目前被甚麼棋子佔據(或是空的)資料就記在 P[x,y]裡面！我們先檢查 P[x,y]，如果資料為 0 表示棋格是空的，才可以進行下棋的動作，否則以下程式不會執行！就是說如果點到已經有棋子的棋格，不會有下棋的動作反應！

[顯示與紀錄黑白棋的位置]

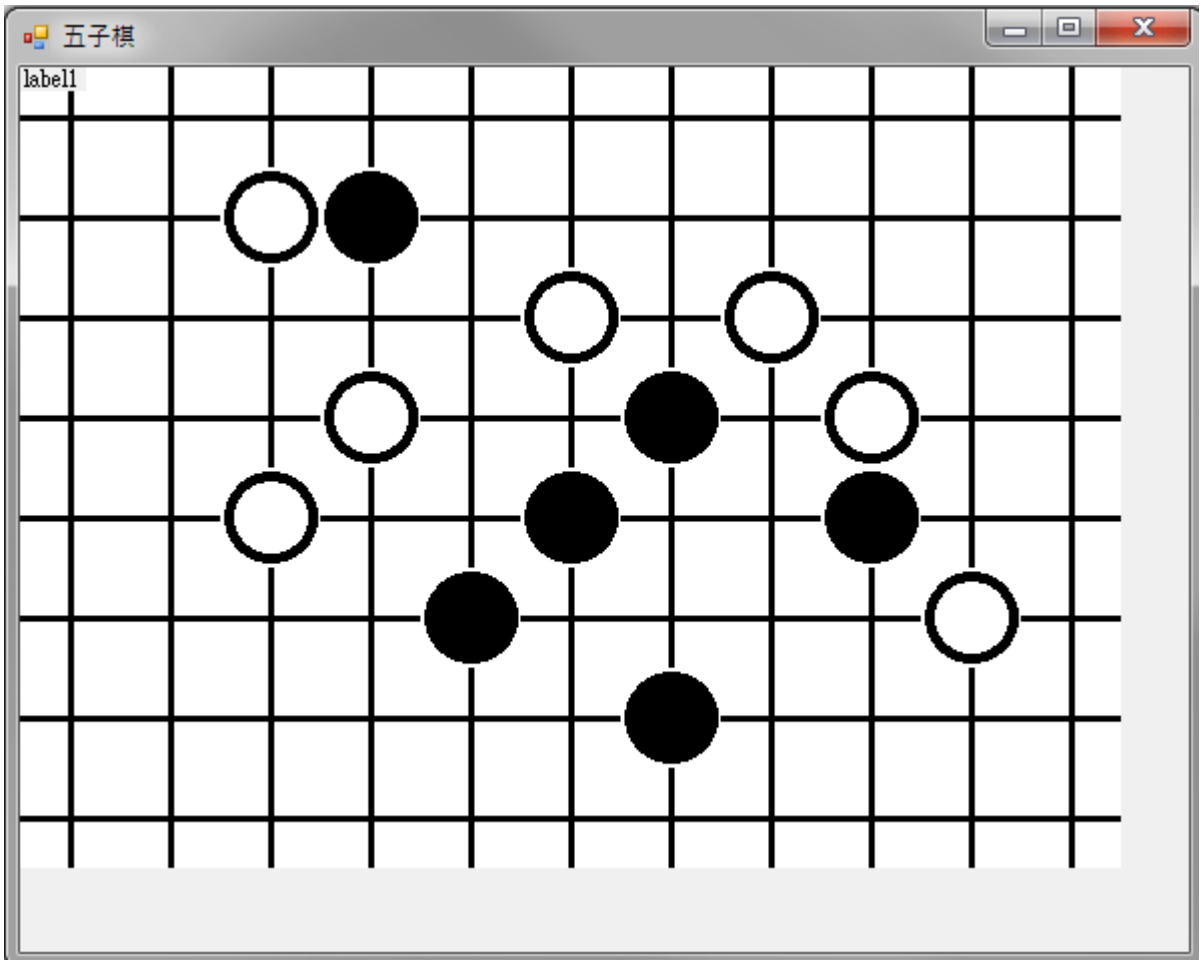
確定可以下棋時就得看該誰下了？(T 變數)如果該下黑棋就換圖片為 B，且將 P[x,y] 的值設為 1；相反的，下白棋就將圖改為 W，P[x,y]=-1。不管黑白棋，下完之後必須將 T 變成相反(T = !T)，下一次就會下不同色的棋了！

[新增事件副程式到棋格物件]

但是僅僅完成這個複雜的程式，它還是與你動態建置的棋格們沒有關係，請在建置棋格的 Cell 副程式加入一行連結此副程式到棋格的程式(粗體字)，這樣每一個棋格都會有正確的下棋反應了！

```
private Label Cell(int x, int y)
{
    Label Q = new Label();
    Q.Width = 50;
    Q.Height = 50;
    Q.Image = Properties.Resources.C;
    Q.Left = x;
    Q.Top = y;
    Q.Click += new EventHandler(this.label1_Click);
    return Q;
}
```

現在執行程式，你已經可以自由在空格下棋，而且黑白棋會交替了！畫面大致如下：



13-4 連線了嗎？

[檢查落子處周邊連線狀況]

下棋的互動介面完成，其實常常才是最複雜程式邏輯的開始！因為完整的遊戲軟體必須能夠判斷下棋的輸贏狀況，在五子棋來說就是黑或白棋有人在垂直、水平、左上到右下、或右上到左下兩種斜線任何一個方向達成五個同色棋子連成一線。聽起來就有夠難了！我們就從簡單一點的水平線開始，一題一題解決吧！請先建立一個稱為 chkWin 的自訂副程式如下：

[檢查水平連線的程式]

```
private int chkWin(int x, int y)
{
    int t = P[x, y]; //檢查棋種
    int n; //連線棋子累積個數
    //水平連線
    n = 0;
    for (int i = -4; i <= 4; i++)
    {
        int xx = x + i;
        if (xx >= 0 && xx <= nx - 1)
        {
            if (P[xx, y] == t)
            {
                n += 1;
                if (n >= 5) { return t; }
            }
            else
            { n = 0; }
        }
    }
    return 0;
}
```

這個副程式的使用邏輯是當某人在某一位置(x, y)下了一顆棋子的時候，可以呼叫此程式檢查以這個棋子為中心，是不是有水平方向連線成功的狀況，如果是黑棋就檢查黑棋是否連線，白棋就是看有無白棋連線了！所以第一行程式 int t = P[x, y];是取得要檢查的棋種(1 或- 1)。接著從該棋子的左邊第四個棋子到右邊第四個棋子之間掃描！如果要看的棋格位置(xx = x + i)沒有超過棋盤(xx >= 0 && xx <= nx - 1)，而且與剛剛下的棋子相同(P[xx, y] == t)則計數器 n 加 1，否則計數器歸零(n=0)！

[連線中斷或成功的狀況處理]

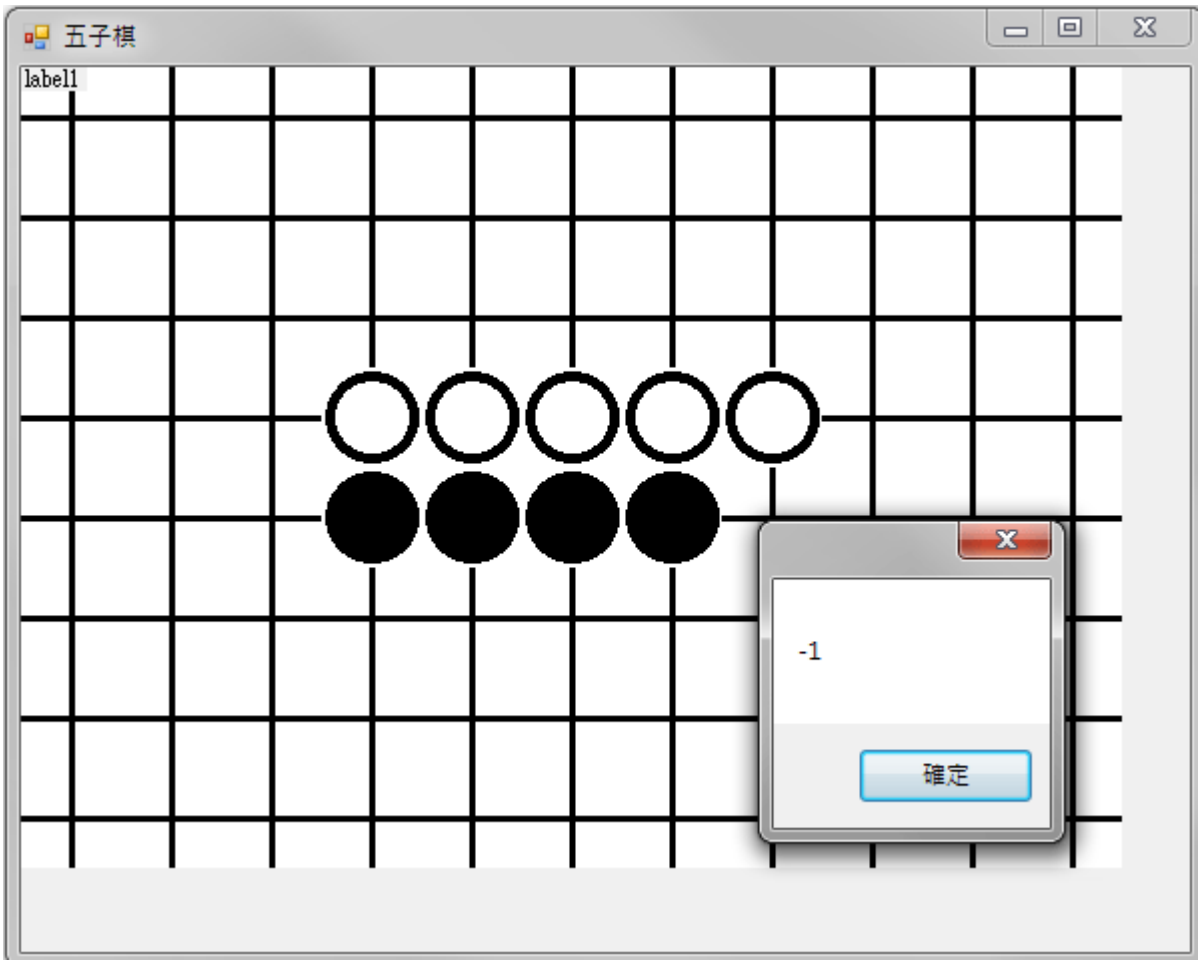
因為連續檢查過程中，一遇見一個不同的黑或白棋就是連線中斷了，必須重新計算 n 值！在累計過程中一旦達到 n=5 就是連線成功了，本副程式會回應呼叫者(return)一個代表棋種的數字(1 或-1)，同時副程式也就執行結束了！一個連線就贏了，是否同時有其他連線就不重要了！那麼，如果一直找完可能連線的範圍(x - 4 到 x + 4)還是沒有累積達到 5，那就是尚未有連線成功的狀態，程式執行到末尾時就回應使用者一個 0 了(return 0;)！

[呼叫連線檢查程式]

這個副程式目前只有檢查水平連線的能力，且還未被叫用，我們就先來試試看它行不行再說吧？請在 label_Click 事件中加入以下粗體字程式碼：

```
private void label1_Click(object sender, EventArgs e)
{
    Label Q = (Label)sender; //取得被點的棋格稱之為Q
    int x = Q.Left / 50; //計算棋格在表格的X位置
    int y = Q.Top  / 50; //計算棋格在表格的Y位置
    if (P[x, y] == 0) //棋格是空的！
    {
        if (T)
        { //下黑棋
            Q.Image = Properties.Resources.B;
            P[x, y] = 1;
        }
        else
        { //下白棋
            Q.Image = Properties.Resources.W;
            P[x, y] = -1;
        }
        int ans = chkWin(x, y);
        if (ans != 0) { MessageBox.Show(ans.ToString()); }
        T = !T;
    }
}
```

上述程式碼意義是宣告一個 ans 變數承接 chkWin 的結果，如果答案不為 0(!=)表示有人連線成功，就簡單的將答案秀出，1 就是黑棋勝，-1 就是白棋勝！當然這樣的訊息顯示方式很爛很簡陋，但是不急啦，先把困難的事情先搞定！測試一下，故意點個水平 5 連線看看程式會不會偵測到，並有訊息反應？如果一切 OK 會如下圖，白棋率先五連線，跳出-1 的訊息！



[檢查垂直連線的程式]

水平連線檢查如果成功了，你只要複製這個區段，將 x 與 y 值對調一下就可以完成垂直線檢查程式段落如下，只有粗體字部分有修改：

```
//垂直連線
n = 0;
for (int i = -4; i <= 4; i++)
{
    int yy = y + i;
    if (yy >= 0 && yy <= ny - 1)
    {
        if (P[x, yy] == t)
        {
            n += 1;
            if (n >= 5) { return t; }
        }
        else
        { n = 0; }
    }
}
```

[檢查斜向連線的程式]

水平或垂直的檢測是移動 x 或 y 值掃描，斜線方向的檢測則是同時移動 x 與 y 座標，

譬如左上到右下就是 x 與 y 同步增加座標 1；右上到左下則是 x - 1 時 y + 1，程式碼如下：

```
//左上到右下
n = 0;
for (int i = -4; i <= 4; i++)
{
    int xx = x + i;
    int yy = y + i;
    if (yy >= 0 && yy <= ny - 1 && xx >= 0 && xx <= nx - 1)
    {
        if (P[xx, yy] == t)
        {
            n += 1;
            if (n >= 5) { return t; }
        }
        else
        { n = 0; }
    }
}
//右上到左下
n = 0;
for (int i = -4; i <= 4; i++)
{
    int xx = x - i;
    int yy = y + i;
    if (yy >= 0 && yy <= ny && xx >= 0 && xx <= nx)
    {
        if (P[xx, yy] == t)
        {
            n += 1;
            if (n >= 5) { return t; }
        }
        else
        { n = 0; }
    }
}
```

[完整的檢查連線程式]

最終這個副程式有點長，但是想想它的聰明程度就會覺得還好！可以偵測四個方向兩種棋子是否連線成功的狀態。程式碼約 70 行而已：

```
//連線檢查程式
private int chkWin(int x, int y)
{
    int t = P[x, y]; //檢查棋種
    int n; //連線棋子累積個數
    //水平連線
    n = 0;
    for (int i = -4; i <= 4; i++)
    {
        int xx = x + i;
        if (xx >= 0 && xx <= nx - 1)
        {
            if (P[xx, y] == t)
            {
```

```
            n += 1;
            if (n >= 5) { return t; }
        }
        else
        { n = 0; }
    }
}
//垂直連線
n = 0;
for (int i = -4; i <= 4; i++)
{
    int yy = y + i;
    if (yy >= 0 && yy <= ny - 1)
    {
        if (P[x, yy] == t)
        {
            n += 1;
            if (n >= 5) { return t; }
        }
        else
        { n = 0; }
    }
}
//左上到右下
n = 0;
for (int i = -4; i <= 4; i++)
{
    int xx = x + i;
    int yy = y + i;
    if (yy >= 0 && yy <= ny - 1 && xx >= 0 && xx <= nx - 1)
    {
        if (P[xx, yy] == t)
        {
            n += 1;
            if (n >= 5) { return t; }
        }
        else
        { n = 0; }
    }
}
//右上到左下
n = 0;
for (int i = -4; i <= 4; i++)
{
    int xx = x - i;
    int yy = y + i;
    if (yy >= 0 && yy <= ny - 1 && xx >= 0 && xx <= nx - 1)
    {
        if (P[xx, yy] == t)
        {
            n += 1;
            if (n >= 5) { return t; }
        }
        else
        { n = 0; }
    }
}
return 0;
```

```
}
```

試試看，你的程式應該可以隨時知道誰贏了！

13-5、進階挑戰

一、如何改善勝負訊息的機制？讓他們顯示白方勝或黑方勝？
提示：使用 if else 結構顯示不同的訊息文字。

二、如何建立重玩機制？
提示：可加入功能表，恢復所有棋格圖案為十字，陣列元素全歸零

三、如何讓棋子可以拖曳移動位置？
提示：替棋格物件加入滑鼠事件，但如何處理資料變動難度頗高。

課後閱讀

淺談陣列

本單元用到了陣列，不好意思，一開始介紹就是二維的陣列。陣列宣告的語法其實很多，如果要宣告一個一維整數陣列 Q 有五個元素，最簡單的是語法是：

```
int[ ] Q=new int[5];
```

上述語法先說明資料型態(int)，再用中括號表明是陣列(非單一變數)，接著是名稱(Q)，至此只是宣告並未建置，等號後面的 new int[5];才是具體建置此五個元素的陣列，之後就可以用 Q[index]的語法引用或設定陣列元素的值了！在本單元中上述基本語法被切開了，先宣告 int[,] P;再到時計算出元素個數的地方宣告 P=new int[nx,ny];這也是常用的方式。如果你想直接宣告陣列的元素內容，也可以這樣寫：

```
int[ ] Q = { 1, 2, 3, 4, 5 };
```

如此 Q[0]就等於 1，Q[4]=5，依此類推，陣列共有五個元素。如果要直接建立 2x2 的二維陣列寫法可以是這樣：

```
int[ ,] Q = {{ 1, 2}, {3, 4} };
```

如此，Q[0, 0]就會是 1，Q[1, 1]就是 4 了！Q[1, 0]呢？應該是 3。

甚麼是 EventHandler？

本單元中將 label_Click 副程式加入為新增棋格物件的程式是：

```
Q.Click += new EventHandler(this.label1_Click);
```

其實任何一個物件加入事件副程式時都有類似語法的一行程式出現在 Designer.cs 檔案內！在此 Event 就是事件，Handler 就是處理事件的程式，上述程式可以將標題為 this.label1_Click 的副程式設定為 Q.Click 的事件副程式。凡事有加入應該就可以刪除，那麼如果要將上述連結切斷，就是要除去這個事件副程式該如何做呢？很有趣，就是：

```
Q.Click -= new EventHandler(this.label1_Click);
```

加號變成減號就是移除副程式了！事實上在 VB 程式中一樣的功能是用 AddHandler 與 RemoveHandler 這樣的語法進行的，網路上也可以找到不少程式範例。所以很多人或許會和我一樣，先學會 VB 之後再學 C#，剛開始會找不到這個完全不同的語法。不過還是鼓勵大家，新版的 VB 與 C#都是使用同樣的基底函式庫.NET Framework，如果其中一種語言可以作到的，99.99%的機率，另一個程式語言也一定可以作到！效能也應該沒有差異。所以下次要作甚麼事情時剛好只找到另一種語言的範例，千萬別放棄，抓回來研究一下必有所獲！