

2022夏季學院 計算機程式設計期末專題書面報告

主控台遊戲四合一

第四組

組員:白謹瑜、宋品頤、張席廊

一、摘要

本次期末專題經由小組討論之後決定做一個小遊戲大全，我們一共分為四個小遊戲，分別為智慧版井字遊戲、推倒提基、變色龍以及四子棋。玩家在啟動程式後可選擇任一小遊戲遊玩，並在獲勝或失敗後得到相應之分數。

二、研究動機

從最初只是在陰極射線管上移動的小光點開始，電子遊戲誕生了，並在這數十年的歲月間經歷無數次演變與創新，漸漸改變自己的型態、也改變了世界。而身為Z世代的我們從出生便有電子遊戲相伴，懷念的掌上機和電腦遊戲帶來大大小小的回憶、陪著我們度過童年。而隨著年紀漸增，它除了早已成為生活的一部份，也讓我們對於親手製作遊戲這件事總有種憧憬，正是抱著這樣的憧憬，讓我們起了製作遊戲的念頭。

決定方向後，我們馬上進行了一番討論與嘗試，在經過好一陣子的方案測試後，我們發現與其製作一個龐大複雜的遊戲，不如各自負責一個規模小一些的遊戲，完成後再將其組合、包裝成一個完整的成果，如此不但可以保留每個人想要的特色與風格，實際製作時也不會受制於三人對於程式語言掌握度的差異。將這個想法再經過一些琢磨，小遊戲三合一的方案便成型了。

三、理論說明

1.遊戲選單部分

主程式執行的一開始，使用者會在主控台看見一排遊戲選單，而使用者將扮演玩家，可以用方向鍵的上、下來選擇自己想要遊玩的遊戲，被選擇的遊戲將會以反白的方式顯示。玩家按下Enter鍵便會開始執行當前所選擇的遊戲，每個遊戲結束後都會計算目前的勝利、失敗、平手次數，並由選單部分的程式在螢幕上刷新目前的戰績。

2.遊戲部分-推倒提基

這個遊戲是使用了桌遊"推倒提基"的概念製作而成，這是一款容易上手的策略型桌遊。首先介紹基本規則，遊戲板上會有按照固定順序排好的不同顏色圖騰、它們被稱為"提基"，而遊戲一開始，每位玩家都會各自分配到幾張用來改變遊戲板上提基順序的手牌，以及一張寫有目標提基的卡片，而每位玩家的目標就是利用手牌將各自的目光標提基移到指定的順序，遊戲結束時，完成越多目標就能得到越多分數。而這次製作的版本會在主控台上印出七種顏色不同的方塊代替圖騰，並由電腦和玩家對戰。

遊戲部分-變色龍

此遊戲算是比較少見的撲克牌遊戲，與UNO牌規則相似，能出與上一張牌相同花色或相同數字的牌。而Jack是變色龍牌，可以無視上一家出的牌。如果沒牌出的話就蓋一張手牌，當整副撲克牌用完時，比較蓋牌的點數和大小，較小者獲勝。此遊戲可以2~8人遊玩，目前在程式裡是設計成電腦與玩家兩人對抗。

遊戲部分-智慧版井字遊戲

老師在上課時曾示範過簡易版的井字遊戲，不過由於電腦是以亂數決定要下哪格，因此有點笨笨的。課後我們改良了原版的井字遊戲，讓電腦能根據現在的局勢來判斷要下哪一格，取得最大的勝率。

遊戲部分-四子棋

遊戲開始時玩家會看到一7x6之棋盤，而玩家與對手輪流選擇任一行放入一枚棋子。棋子由棋盤上方放入並從最底層格子開始向上累積，我們可藉由右側之圖示看出棋盤之樣式。四子棋之遊戲目標為比對手率先將己方四顆棋子連成一條線即為獲勝，而這四顆棋子之連線方式無論是縱、橫或斜方向，只要連成一線皆能達成獲勝條件。玩家必須與對方進行攻防戰，一邊思考如何讓己方棋子連線成功，一邊也要顧慮著防守對方的進攻。

以下簡單歸納四子棋規則：

1. 棋盤大小為7x6。
2. 雙方必須輪流把一枚己棋選擇投入任一行，讓棋子落下在底部或其他棋子上。
3. 玩家輸入1~7代表選擇將棋放入哪一行。
4. 當己方4枚棋子以縱、橫、斜方向連成一線時獲勝。
5. 棋盤滿棋時，無任何連成4子，則平手。



四、實作流程架構

1. 遊戲選單部分

程式虛擬碼:

```
1. while(程式還沒結束)
{
    2. 根據讀到的鍵盤按鍵改變選擇的遊戲
    3. 刷新畫面
}
```

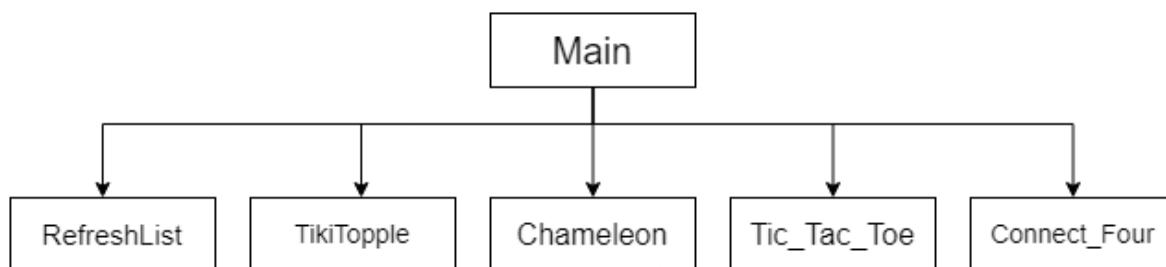
根據讀到的按鍵改變選擇的遊戲：

- 2.1 上鍵：選擇的位置上移一項，已經在第一項的話就改成選擇最後一項
- 2.2 下鍵：選擇的位置下移一項，已經在最後一項的話就改成選擇第一項
- 2.3 Enter鍵：執行目前所選遊戲名稱對應的遊戲
- 2.4 Esc鍵：程式結束

刷新畫面：

- 3.1 清除畫面
- 3.2 印出戰績、評價、遊戲列表
- 3.3 反白目前選擇的遊戲名

函式結構圖：



2.遊戲部分-推倒提基

程式虛擬碼:

```
1. 設定玩家和電腦的目標  
2. while ( 還有手牌 )  
{  
3. 電腦的回合  
4. 玩家的回合  
}  
5. 計算分數  
6. 判斷勝負並改變玩家戰績
```

設定玩家和電腦的目標:

- 1.1 從各種顏色的方塊隨機挑三個
- 1.2 加進存目標的陣列

電腦的回合:

- 3.1 抽一個亂數
- 3.2 根據抽到的數選擇要執行的手牌
- 3.3 刪除執行過的手牌
- 3.4 把剛剛的行動顯示在螢幕上

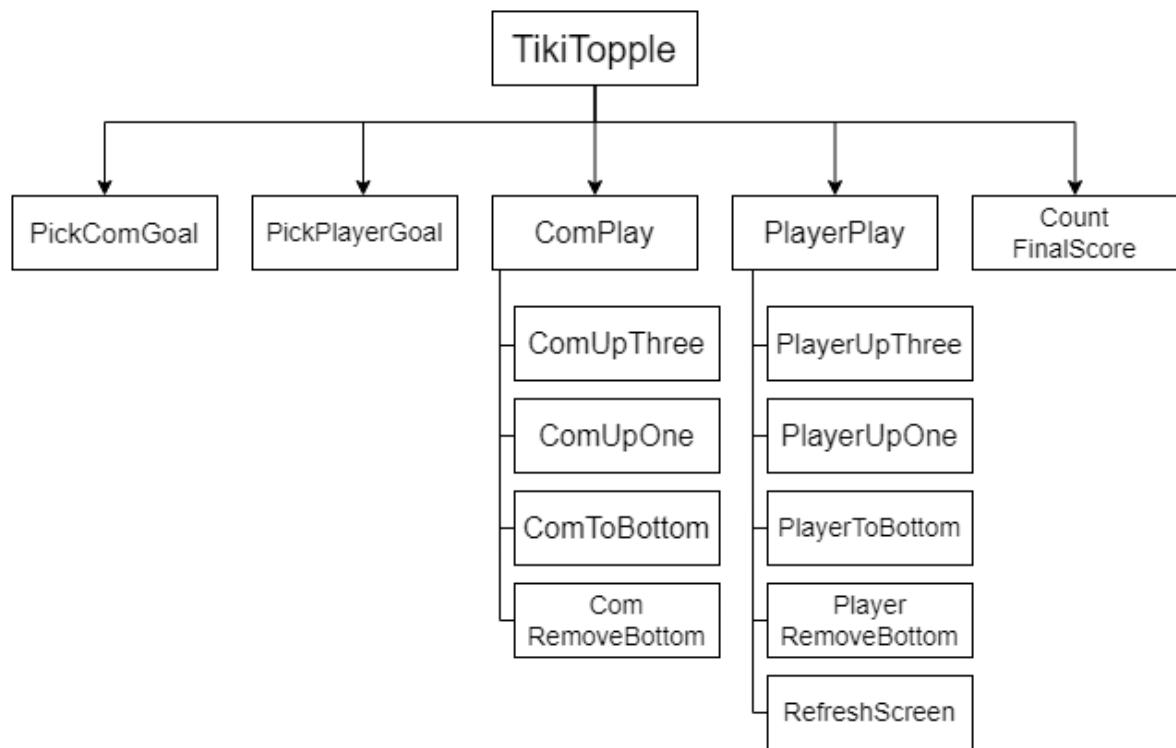
玩家的回合:

- 4.1 刷新螢幕、顯示玩家目標和手牌
- 4.2 while(回合還沒結束)
{
4.3 讀取鍵盤輸入
4.4 方向鍵=>選擇手牌
4.5 enter鍵=>執行選擇的手牌效果，刪除執行過的手牌，回合結束
}

計算分數:

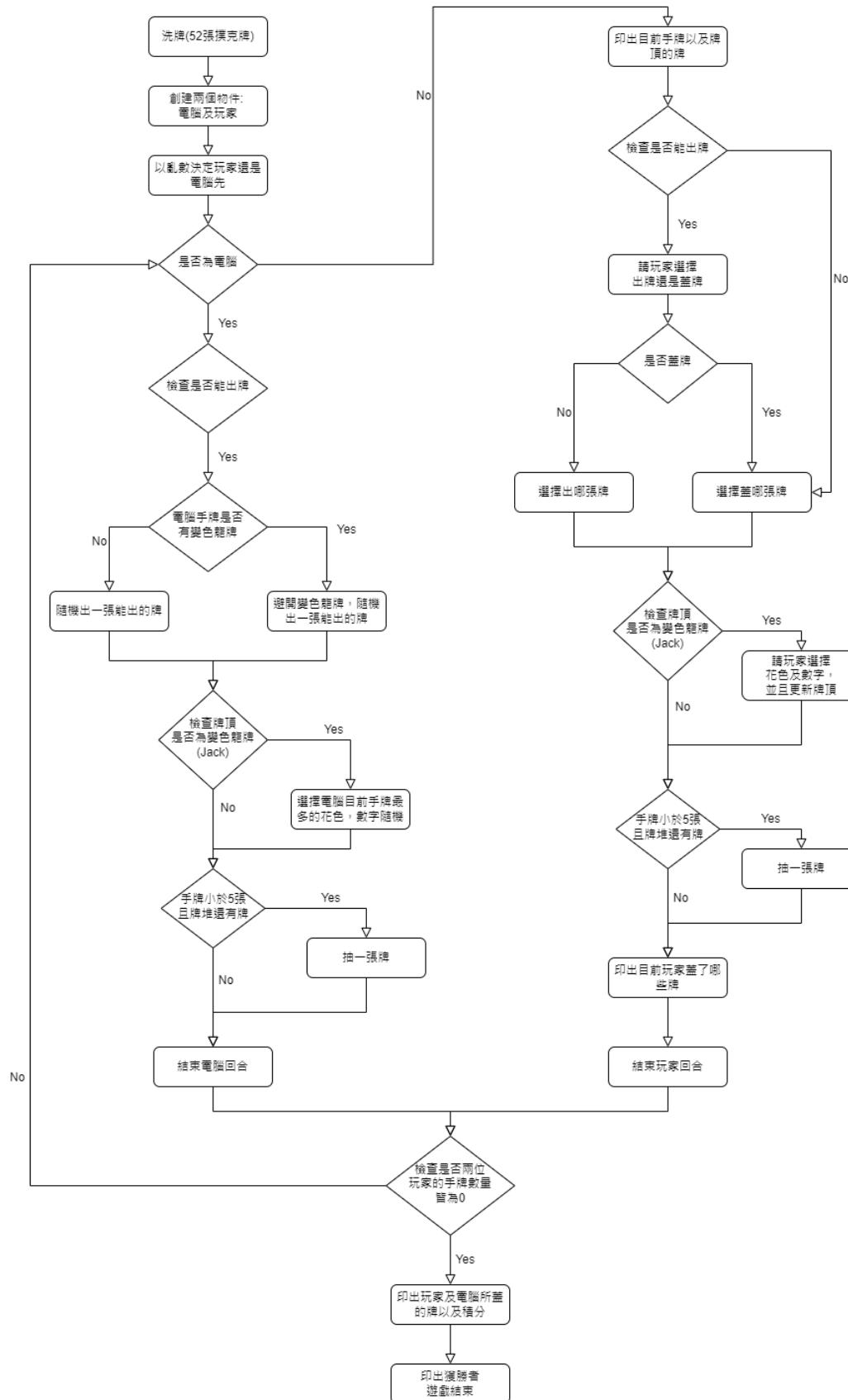
- 5.1 對照目標方塊和遊戲板上的方塊順序
- 5.2 符合就加上對應分數

函式結構圖:



遊戲部分-變色龍

流程圖：



程式虛擬碼:

```
1. 初始化變數  
while(有玩家還有手牌)  
{  
2. 玩家回合  
3. 電腦回合  
}  
4. 結算積分  
5. 印出勝負並且改變玩家戰績
```

初始化變數:

```
1.1 洗牌  
1.2 翻一張牌當作牌頂  
1.3 發手牌
```

玩家回合:

```
2.1 用方向鍵選擇出牌或蓋牌  
2.2 用方向鍵選擇哪張牌  
if (出了變色龍牌)  
{  
2.3 請玩家選擇花色及數字  
}  
2.4 改變頂牌及抽牌堆
```

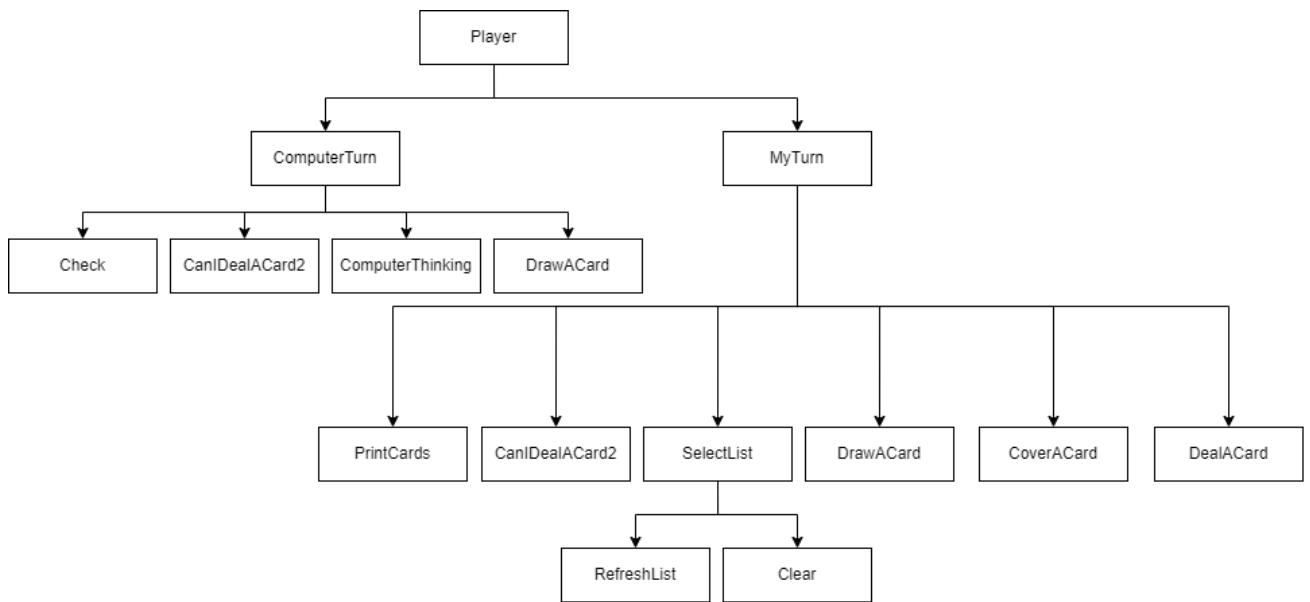
電腦回合;

```
3.1 能出牌就出牌(避開變色龍牌)  
3.2 蓋牌時選擇數字最小的蓋  
if (出了變色龍牌)  
{  
3.3 選擇對自己最有利的花色，數字隨機  
}  
3.4 改變頂牌及抽牌堆
```

結算積分:

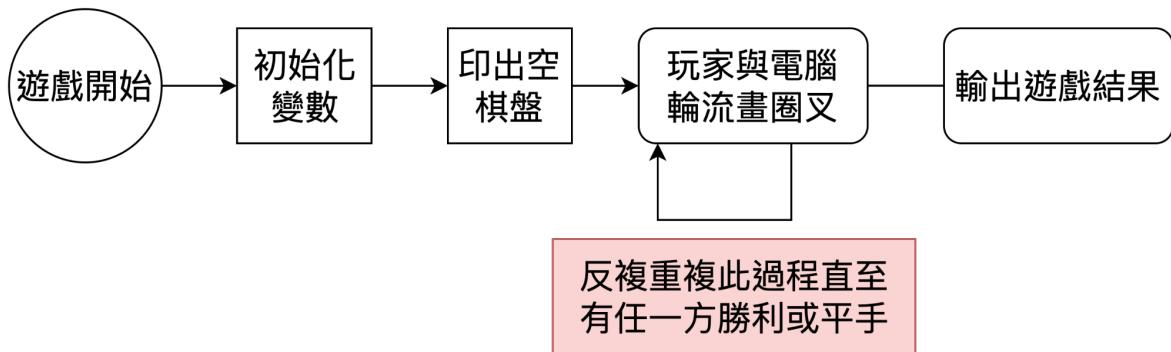
```
4.1 計算電腦及玩家的積分
```

結構圖：



遊戲部分-智慧版井字遊戲

流程圖：



程式虛擬碼：

- 1. 初始化變數**
- 2. 展示目前棋盤**

while(是否有玩家或電腦勝利或平手)

{

- 3. 玩家的回合**

- 4. 電腦的回合**

}

- 5. 輸出遊戲結果**

初始化變數

- 1.1 char[,] board = 棋盤大小(3x3)**
- 1.2 int totalTurn = 目前已下格子數**
- 1.3 bool userWin = 玩家獲勝與否**
- 1.4 bool computerWin = 電腦獲勝與否**
- 1.5 int[] oCount = 在任一方向(直、橫、斜)已放置o數量**
- 1.6 int[] xCount = 在任一方向(直、橫、斜)已放置x數量**

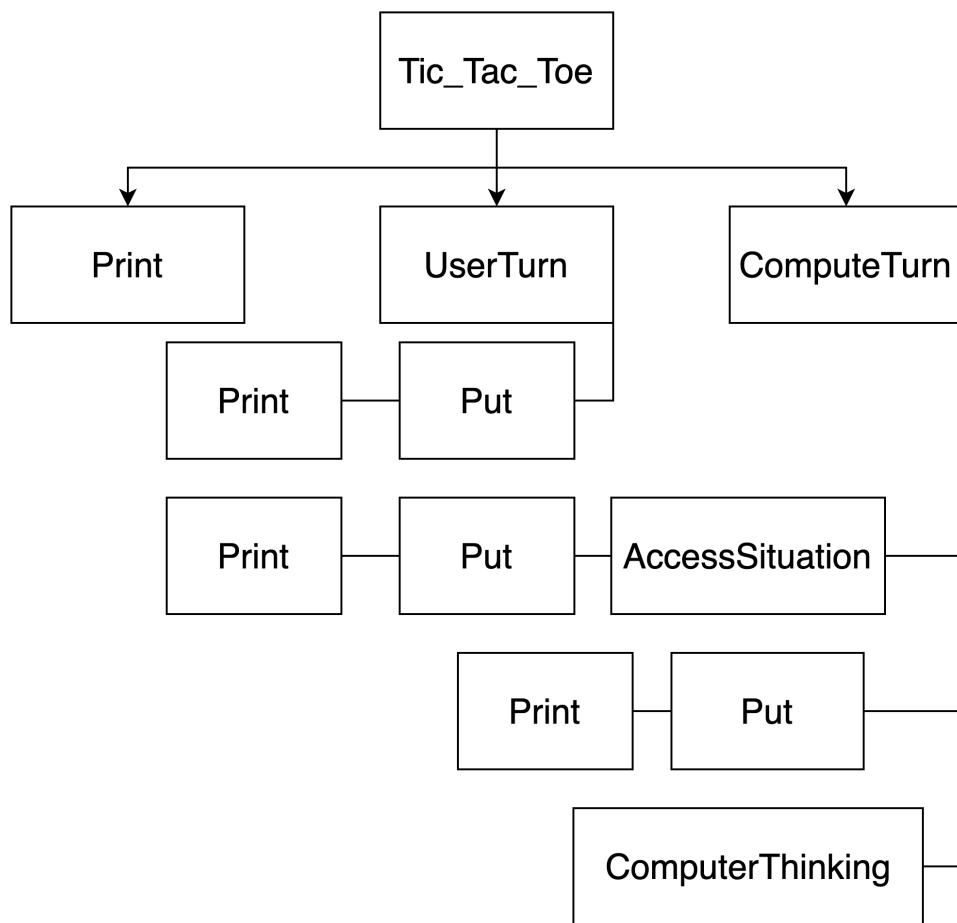
玩家的回合

- 3.1 檢查玩家輸入格式**
- 3.2 於相應格子畫上圈圈並輸出棋盤**

電腦的回合

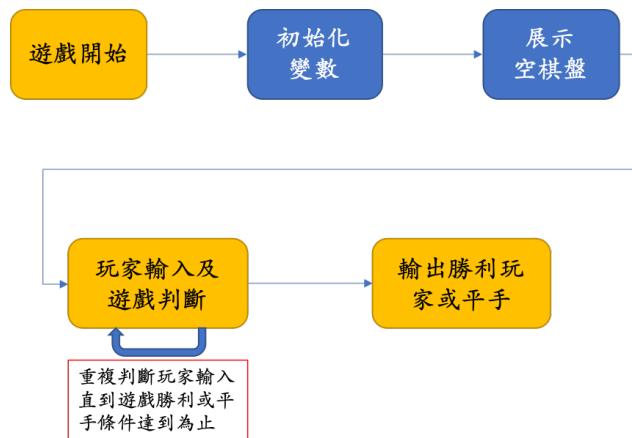
- 4.1 totalTurnNow = 這回合已下格子數**
- 4.2 判斷有沒有差一顆就贏之格子能選擇**
- 4.3 判斷玩家的棋子有沒有差一個就贏(要擋住)**
- 4.4 若這回合還沒下，則判斷是否能下在中間格子**
- 4.5 若這回合還沒下，則隨機選一格子下**

函式結構圖：



遊戲部分-四子棋

流程圖：



程式虛擬碼：

```

1. 初始化變數
2. 展示空棋盤
do
{
    3. 確認棋盤全空
    4. 玩家放入棋子
    5. 判斷放入那行有沒有滿
    6. 確認勝利與否
    7. 變更玩家
} while(是否有玩家勝利)
8. 輸出遊戲結果(勝利or平手)

```

初始化變數

1.1 int nowplayer = 定義目前玩家
1.2 char[,] checkerboard = 棋盤大小(7 行 6 列)
1.3 char[] player_piece = 定義棋子的樣式
1.4 char put_piece = 定義目前玩家的棋子
1.5 bool check_full = 確認玩家放入的那一行是否已滿
1.6 bool is_board_all_full = 確認棋盤滿格與否
1.6 bool is_win = 確認玩家勝利與否
1.7 int winner = 勝利之玩家

確認棋盤全空

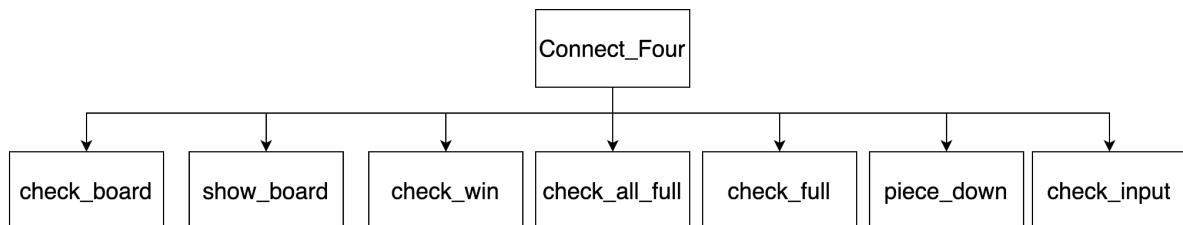
3.1 檢查每一行中之每個格子不等於空格的數量是否為總列數
3.2 有任一行有空格則代表並非全空

判斷放入那行有沒有滿(與3相似)

確認勝利與否

6.1 檢查每一個格子並分成四類
6.2 每一類各自有相應之檢查方式

函式結構圖:



五、研究過程

1.遊戲選單部分

在這個專案中，最優先考慮的是如何將每個人分工的部分結合起來，而我們其實也有考慮過其他整合模式。最初想到的最簡單構想便是讓玩家以固定順序將每個遊戲玩過一遍，都玩完後程式便會結束並顯示戰績，但這個方法給玩家的自由度低、即使只想玩其中一款遊戲也得將程式完整跑完一遍。於是又經過幾次討論後才有了現在方案的雛型，不但可以自由選擇遊戲，額外製作的遊戲選單也讓整個專案的介面多了一分類似從前Game-Boy等復古遊戲機的味道。

2.遊戲部分-推倒提基

當決定好要製作遊戲後，考量到作業時限和自己的能力，我並不打算從無到有構想一個遊戲，而是試著將現有的遊戲改為能在電腦上遊玩的版本，而由於我又恰好是個桌遊愛好者，最後便往改編桌遊的方向去思考了。在實際製作的過程中值得一提的優化有幾點。

首先是在最初的版本中，我是以一維陣列的資料結構來儲存各個提基目前的位置，但隨後便發現若要頻繁進行元素位置的更動，使用List似乎會更加容易，於是參考網路上的資源學習了基本的List操作，這才建立了比較適合儲存提基的資料結構。

再來便是在畫面顯示的部分，在顯示目前提基的順序時，我原本是將桌遊版本中提基真正的名字(如:HUHU、NUI)排列在畫面上，但隨後發現如此要辨識各個提基變得非常不容易，才把提基的名字都改成'■'字元，並用改變字元顏色的方式來作區別；還有一開始程式並不會顯示電腦執行的動作，最後也不會公布電腦的目標，而是遊戲結束後直接印出比分和勝負，讓玩家對於目前狀況經常是滿頭霧水，後來對這些動作與目標進行顯示後，才讓玩家不會連輸贏的原因都不明不白的。

遊戲部分-智慧版井字遊戲

在看完老師做了第一個版本的井字遊戲後，我便立刻想做出能像真人一樣"思考"的電腦來玩井字遊戲。我是以正常人的角度來設計電腦的下法，因此電腦會自動判斷自己差一顆就贏或是玩家差一顆就贏來下對應的位置。假如中間是空的，電腦會優先下中間。如果中間已經被佔走了且沒有人即將要贏時，電腦才會亂數決定下的位置。

在console中的井字格子要玩家手動輸入才能得知玩家要下的位置，雖然已經有提示，但玩家仍可能會把兩個軸的順序搞反，造成不小心下錯位置的問題。也許未來學了視窗化設計後，能把這個遊戲做成有獨立視窗能玩，如此一來，玩家便不需要用輸入的方式輸入位置，能直接用點的，減少錯誤率。

遊戲部分-變色龍

變色龍遊戲我嘗試用class物件來寫，因為一個玩家以一個物件來代表再也適合不過。一個玩家擁有自己的手牌、蓋牌區(成員變數)且可以出牌、蓋牌(成員方法)，所有玩家共有的是一開始的那副撲克牌還有中間的頂牌。

中間遇到最困難的地方是要電腦判斷自己該出什麼牌或蓋什麼牌，我不想直接亂數決定，這樣電腦太笨了。所以我讓電腦有變色龍牌(Jack)就先留著，直到沒牌出的時候再出變色龍牌。在蓋牌的時候我也讓電腦選擇數字最小的牌去蓋，如此能增加獲勝機率。

遊戲部分-四子棋

在實作四子棋的過程中，首先如何表示棋盤是我一開始在思考的事情。在上課過程中有很類似的例子，就是使用二維陣列，也因此這次的四子棋我也使用一樣的方式。而棋盤的序列從左上方開始(代表陣列[0,0])，與陣列之序號對應。例如陣列[0,1]代表第二行之第一格，陣列[0,3]代表第三行的第一格，依此類推。再來是考慮玩家輸入的數字，雖然可以直接預設玩家不會輸入錯誤，但是連我自己在玩有時候還真的會打錯，後來我想到之前在Java中有學過try…catch的用法，C#裡也有類似做法，因此我就採用這一種方式避免玩家輸入的不是數字。四子棋程式中有兩個比較重要的函式，分別是判斷有沒有某行已被填滿(check_board)以及勝利條件的判斷(check_win)，接下來針對這裡個部分加以說明。

判斷某行已被填滿：

一開始先設置Row變數並且假設被判斷的那一行並未被填滿，因此isRowFull為false。再來將那一行每個格子中之字串與空格作比較，若並非空格則將Row變數加一。最後檢查Row變數是否等於那一行中總格子數，若等於則isRowFull設成true表示此行已滿;反之則不變(原本就是false)。

判斷放入某行數是否已滿

```
Function check_board(char[,] checkerboard, int player_put_position)
{
    bool isRowFull = false;
    int Row = 0;
    for (i ← 0 to 行數){
        if checkerboard[i, player_put_position] != ''
            Row = Row + 1;
    }
    if (Row == 6)
    {
        isRowFull = true;
    }
    return isRowFull;
}
```

勝利條件的判斷：

對於勝利之條件，我會跑過棋盤中之每一個格子並將他們分類，一共分為四種Case。

由圖示可以清楚地看出哪些格子被分到哪種類別。

```
.....
for (i ← 0 to 列數){
    for (j ← 0 to 行數){
        int check_win_case;
        if 7 - j ≥ 4 & 6 - i ≥ 4 check_win_case = 1;
        else if 7 - j ≥ 4 & 6 - i < 4 check_win_case = 2;
        else if 7 - j < 4 & 6 - i ≥ 4 check_win_case = 3;
        else check_win_case = 4;
    }
    return win;
}
```

這四種Case在之後會對應到不同的檢查條件，去檢查從那一個格子算起，往後、往下、斜上或斜下四個格子是否一樣且不是空格。而要往哪個方向檢查則在各自的Case中不同，詳細說明如下圖。

```
switch (check_win_case)
{
    case 1:
        檢查從checkerboard[i, j]往右,往下及往右下四個格子是否一樣且不等於空格。
    case 2:
        檢查從checkerboard[i, j]往右及往右上四個格子是否一樣且不等於空格。
    case 3:
        檢查從checkerboard[i, j]往下及往左下四個格子是否一樣且不等於空格。
    case 4:
        不需檢查。
    break;
}
```

六、實作結果與討論

在團隊的分工下，我們成功達成一開始所期望的基本要求，也就是在主控台上運行一個能夠以多種小遊戲對決、並計算勝負的程式。並且以此為基礎，不論小遊戲的防呆、遊戲流程和介面等部分都進行了幾個版本的優化。

雖然順利做出符合原本期望的成果，但程式的許多部分還有可以改進的空間。例如不少函式的使用上，若再投注更多時間思考，應該還能再寫得更精簡；以及由於我們製作的小遊戲有好幾個都是讓玩家和電腦對戰，電腦對手的行動理所當然是相當重要的部分，若能讓電腦的行動更接近真人玩家的邏輯判斷，相信這些遊戲的質量還能再更提升。

七、結論

這次的實作讓我們發現要在不使用視窗化程式的情況下，要為一款遊戲做出合適的介面真的是蠻不容易的一件事。雖然我們製作的遊戲在畫面呈現上難度不算高，但要在主控台上顯示時仍有許多地方做起來有綁手綁腳的感覺，尤其在畫面更新的部分總是會遭遇許多預料之外的問題。

還有相當重要的一部分便是我們理解到團隊合作真的是左右程式開發能否順利的關鍵。即使事先已經從課堂上得知許多團隊開發的思維與關鍵，但還是得在實際操作後才能真正熟悉許多合作上的細節。以這次的程式來說，我們需要把用來代表玩家戰績的win,lose,tie三個變數傳進各個遊戲函數並做計算，但因為我們最初對於整個專案共用的變數在傳遞、命名上並沒有太多討論，導致最後要把每人負責的部分組合起來時才注意到這部分，又多花了一些時間在程式碼的整合上。

八、參考文獻

[List<T> 類別](#) Microsoft

[C# List 定義及七種常用方法](#) JOBDAREN工作達人

[Console.ForegroundColor屬性](#) Microsoft

[推倒提基\(Tiki Topple\)規則+心得](#) 瘋桌遊台中店

[四子棋規則及圖片](#) 維基百科—屏風式四子棋

[變色龍規則](#) 萌娘百科-變色龍(牌類遊戲)

九、學習心得

在完成四子棋之後有分享給我的小組成員做測試，結果還真的有一些小問題。而在構想的過程中令我最困擾的是判斷四子連起來的獲勝條件，在玩家放入某一行中時，若那一行已滿會直接變平手。這是我沒有考慮到的地方，實際的情況中，我應該要在程式中判斷每一行棋盤而不是只有檢查使用者放入的那行。

這次除了了解函式的用法及多維陣列使用之外，也學到很多不同的迴圈的使用方式，最有收穫的是do…while這一種用法。畫面的觀感本來是我們很在意的地方，不過後來老師其實也有說明，畫面只要能夠表達清楚到一個程度即可，因為以後的視窗程式設計能夠有更彈性更廣泛的發揮空間，不需要執著於主控台的畫面設計。另外，對於玩家輸入的防呆判斷，我們也學到了類似java中try…catch的簡單用法。還有continue是跳過一次迴圈的好幫手，只是以前從來沒有想過可以在何處使用。經過這一次的期末專題，我們體會到團隊合作的益處甚多，像是在四子棋遊戲的最初版本完成後，有分享給所有小組成員幫忙做測試，就又找出了一些原先沒有考慮到的情況。有隊友一起討論專案並且一起學習，能夠思考出的想法確實比自己埋頭苦思還要來得多。

十、附錄



```
namespace finalproject
{
    class Program
    {
        static void Main(string[] args)
        {
            const int gameQuantity = 4;
            string[] gameList = new string[] { "game1-推倒提基", "game2-變色龍", "game3-圈圈叉叉", "game4-四子棋" };
            int gameSelect = 0;
            int win = 0, lose = 0, tie = 0;
            bool quit = false;

            RefreshList(gameList, gameSelect, gameQuantity, win, lose, tie);
            while (!quit)
            {
                switch (Console.ReadKey().Key)
                {
                    case ConsoleKey.UpArrow:
                        gameSelect--;
                        if (gameSelect < 0) gameSelect = gameQuantity - 1;
                        RefreshList(gameList, gameSelect, gameQuantity, win, lose, tie);
                        break;

                    case ConsoleKey.DownArrow:
                        gameSelect++;
                        if (gameSelect > gameQuantity - 1) gameSelect = 0;
                        RefreshList(gameList, gameSelect, gameQuantity, win, lose, tie);
                        break;

                    case ConsoleKey.Enter:
                        switch (gameSelect)
                        {
                            case 0:
                                Console.Clear();
                                Tiki_Tipple(ref win, ref lose, ref tie);
                                Console.WriteLine("按下enter鍵再玩一次；按方向鍵選擇其他遊戲；按下esc鍵結束程序");
                                break;
                            case 1:
                                Console.Clear();
                                Chameleon(ref win, ref lose, ref tie);
                                Console.WriteLine("按下enter鍵再玩一次；按方向鍵選擇其他遊戲；按下esc鍵結束程序");
                                break;
                            case 2:
                                Console.Clear();
                                Tic_Tac_Toe(ref win, ref lose, ref tie);
                                Console.WriteLine("按下enter鍵再玩一次；按方向鍵選擇其他遊戲；按下esc鍵結束程序");
                                break;
                            case 3:
                                Console.Clear();
                                Connect_Four(ref win, ref lose, ref tie);
                                Console.WriteLine("按下enter鍵再玩一次；按方向鍵選擇其他遊戲；按下esc鍵結束程序");
                                break;
                            default:
                                break;
                        }
                        break;

                    case ConsoleKey.Escape:
                        quit = true;
                        break;

                    default:
                        break;
                }
            }
            Console.WriteLine("遊戲結束，下次見");
        }

        static void RefreshList(string[] gameList, int gameSelect, int gameQuantity, int win, int lose, int tie)
        {
            Console.Clear();
            Console.WriteLine("目前戰績{0}勝{1}敗{2}平手", win, lose, tie);
            Console.WriteLine("請選擇要玩的遊戲");
            int i;
            for(i = 0; i < gameQuantity; i++)
            {
                if(i == gameSelect)
                {
                    Console.BackgroundColor = ConsoleColor.White;
                    Console.ForegroundColor = ConsoleColor.Black;
                    Console.WriteLine(gameList[i]);
                    Console.BackgroundColor = ConsoleColor.Black;
                    Console.ForegroundColor = ConsoleColor.White;
                }
                else Console.WriteLine(gameList[i]);
            }
        }
    }
}
```

```

//*****推倒堤基*****
static void TikiTopple(ref int win, ref int lose, ref int tie)
{
    List<string> Tiki = new List<string>() {"紅色", "咖啡色", "黃色", "綠色", "淺藍色", "深藍色", "紫色"};
    int tikiLength = Tiki.Count;

    List<string> playerHand = new List<string>() { "上移三層", "上移一層", "上移一層", "移到最下層",
    "移除最底部的方塊" };
    List<string> comHand = new List<string>() { "上移三層", "上移一層", "上移一層", "移到最下層", "移
    除最底部的方塊" };
    int cardQuantity = 5;

    int playerScore = 0, comScore = 0;

    List<string> playerGoal = new List<string>(); //設定目標堤基
    List<string> comGoal = new List<string>();
    Random rand = new Random();
    PickPlayerGoal(rand, Tiki, tikiLength, ref playerGoal);
    PickComGoal(rand, Tiki, tikiLength, ref comGoal);

    while (cardQuantity != 0)
    {
        ComPlay(rand, comHand, comGoal, playerGoal, ref Tiki);
        PlayerPlay(playerGoal, ref playerHand, ref cardQuantity, ref Tiki);
    }

    Console.WriteLine("電腦的目標是{0}、{1}、{2}\n", comGoal[0], comGoal[1], comGoal[2]);
    CountFinalScore(Tiki, playerGoal, out playerScore);
    CountFinalScore(Tiki, comGoal, out comScore);
    if (playerScore > comScore)
    {
        Console.WriteLine("玩家{0}分、電腦{1}分 · 你贏了!", playerScore, comScore);
        win++;
    }
    else if (playerScore < comScore)
    {
        Console.WriteLine("玩家{0}分、電腦{1}分 · 你輸了...", playerScore, comScore);
        lose++;
    }
    else
    {
        Console.WriteLine("玩家{0}分、電腦{1}分 · 平手", playerScore, comScore);
        tie++;
    }
}

//設定目標堤基
static void PickPlayerGoal(Random rand, List<string> Tiki, int tikiLength, ref List<string>
playerGoal)
{
    int First, Second, Third;

    First = rand.Next(0, tikiLength);
    if (First + 2 >= tikiLength) Second = (First + 2) - tikiLength;
    else Second = First + 2;
    if (First + 4 >= tikiLength) Third = (First + 4) - tikiLength;
    else Third = First + 4;

    playerGoal.Add(Tiki[First]);
    playerGoal.Add(Tiki[Second]);
    playerGoal.Add(Tiki[Third]);
}

static void PickComGoal(Random rand, List<string> Tiki, int tikiLength, ref List<string>
comGoal)
{
    int First, Second, Third;

    First = rand.Next(0, tikiLength);
    if (First + 1 >= tikiLength) Second = (First + 1) - tikiLength;
    else Second = First + 1;
    if (First + 3 >= tikiLength) Third = (First + 3) - tikiLength;
    else Third = First + 3;
}

```

```

        comGoal.Add(Tiki[First]);
        comGoal.Add(Tiki[Second]);
        comGoal.Add(Tiki[Third]);
    }

    //電腦出牌
    static void ComPlay(Random rand, List<string> comHand, List<string> comGoal, List<string>
playerGoal, ref List<string> Tiki)
    {
        string movedTiki;
        switch (comHand[rand.Next(0, comHand.Count)])
        {
            case "上移三層":
                ComUpThree(rand, comGoal, ref Tiki, ref comHand, out movedTiki);
                Console.WriteLine("電腦對{0}方塊執行了\"上移三層\"", movedTiki);
                break;

            case "上移一層":
                ComUpOne(rand, comGoal, ref Tiki, ref comHand, out movedTiki);
                Console.WriteLine("電腦對{0}方塊執行了\"上移一層\"", movedTiki);
                break;

            case "移到最下層":
                ComToBottom(rand, playerGoal, ref Tiki, ref comHand, out movedTiki);
                Console.WriteLine("電腦對{0}方塊執行了\"移到最下層\"", movedTiki);
                break;

            case "移除最底部的方塊":
                ComRemoveBottom(ref Tiki, ref comHand);
                Console.WriteLine("電腦移除了最底部的方塊");
                break;

            default:
                break;
        }
    }

    static void ComUpThree(Random rand, List<string> comGoal, ref List<string> Tiki, ref
List<string> comHand, out string movedTiki)
    {
        int chosenGoal;
        string chosenTiki;

        do
        {
            chosenGoal = rand.Next(0, 3); //從目標堤基中選擇
            chosenTiki = comGoal[chosenGoal]; //取得目標堤基名字
        } while (!Tiki.Contains(chosenTiki)); //若名字不在遊戲板中就重選

        int tikiIndex = Tiki.IndexOf(chosenTiki); //取得目標堤基在遊戲板的索引
        Tiki.RemoveAt(tikiIndex); //從遊戲板上移除所選堤基
        if (tikiIndex - 3 < 0) Tiki.Insert(0, chosenTiki); //插入所選堤基在指定位置
        else Tiki.Insert(tikiIndex - 3, chosenTiki);

        comHand.RemoveAt(comHand.IndexOf("上移三層"));
        movedTiki = chosenTiki;
    }

    static void ComUpOne(Random rand, List<string> comGoal, ref List<string> Tiki, ref List<string>
comHand, out string movedTiki)
    {
        int chosenGoal;
        string chosenTiki;

        do
        {
            chosenGoal = rand.Next(0, 3); //從目標堤基中選擇
            chosenTiki = comGoal[chosenGoal]; //取得目標堤基名字
        } while (!Tiki.Contains(chosenTiki)); //若名字不在遊戲板中就重選

        int tikiIndex = Tiki.IndexOf(chosenTiki); //取得目標堤基在遊戲板的索引
        Tiki.RemoveAt(tikiIndex); //從遊戲板上移除所選堤基
        if (tikiIndex - 1 < 0) Tiki.Insert(0, chosenTiki); //插入所選堤基在指定位置
        else Tiki.Insert(tikiIndex - 1, chosenTiki);

        comHand.RemoveAt(comHand.IndexOf("上移一層"));
        movedTiki = chosenTiki;
    }
}

```

```

        static void ComToBottom(Random rand, List<string> playerGoal, ref List<string> Tiki, ref
List<string> comHand, out string movedTiki)
{
    int chosenGoal;
    string chosenTiki;

    do
    {
        chosenGoal = rand.Next(0, 3); //從目標堤基中選擇
        chosenTiki = playerGoal[chosenGoal]; //取得目標堤基名字
    } while (!Tiki.Contains(chosenTiki)); //若名字不在遊戲板中就重選

    int tikiIndex = Tiki.IndexOf(chosenTiki); //取得目標堤基在遊戲板的索引
    Tiki.RemoveAt(tikiIndex); //從遊戲板上移除所選堤基
    Tiki.Add(chosenTiki); //插入所選堤基到底部

    comHand.RemoveAt(comHand.IndexOf("移到最下層"));
    movedTiki = chosenTiki;
}
static void ComRemoveBottom(ref List<string> Tiki, ref List<string> comHand)
{
    Tiki.RemoveAt(Tiki.Count - 1);
    comHand.RemoveAt(comHand.IndexOf("移除最底部的方塊"));
}

//玩家出牌
static void PlayerPlay(List<string> playerGoal, ref List<string> playerHand, ref int
cardQuantity, ref List<string> Tiki)
{
    int cardSelect = 0;
    bool roundOver = false;

    RefreshScreen(Tiki, playerGoal, cardQuantity, cardSelect, playerHand);
    while (!roundOver)
    {
        switch (Console.ReadKey().Key)
        {
            case ConsoleKey.RightArrow:
                Console.Clear();
                cardSelect++;
                if (cardSelect >= cardQuantity) cardSelect = 0;
                RefreshScreen(Tiki, playerGoal, cardQuantity, cardSelect, playerHand);
                break;
            case ConsoleKey.LeftArrow:
                Console.Clear();
                cardSelect--;
                if (cardSelect < 0) cardSelect = cardQuantity - 1;
                RefreshScreen(Tiki, playerGoal, cardQuantity, cardSelect, playerHand);
                break;
            case ConsoleKey.Enter:
                switch (playerHand[cardSelect])
                {
                    //, "上移一層", "上移一層", "移到最下層", "移除最底部的方塊"
                    case "上移三層":
                        PlayerUpThree(ref Tiki);
                        playerHand.Remove("上移三層");
                        cardQuantity--;
                        Console.Clear();
                        break;
                    case "上移一層":
                        PlayerUpOne(ref Tiki);
                        playerHand.Remove("上移一層");
                        cardQuantity--;
                        Console.Clear();
                        break;
                    case "移到最下層":
                        PlayerToBottom(ref Tiki);
                        playerHand.Remove("移到最下層");
                        cardQuantity--;
                        Console.Clear();
                        break;
                    case "移除最底部的方塊":
                        Tiki.RemoveAt(Tiki.Count - 1);
                        playerHand.Remove("移除最底部的方塊");
                        break;
                }
        }
    }
}

```

```

        playerHand.Remove("移除最底部的方塊");
        cardQuantity--;
        Console.Clear();
        break;
    }
    roundOver = true;
    break;
}
}
if(cardQuantity != 0) Console.WriteLine("輪到電腦");
else
{
    Console.WriteLine("遊戲結束!");
    RefreshScreen(Tiki, playerGoal, cardQuantity, cardSelect, playerHand);
}
}
static void RefreshScreen(List<string> Tiki, List<string> playerGoal, int cardQuantity, int
cardSelect, List<string> playerHand)
{
    Console.Write("\n");
    foreach (string tiki in Tiki)
    {
        switch (tiki)
        {
            case "紅色":
                Console.ForegroundColor = ConsoleColor.Red;
                Console.WriteLine("■");
                Console.ForegroundColor = ConsoleColor.White;
                break;
            case "咖啡色":
                Console.ForegroundColor = ConsoleColor.DarkYellow;
                Console.WriteLine("■");
                Console.ForegroundColor = ConsoleColor.White;
                break;
            case "黃色":
                Console.ForegroundColor = ConsoleColor.Yellow;
                Console.WriteLine("■");
                Console.ForegroundColor = ConsoleColor.White;
                break;
            case "綠色":
                Console.ForegroundColor = ConsoleColor.Green;
                Console.WriteLine("■");
                Console.ForegroundColor = ConsoleColor.White;
                break;
            case "淺藍色":
                Console.ForegroundColor = ConsoleColor.Cyan;
                Console.WriteLine("■");
                Console.ForegroundColor = ConsoleColor.White;
                break;
            case "深藍色":
                Console.ForegroundColor = ConsoleColor.DarkBlue;
                Console.WriteLine("■");
                Console.ForegroundColor = ConsoleColor.White;
                break;
            case "紫色":
                Console.ForegroundColor = ConsoleColor.DarkMagenta;
                Console.WriteLine("■");
                Console.ForegroundColor = ConsoleColor.White;
                break;
        }
    }
    Console.WriteLine("\n{0}在第一格得5分、{1}在第二格以上得4分、{2}在第三格以上得3分", playerGoal[0],
playerGoal[1], playerGoal[2]);
    if (cardQuantity != 0) Console.WriteLine("\n請選擇你要出的牌:\n");
    int i;
    for (i = 0; i < cardQuantity; i++)
    {
        if (i == cardSelect)
        {
            Console.BackgroundColor = ConsoleColor.White;
            Console.ForegroundColor = ConsoleColor.Black;
            Console.Write("{0} ", playerHand[i]);
            Console.BackgroundColor = ConsoleColor.Black;
            Console.ForegroundColor = ConsoleColor.White;
        }
        else Console.Write("{0} ", playerHand[i]);
    }
    Console.Write("\n");
}

```

```
    static void PlayerUpThree(ref List<string> Tiki)
    {
        string target;
        int targetIndex;

        Console.WriteLine("請輸入使用對象的方塊顏色");
        do
        {
            target = Console.ReadLine();
            if (Tiki.Contains(target)) break;
            else Console.WriteLine("請輸入有效目標名稱");
        } while (true);

        targetIndex = Tiki.IndexOf(target);
        Tiki.RemoveAt(targetIndex);
        if (targetIndex - 3 < 0) Tiki.Insert(0, target);
        else Tiki.Insert(targetIndex - 3, target);
    }
    static void PlayerUpOne(ref List<string> Tiki)
    {
        string target;
        int targetIndex;

        Console.WriteLine("請輸入使用對象的方塊顏色");
        do
        {
            target = Console.ReadLine();
            if (Tiki.Contains(target)) break;
            else Console.WriteLine("請輸入有效目標名稱");
        } while (true);

        targetIndex = Tiki.IndexOf(target);
        Tiki.RemoveAt(targetIndex);
        if (targetIndex - 1 < 0) Tiki.Insert(0, target);
        else Tiki.Insert(targetIndex - 1, target);
    }
    static void PlayerToBottom(ref List<string> Tiki)
    {
        string target;

        Console.WriteLine("請輸入使用對象的方塊顏色");
        do
        {
            target = Console.ReadLine();
            if (Tiki.Contains(target)) break;
            else Console.WriteLine("請輸入有效目標名稱");
        } while (true);

        Tiki.Remove(target);
        Tiki.Add(target);
    }

    //算分數
    static int CountFinalScore(List<string> Tiki, List<string> Goal, out int score)
    {
        int sum = 0;
        if (Goal[0] == Tiki[0]) sum += 5;
        if (Goal[1] == Tiki[0] || Goal[1] == Tiki[1]) sum += 4;
        if (Goal[2] == Tiki[0] || Goal[2] == Tiki[1] || Goal[2] == Tiki[2]) sum += 3;
        score = sum;
        return score;
    }
}
```

```

//*****變色龍*****
static int[] Shuffle(int cardNum)
{
    int[] cardDeck = new int[cardNum];
    Random rnd = new Random();
    for (int i = 0; i < cardNum; i++)
    {
        do
        {
            cardDeck[i] = rnd.Next() % cardNum;
        } while (Array.IndexOf(cardDeck, cardDeck[i]) != i);
    }
    return cardDeck;
}
static void Chameleon(ref int win, ref int lose, ref int tie)
{
    int cardNum = 52; //牌的數量
    int[] cardDeck = Shuffle(cardNum); //洗好的牌 (固定)
    int currentCard = 0; //現在要抽第幾張卡 (浮動)
    Player.TopCard = cardDeck[currentCard];
    currentCard++;
    Player computer = new Player(cardDeck, ref currentCard, "computer", 0);
    Player user = new Player(cardDeck, ref currentCard, "user", 1);
    Player[] player_array = new Player[2];
    player_array[0] = computer;
    player_array[1] = user;
    Random rnd = new Random();
    int currentPlayer = rnd.Next() % player_array.Length; //亂數決定第一個玩家
    int howManyPlayersAlive = player_array.Length;
    Console.WriteLine("規則說明:\n1.出牌者必須打出與上一張打出的牌有同樣花色或點數的牌或是變色龍牌。" +
        "\n2.四種花色的J為變色龍牌。出牌者可以無視上一張打出的牌的花色或點數，並且選擇這張牌代表的花色和點數。" +
        "\n3.若出牌者無牌可出或是不想出牌，則從手牌中選一張背面朝上蓋在自己面前。" +
        "\n4.無論出牌還是蓋牌，都必須立即摸牌，將自己的手牌補滿5張(若牌堆無牌可摸，則不摸牌)。" +
        "\n5.當全部玩家皆無手牌時，本局遊戲結束，開始計分。" +
        "\n6.所有玩家將自己的蓋牌亮出，牌面點數相加即為自己該局的積分，積分較低者獲勝。" +
        "\n\n按Enter鍵以繼續....");
    Console.ReadLine();
    Player.Clear(2);
    Console.WriteLine("\n變色龍遊戲開始!!!!");

    while (howManyPlayersAlive != 0)
    {
        player_array[currentPlayer].MyTurn(cardDeck, ref currentCard);
        if (player_array[currentPlayer].HandCardNum == 0)
        {
            howManyPlayersAlive--;
        }
        currentPlayer++;
        if (currentPlayer >= player_array.Length)
            currentPlayer = 0;
    }

    Console.WriteLine();
    int[] scores = new int[player_array.Length]; //計分
    int min = 100000;
    for (int i = 0; i < player_array.Length; i++)
    {
        int score = 0;
        for (int j = 0; j < player_array[i].coveredCard.Count; j++)
            score += player_array[i].coveredCard[j] % 13 + 1;
        scores[i] = score;
        if (scores[i] < min)
            min = scores[i];
        Console.WriteLine("玩家 {0} 積分: {1}", player_array[i].Name, scores[i]);
        player_array[i].PrintCards("已蓋的牌", player_array[i].coveredCard);
        Console.WriteLine();
    }

    Player winner = player_array[Array.IndexOf(scores, min)];
    if (scores[0] == scores[1])
        tie++;
    else if (winner.PlayerId == 0)
        lose++;
    else
        win++;
    Console.WriteLine("\n恭喜 {0} 獲勝!!!", winner.Name);
}

```

```

//*****圈叉*****
static void Print(char[,] board)
{
    Console.WriteLine("\n  0  1  2");
    Console.WriteLine("      ");
    Console.WriteLine("0 {0} | {1} | {2}", board[0, 0], board[0, 1], board[0, 2]);
    Console.WriteLine(" ---+---+---");
    Console.WriteLine("1 {0} | {1} | {2}", board[1, 0], board[1, 1], board[1, 2]);
    Console.WriteLine(" ---+---+---");
    Console.WriteLine("2 {0} | {1} | {2}\n", board[2, 0], board[2, 1], board[2, 2]);
}
static void UserTurn(ref char[,] board, ref int[] oCount, ref int totalTurn) //user: o
{
    Console.Write("換你下(請先輸入左方座標，再輸入上方做標，並以逗號隔開): ");
    int i = 0;
    int j = 0;
    bool successPut = false;
    do
    {
        do
        {
            string[] userInput;
            try
            {
                userInput = Console.ReadLine().Split(',');
                i = int.Parse(userInput[0]);
                j = int.Parse(userInput[1]);
                successPut = true;
            }
            catch
            {
                Console.WriteLine("輸入格式錯誤，請重新輸入!");
                successPut = false;
                continue;
            }
        } while (!successPut);
        successPut = false;
        if ((i > 2 || i < -1) || (j > 2 || j < 0))
        {
            Console.WriteLine("無此格，請下另一格!");
            continue;
        }
        else if (board[i, j] != ' ')
            Console.WriteLine("此格已被填滿，請下另一格!");
        else
            successPut = true;
    } while (!successPut);
    Put(ref board, ref oCount, i, j, 'o', ref totalTurn);
}
static void ComputerTurn(ref char[,] board, ref int[] xCount, ref int totalTurn, int[] oCount)
//computer: x
{
    Console.WriteLine("換電腦下:");
    ComputerThinking();
    ComputerThinking();
    int totalTurnNow = totalTurn;
    //電腦判斷1：電腦差一顆就贏
    for (int i = 0; i < 8; i++)
    {
        bool isVertical = false;
        if (xCount[i] == 2 && oCount[i] == 0)
        {
            AccessSituation(i, ref isVertical, ref board, ref xCount, ref totalTurn);
            if (isVertical)
                i += 3;
        }
        if (totalTurn == totalTurnNow + 1) //擋一顆就好
            break;
    }
    //電腦判斷2：玩家差一顆就贏時，電腦要擋
    for (int i = 0; i < 8; i++)
    {
        bool isVertical = false;
    }
}

```

```

        if (oCount[i] == 2 && xCount[i] == 0)
        {
            AccessSituation(i, ref isVertical, ref board, ref xCount, ref totalTurn);
            if (isVertical)
                i += 3;
        }
        if (totalTurn == totalTurnNow + 1) //擋一顆就好
            break;
    }
    //電腦判斷2：如果中間是空的，就下中間
    if (totalTurn == totalTurnNow && board[1, 1] == ' ')
        Put(ref board, ref xCount, 1, 1, 'x', ref totalTurn);
    //電腦判斷3：隨機下
    if (totalTurn == totalTurnNow)
    {
        Random rnd = new Random();
        do
        {
            int key = rnd.Next() % 9;
            int i = key / 3;
            int j = key % 3;
            if (board[i, j] == ' ')
                Put(ref board, ref xCount, i, j, 'x', ref totalTurn);
        } while (totalTurn == totalTurnNow);
    }
}
static void Put(ref char[,] board, ref int[] count, int i, int j, char xo, ref int totalTurn)
//下一顆棋
{
    board[i, j] = xo;
    count[i]++;
    count[j + 3]++;
    if (i == j)
        count[6]++;
    if (i + j == 2)
        count[7]++;
    totalTurn++;
    Print(board);
}
static void AccessSituation(int i, ref bool isVertical, ref char[,] board, ref int[] xCount,
ref int totalTurn) //專門給ComputerTurn用來判斷哪一行差一個就要贏了，並且下在正確位置
{
    if (i >= 3 && i <= 5) //直的
    {
        i -= 3;
        isVertical = true;
    }
    for (int j = 0; j < 3; j++)
    {
        switch (i)
        {
            case 0:
            case 1:
            case 2:
                if (isVertical == false && board[i, j] == ' ') //橫的判斷
                {
                    Put(ref board, ref xCount, i, j, 'x', ref totalTurn);
                    break;
                }
                if (isVertical == true && board[j, i] == ' ') //直的判斷
                {
                    Put(ref board, ref xCount, j, i, 'x', ref totalTurn);
                    break;
                }
                break;
            case 6: //斜的判斷：\
                if (board[j, j] == ' ')
                {
                    Put(ref board, ref xCount, j, j, 'x', ref totalTurn);
                    break;
                }
                break;
            case 7: //斜的判斷：/
                if (board[j, 2 - j] == ' ')
                {
                    Put(ref board, ref xCount, j, 2 - j, 'x', ref totalTurn);
                    break;
                }
                break;
        }
    }
}

```

```
        break;
    default:
        Console.WriteLine("錯誤訊息");
        break;
    }
}
static void ComputerThinking() //純粹只是讓電腦看起來有在思考
{
    Console.WriteLine();
    Thread.Sleep(700);
    for (int i = 0; i < 3; i++)
    {
        Console.Write(".");
        Thread.Sleep(500);
    }
    Console.SetCursorPosition(0, Console.CursorTop);
    Console.Write(new string(' ', Console.WindowWidth));
    Console.SetCursorPosition(0, Console.CursorTop - 1);
}
static void Tic_Tac_Toe(ref int win, ref int lose, ref int tie) //主程式在這!!!
{
    char[,] board =
    {
        { ' ', ' ', ' ' },
        { ' ', ' ', ' ' },
        { ' ', ' ', ' ' }};
    int totalTurn = 0;
    bool userWin = false;
    bool computerWin = false;
    int[] oCount = { 0, 0, 0, 0, 0, 0, 0, 0 }; //row0, row1, row2, col1, col2, col3, \, /
    int[] xCount = { 0, 0, 0, 0, 0, 0, 0, 0 };
    Print(board);

    while (!(userWin || computerWin || (totalTurn >= 9))) //結束遊戲條件是其中一方贏或9格下滿
    {
        UserTurn(ref board, ref oCount, ref totalTurn);
        if (Array.IndexOf(oCount, 3) >= 0)
            userWin = true;
        if (userWin || (totalTurn >= 9)) break;
        ComputerTurn(ref board, ref xCount, ref totalTurn, oCount);
        if (Array.IndexOf(xCount, 3) >= 0)
            computerWin = true;
    }

    if (userWin)
    {
        Console.WriteLine("\n你贏了");
        win++;
    }
    else if (computerWin)
    {
        Console.WriteLine("\n你輸了");
        lose++;
    }
    else
    {
        Console.WriteLine("\n平手");
        tie++;
    }
}
```

```
//*****四子棋*****
static void Connect_Four(ref int win, ref int lose, ref int tie)
{
    Console.WriteLine("歡迎來到四子棋!");
    Console.WriteLine("遊戲規則:");
    Console.WriteLine("1. 棋盤大小為7x6。");
    Console.WriteLine("2. 雙方必須輪流把一枚己棋選擇投入任一行，讓棋子落在在底部或其他棋子上。");
    Console.WriteLine("3. 玩家輸入1~7代表選擇將棋放入哪一行。");
    Console.WriteLine("4. 當己方4枚棋子以縱、橫、斜方向連成一線時獲勝。");
    Console.WriteLine("5. 棋盤滿棋時，無任何連成4子，則平手。");
    Console.WriteLine("遊戲開始!");

    //定義玩家
    int nowplayer = 1;
    string[] playername = new string[] { "主玩家", "謫館者" };
    //定義棋盤及棋子
    char[,] checkerboard = new char[6, 7];
    char[] player_piece = { 'o', 'x' };
    char put_piece = player_piece[nowplayer];
    //定義變數
    bool check_full = false;
    bool is_board_all_full = false;
    bool is_win = false;
    int winner = 0;
    int player_put_position;
    //展示棋盤
    Console.WriteLine(" 1 2 3 4 5 6 7 ");
    for (int i = 0; i < checkerboard.GetUpperBound(0) + 1; i++)
    {
        Console.WriteLine("-----+");
        Console.Write("|");
        for (int j = 0; j < checkerboard.GetUpperBound(1) + 1; j++)
        {
            checkerboard[i, j] = ' ';
            Console.Write(" {0} |", checkerboard[i, j]);
        }
        Console.WriteLine("");
    }
    Console.WriteLine("-----+");
    Console.WriteLine(" 1 2 3 4 5 6 7 ");

    do
    {
        //確認棋盤空位
        is_board_all_full = check_all_full(checkerboard);
        if (is_board_all_full)
        {
            break;
        }
        //玩家放入旗子
        Console.WriteLine("目前玩家:{0}", playername[nowplayer - 1]);
        Console.WriteLine("輸入放入第幾行(1~7)");
        try
        {
            player_put_position = int.Parse(Console.ReadLine()) - 1;
        }
        catch
        {
            Console.WriteLine("輸入錯誤！請重新輸入1~7行：");
            continue;
        }
        //判斷輸入格式是否為數字
        if (!check_input(player_put_position))
        {
            Console.WriteLine("輸入錯誤！請重新輸入1~7行：");
            continue;
        }
        //判斷放入的欄位有沒有滿
        check_full = check_board(checkerboard, player_put_position);
        if (!check_full)
        {
            piece_down(ref checkerboard, put_piece, player_put_position);
            show_board(checkerboard);
        }
    }
}
```

```

        else
        {
            Console.WriteLine("這一行滿了!請換一行!");
            continue;
        }
        //確認勝利與否
        is_win = check_win(checkerboard);
        if (is_win)
        {
            winner = nowplayer;
        }
        //改變玩家
        if (nowplayer == 1)
        {
            nowplayer = 2;
            put_piece = player_piece[1];
        }
        else
        {
            nowplayer = 1;
            put_piece = player_piece[0];
        }
    } while (!is_win);
    if (is_win)
    {
        Console.WriteLine("{0}獲勝!", playername[winner - 1]);
        if (winner == 1) win++;
        else if (winner == 2) lose++;
    }
    else if (is_board_all_full)
    {
        Console.WriteLine("此局平手!");
        tie++;
    }
}

private static bool check_all_full(char[,] checkerboard) //確認全部棋盤空格
{
    bool isRowFull = true;
    int Row = 0;
    for (int j = 0; j < checkerboard.GetUpperBound(1) + 1; j++)
    {
        for (int i = 0; i < checkerboard.GetUpperBound(0) + 1; i++)
        {
            if (checkerboard[i, j] != ' ')
            {
                Row = Row + 1;
            }
        }
        if (Row != 6)
        {
            isRowFull = false;
            return isRowFull;
        }
        Row = 0;
    }
    return isRowFull;
}

private static bool check_win(char[,] checkerboard) //確認有無四個連續棋子(獲勝之條件)
{
    bool win = false;
    for (int i = 0; i < checkerboard.GetUpperBound(0) + 1; i++)
    {
        for (int j = 0; j < checkerboard.GetUpperBound(1) + 1; j++)
        {
            int check_win_case;
            if (7 - j >= 4 & 6 - i >= 4)
            {
                check_win_case = 1;
            }
            else if (7 - j >= 4 & 6 - i < 4)
            {
                check_win_case = 2;
            }
            else if (7 - j < 4 & 6 - i >= 4)
            {
                check_win_case = 3;
            }
        }
    }
}

```

```

        else
        {
            check_win_case = 4;
        }

        switch (check_win_case)
        {
            case 1:
                if (checkerboard[i, j] == checkerboard[i + 1, j] & checkerboard[i, j] ==
checkerboard[i + 2, j] & checkerboard[i, j] == checkerboard[i + 3, j] & checkerboard[i, j] != ' ')
                {
                    win = true;
                    return win;
                }
                else if (checkerboard[i, j] == checkerboard[i, j + 1] & checkerboard[i, j] ==
checkerboard[i, j + 2] & checkerboard[i, j] == checkerboard[i, j + 3] & checkerboard[i, j] != ' ')
                {
                    win = true;
                    return win;
                }
                else if (checkerboard[i, j] == checkerboard[i + 1, j + 1] & checkerboard[i,
j] == checkerboard[i + 2, j + 2] & checkerboard[i, j] == checkerboard[i + 3, j + 3] & checkerboard[i,
j] != ' ')
                {
                    win = true;
                    return win;
                }
                else
                {
                    win = false;
                    break;
                }
            case 2:
                if (checkerboard[i, j] == checkerboard[i, j + 1] & checkerboard[i, j] ==
checkerboard[i, j + 2] & checkerboard[i, j] == checkerboard[i, j + 3] & checkerboard[i, j] != ' ')
                {
                    win = true;
                    return win;
                }
                else if (checkerboard[i, j] == checkerboard[i - 1, j + 1] & checkerboard[i,
j] == checkerboard[i - 2, j + 2] & checkerboard[i, j] == checkerboard[i - 3, j + 3] & checkerboard[i,
j] != ' ')
                {
                    win = true;
                    return win;
                }
                else
                {
                    win = false;
                    break;
                }
            case 3:
                if (checkerboard[i, j] == checkerboard[i + 1, j] & checkerboard[i, j] ==
checkerboard[i + 2, j] & checkerboard[i, j] == checkerboard[i + 3, j] & checkerboard[i, j] != ' ')
                {
                    win = true;
                    return win;
                }
                else if (checkerboard[i, j] == checkerboard[i + 1, j - 1] & checkerboard[i,
j] == checkerboard[i + 2, j - 2] & checkerboard[i, j] == checkerboard[i + 3, j - 3] & checkerboard[i,
j] != ' ')
                {
                    win = true;
                    return win;
                }
                else
                {
                    win = false;
                    break;
                }
            case 4:
                break;
        default:
            Console.WriteLine("發生例外狀況");
            break;
    }
}

```



```

namespace FinalProject
{
    class Player
    {
        private string name;
        private readonly int playerId; //readonly和const - 不過readonly不須提供初值，可在變數中賦值
        private List<int> handCard = new List<int>(); //用數形式存手牌，較好判斷數字與花色。印出來時再用
    }

    dictionary類型文字
    public List<int> coveredCard = new List<int>();
    private static int topCard;
    private static Dictionary<int, string> Int2poker = new Dictionary<int, string>()
    {
        { 0, "Ace of Spades" }, { 1, "2 of Spades" }, { 2, "3 of Spades" }, { 3, "4 of Spades" }, { 4, "5 of Spades" }, { 5, "6 of Spades" }, { 6, "7 of Spades" }, { 7, "8 of Spades" }, { 8, "9 of Spades" }, { 9, "10 of Spades" }, { 10, "Jack of Spades" }, { 11, "Queen of Spades" }, { 12, "King of Spades" },
        { 13, "Ace of Hearts" }, { 14, "2 of Hearts" }, { 15, "3 of Hearts" }, { 16, "4 of Hearts" }, { 17, "5 of Hearts" }, { 18, "6 of Hearts" }, { 19, "7 of Hearts" }, { 20, "8 of Hearts" }, { 21, "9 of Hearts" }, { 22, "10 of Hearts" }, { 23, "Jack of Hearts" }, { 24, "Queen of hearts" }, { 25, "King of Hearts" },
        { 26, "Ace of Diamonds" }, { 27, "2 of Diamonds" }, { 28, "3 of Diamonds" }, { 29, "4 of Diamonds" }, { 30, "5 of Diamonds" }, { 31, "6 of Diamonds" }, { 32, "7 of Diamonds" }, { 33, "8 of Diamonds" }, { 34, "9 of Diamonds" }, { 35, "10 of Diamonds" }, { 36, "Jack of Diamonds" }, { 37, "Queen of Diamonds" },
        { 38, "King of Diamonds" }, { 39, "Ace of clubs" }, { 40, "2 of Clubs" }, { 41, "3 of Clubs" }, { 42, "4 of Clubs" }, { 43, "5 of Clubs" }, { 44, "6 of Clubs" }, { 45, "7 of Clubs" }, { 46, "8 of Clubs" }, { 47, "9 of Clubs" }, { 48, "10 of Clubs" }, { 49, "Jack of Clubs" }, { 50, "Queen of clubs" }, { 51, "King of Clubs" };
    };

    public Player(int[] cardDeck, ref int currentCard, string name, int id)
    {
        this.name = name;
        playerId = id;
        while (handCard.Count < 5) //一開始先抽五張手牌
            DrawCard(cardDeck, ref currentCard);
    }

    public string Name
    {
        get { return name; }
    }

    public int PlayerId
    {
        get { return playerId; }
    }

    public int HandCardNum
    {
        get { return handCard.Count; }
    }

    public static int TopCard
    {
        set { topCard = value; }
    }

    private void ComputerTurn(int[] cardDeck, ref int currentCard) //id = 0 賽個人
    {
        Console.WriteLine("\n輪到電腦");
        ComputerThinking();
        /* //取消此註解可以看電腦的出牌情況
        Console.WriteLine("牌面: " + Int2poker[topCard]);
        PrintCards("電腦手牌", handCard);
        */
        if (CanIDealACard2() == -1)
        {
            bool turnOver;
            Random rnd = new Random();
            do
            {
                turnOver = Check(ref topCard, rnd.Next() % handCard.Count + 1);
            } while (!turnOver);
            if (currentCard < cardDeck.Length && handCard.Count < 5)
                DrawCard(cardDeck, ref currentCard);
            if (topCard % 13 == 10) //如果出了Jack 要決定花色與數字，花色決定為電腦最多的手色，數字隨亂數決定
            {
                if (specificCard == 10)
                {
                    int[] fourSuitsCount = new int[] { 0, 0, 0, 0 };
                    for (int i = 0; i < handCard.Count; i++)
                        if (handCard[i] % 13 == 10)
                            fourSuitsCount[handCard[i] / 13]++;
                    int[] temp = new int[fourSuitsCount.Length];
                    Array.Copy(fourSuitsCount, temp, fourSuitsCount.Length);
                    Array.Sort(temp);
                    specificCard += Array.IndexOf(temp, fourSuitsCount[0]) * 13; //從後面數來第一個
                    specificCard += rnd.Next() % 13;
                    topCard = specificCard;
                    Console.WriteLine("他選出了Jack!");
                }
            }
            else if ((CanIDealACard2() > 0) //有其他牌可出，不需出Jack(亂數派到Jack就跳過)
            {
                bool turnOver;
                Random rnd = new Random();
                do
                {
                    int whichHandCard = rnd.Next() % handCard.Count + 1;
                    if (handCard[whichHandCard - 1] % 13 == 10)
                    {
                        turnOver = false;
                        continue;
                    }
                    turnOver = Check(ref topCard, whichHandCard);
                } while (!turnOver);
                if (currentCard < cardDeck.Length && handCard.Count < 5)
                    DrawCard(cardDeck, ref currentCard);
            }
            else
            {
                int whichCard = 1;
                for (int i = 0; i < handCard.Count; i++)
                {
                    if ((handCard[i] % 13) < (handCard[whichCard - 1] % 13))
                        whichCard = i + 1;
                }
                coveredCard.Add(handCard[whichCard - 1]);
                handCard.Remove(handCard[whichCard - 1]);
                if (currentCard < cardDeck.Length && handCard.Count < 5)
                    DrawCard(cardDeck, ref currentCard);
            }
        }
        private void ComputerThinking() //牌只是讓電腦看起來有在思考
        {
            Console.WriteLine();
            Thread.Sleep(700);
            for (int i = 0; i < 3; i++)
            {
                Console.WriteLine(".");
                Thread.Sleep(500);
            }
            Console.SetCursorPosition(0, Console.CursorTop);
            Console.WriteLine(new string(' ', Console.WindowWidth));
            Console.SetCursorPosition(0, Console.CursorTop - 1);
        }

        public void MyTurn(int[] cardDeck, ref int currentCard)
        {
            if (playerId == 0)
            {
                ComputerTurn(cardDeck, ref currentCard);
                return;
            }
            Console.WriteLine("\n輪到我了！牌面为: " + name, Int2poker[topCard]);
            PrintCards("我的手牌", handCard);
            if (CanIDealACard() > -1)
            {
                Console.WriteLine("出牌還是選擇: ");
                switch (SelectList(new string[] { "蓋牌", "出牌" }))
                {
                    case 0: Console.WriteLine("請選擇要蓋牌:"); CoverACard(); break;
                    case 1: Console.WriteLine("請選擇要出牌:"); DealACard(); break;
                    default: Console.WriteLine("錯誤訊息：出牌蓋牌"); break;
                }
            }
            else
            {
                Console.WriteLine("請選擇要蓋牌: ");
                CoverACard();
            }
        }
    }

    private void DealACard() //出牌
    {
        if (topCard % 13 == 10) //如果出的是Jack
        {
            int specificCard = 0;
            Console.WriteLine("請決定花色:");
            specificCard += 13 * SelectList(new string[] { "Spades", "Hearts", "Diamonds", "Clubs" });
            bool successChosen = false;
            int number = 0;
            do
            {
                try
                {
                    Console.WriteLine("輸入預定花色(1~13):");
                    number = Convert.ToInt32(Console.ReadLine());
                    successChosen = true;
                }
                catch
                {
                    Console.WriteLine("輸入格式錯誤 請重新選擇!");
                }
            } while (!successChosen);
            specificCard += number;
            topCard = specificCard;
        }
        if (currentCard < cardDeck.Length && handCard.Count < 5) //如果牌堆還有且手牌<5張，則抽牌
        {
            DrawCard(cardDeck, ref currentCard);
            PrintCards("已翻牌", coveredCard);
        }
    }

    public void PrintCards(string title, List<int> handCovered)
    {
        Console.WriteLine("牌面: " + title);
        for (int i = 0; i < handCovered.Count; i++)
        {
            if (i < 8)
                Console.WriteLine("  ");
            Console.WriteLine("  " + Int2poker[handCovered[i]]);
        }
        Console.WriteLine();
    }

    private int CanIDealACard2() //不刪出牌:因牌-2，能出幾且不用考慮(沒有或只有1張):因牌-1，能出幾且要考慮
    {
        bool canDealACard = false;
        bool haveJack = false;
        int JackCount = 0;
        List<int> available = new List<int>();
        for (int i = 0; i < handCard.Count; i++)
        {
            if (((topCard % 13) == (handCard[i] % 13)) || ((topCard / 13) == (handCard[i] / 13))) ||
                (handCard[i] % 13 == 10)) //如果花色相同且相等或同是Jack
            {
                canDealACard = true;
                available.Add(handCard[i]);
                if (handCard[i] % 13 == 10)
                {
                    JackCount++;
                    haveJack = true;
                }
            }
        }
        if (!canDealACard)
            return -2;
        else
            return haveJack;
    }

    private void DrawCard(int[] cardDeck, ref int currentCard) //一次只抽一張卡
    {
        handCard.Add(cardDeck[currentCard]);
        currentCard++;
    }

    private string[] HandCard2String()
    {
        string[] handCardNames = new string[handCard.Count];
        for (int i = 0; i < handCardNames.Length; i++)
            handCardNames[i] = Int2poker[handCard[i]];
        return handCardNames;
    }

    private bool Check(ref int topCard, int dealtCard)
    {
        if (((topCard % 13) == (handCard[dealtCard - 1] % 13)) || ((topCard / 13) == (handCard[dealtCard - 1] / 13)) || (handCard[dealtCard - 1] % 13 == 10)) //當底子相等且花色相同或Jack
        {
            topCard = handCard[dealtCard - 1];
            handCard.Remove(handCard[dealtCard - 1]);
            return true;
        }
        else
        {
            if (playerId != 0)
                Console.WriteLine("此時不能出!");
            return false;
        }
    }

    private void DealACard() //出牌
    {
        bool turnOver;
        do
        {
            turnOver = Check(ref topCard, SelectList(HandCard2String()) + 1);
        } while (!turnOver);
    }

    private void CoverACard() //蓋牌
    {
        int whichCard = SelectList(HandCard2String()) + 1;
        coveredCard.Add(handCard[whichCard - 1]);
        handCard.Remove(handCard[whichCard - 1]);
    }

    public static int SelectList(string[] nameList) //篩選式
    {
        bool successSelect = false;
        List<string> nameList = new List<string>();
        RefreshList(nameList, nameSelected);
        while (!successSelect)
        {
            switch (Console.ReadKey().Key)
            {
                case ConsoleKey.UpArrow:
                    nameSelected = (nameSelected - 1 < 0) ? nameList.Length - 1 : nameSelected - 1;
                    ClearNameList();
                    RefreshList(nameList, nameSelected);
                    break;
                case ConsoleKey.DownArrow:
                    nameSelected = (nameSelected + 1 >= nameList.Length) ? 0 : nameSelected + 1;
                    ClearNameList();
                    RefreshList(nameList, nameSelected);
                    break;
                case ConsoleKey.Enter:
                    successSelect = true;
                    break;
                default:
                    break;
            }
        }
        return nameSelected;
    }

    private static void RefreshList(string[] nameList, int nameSelected)
    {
        for (int i = 0; i < nameList.Length; i++)
        {
            if (i == nameSelected)
            {
                Console.BackgroundColor = ConsoleColor.White;
                Console.ForegroundColor = ConsoleColor.Black;
                Console.Write(nameList[i]);
                Console.BackgroundColor = ConsoleColor.Black;
                Console.ForegroundColor = ConsoleColor.White;
                Console.WriteLine();
            }
            else
            {
                Console.WriteLine(nameList[i]);
            }
        }
    }

    public static void Clear(int num)
    {
        for (int i = 0; i < num; i++)
        {
            Console.SetCursorPosition(0, Console.CursorTop - 1);
            Console.WriteLine(new string(' ', Console.WindowWidth));
            Console.SetCursorPosition(0, Console.CursorTop);
        }
    }
}

```