HW6

班級	姓名	學號	日期
四機械四乙	吳宇昕	B10831020	12/20/2022

Q3A

sorce code and replit

終端機輸出

```
Student ID: B10831020
The two points fartest apart are (-9.5, -2.1) and (10.3, -2.1)
With distance 19.800
```

計算最長距離

題目給7個點的x,y座標，求最遠兩點的距離。求解過程如下：

1. 找出擁有最大與最小x,y座標值的四個點，最遠距離一定是此四點其中兩點距離
2. 設一變數紀錄最長距離
3. 計算此四點兩兩之間的距離，若當下的兩點距離大於紀錄的最長距離，就取而代之

心得

計算歐式距離需要開根號，耗費較多計算資源，應盡可能降低開根號次數。若要計算每一個點與其他6個點之間的距離，須至少開C(7,2)次根號。但是可以確定最大距離一定發生在四個邊界點之間，只需要計算四個邊界點兩兩之間的距離，開C(4,2)次根號就夠了。若題目加入更多點的座標，不會增加開根號次數。

尋找四個邊界點所需的時間會隨題目的點數增加而線性上升，比起指數型上升是相當大的改善。

或許這個題目還有更好的解法，進一步減少計算成本，目前這是我想到最好的做法。

Q3B

sorce code and replit

終端機輸出

Student	Grades			Stu Avg
0	90	90	80	86.667
1	80	80	70	76.667
2	50	60	70	60.000
3	40	80	80	66.667
4	60	60	70	63.333
5	70	80	70	73.333
6	90	60	50	66.667
7	30	80	60	56.667
8	60	60	50	56.667
Avg	63.333	72.222	66.667	

心得

C#有個很好用的關鍵字`readonly`，讓一個class attribute的值經初始化後便改為唯讀，不可變更。這比C++的`const`關鍵字好用，因為一個const member沒辦法初始化賦值。 C#好像不讓我們把的class member設為const，若要一個class member值固定不變，必須用readonly。因此這題我把學生的成績設為`readonly int[,]`，放在`class Program`裡面。

```
59          private static readonly int[,] sGrades =
60          {
61              {90, 90, 80},
62              {80, 80, 70},
63              {50, 60, 70},
64              {40, 80, 80},
65              {60, 60, 70},
66              {70, 80, 70},
67              {90, 60, 50},
68              {30, 80, 60},
69              {60, 60, 50}
70          };
```

Q5

sorce code main.cs Deck.cs Card.cs Player.cs and replit

三份cs檔分別包含`class Program`、`class Deck`、`class card`及`class Player`，皆屬於`namespace Q5`

終端機輸出

```
Studnet ID: B10831020

The entire deck after shuffling
 2--Club      Q--Spade     6--Club      8--Heart
 Q--Club      4--Spade     4--Heart     A--Club
10--Spade     3--Club      4--Diamond   J--Spade
10--Diamond   5--Diamond   J--Diamond   6--Heart
 8--Spade     2--Heart     J--Club      A--Heart
 K--Club      6--Diamond  10--Club      9--Diamond
 7--Spade     7--Club      8--Club      9--Club
 3--Spade     6--Spade     8--Diamond   3--Heart
 9--Spade     5--Spade     A--Spade     2--Diamond
 5--Heart     K--Diamond  10--Heart     5--Club
 A--Diamond   4--Club      9--Heart     2--Spade
 7--Diamond   J--Heart     Q--Diamond   Q--Heart
 K--Spade     3--Diamond   7--Heart     K--Heart
Deal #1
Player 0|:  2--Club      Q--Spade
Player 1|:  6--Club      8--Heart
Player 2|:  Q--Club      4--Spade

Deal #2
Player 0|:  4--Heart     A--Club
Player 1|: 10--Spade     3--Club
Player 2|:  4--Diamond   J--Spade

Deal #3
Player 0|: 10--Diamond   5--Diamond
Player 1|:  J--Diamond   6--Heart
Player 2|:  8--Spade     2--Heart

Deal #4
Player 0|:  J--Club      A--Heart
Player 1|:  K--Club      6--Diamond
Player 2|: 10--Club      9--Diamond
```

自定義Card class

```
class Card
{
    private readonly static string[] sSuit = {"Spade", "Club", "Diamond",
"Heart"};
    private readonly static string[] sNumber = {"A", "2", "3", "4", "5", "6", "7",
"8", "9", "10", "J", "Q", "K"};
    private int suitIdx;
    private int numberIdx;

    public string Suit => sSuit[this.suitIdx]; // custom get accessor for suit of
a card
    public string Number => sNumber[this.numberIdx]; // custom get accessor for
Number of a card

    /// <summary>
    /// Create an instance of a card.
    /// </summary>
    /// <param name="_suitIdx">The index to retrieve the suit of this card as a
```

```
string from array Card.sSuit</param>
    /// <param name="_numberIdx">The index to retreve the number of this card as a
string Card from array Card.sNumber.</param>
    public Card(int _suitIdx, int _numberIdx)
    {
        this.suitIdx = _suitIdx;
        this.numberIdx = _numberIdx;
    }

    public override string ToString()
    {
        return string.Format("{0,2}--{1,-9}", this.Number, this.Suit);
    }
}
```

每張牌都有一個花色與一個數值，兩者都應該是string。然而，過去似乎聽說string是指向heap的char pointer, 在程式裡生成過多string容易使記憶體零散。因此，每張牌的花色與數值欄位我並沒有用string的方式儲存，而是以int儲存，作為索引另外兩個static string array `sSuit`與`sNumber`的索引值。如此一來，每個card instance只佔據記憶體連續的16個byte。也就是說，每個instance的`this.Suit`跟`this.Number`並不佔據記憶體空間，它們只是個method，被呼叫的時候去索引`Card.sSuit`跟`Card.sNumber`陣列，回傳一個字串。

有了這兩個accessor，即使每個card instance並沒有真正的`this.Number`跟`this.Suit`兩個attribute，也可以對一個card instance打點簡單取出它的數值跟花色。

```
Card c = new Card(2, 10);
Console.WriteLine($"{c.Number}--{c.Suit}"); // call the accessors of Number and
Suit
// Diamond--J
```

不知道這樣做是否真的可以提升程式效能，減少記憶體零散，或是只是我自找麻煩？

自定義Deck class

含有一個長度52的`Card`陣列`this.AllCards`，代表整副牌的所有卡片。

`Deal`方法

發牌的方法`this.Deal` pass by reference輸入一個玩家陣列，發兩張牌給每位玩家。每個Deck instance都會用一個int `this.lastGivenCardIdx`記錄自己`this.AllCards`陣列發到第幾張牌了，避免一張牌在不同次發牌間重複出現。發牌時，一律從洗好的牌組抽出最上面的一張牌發給玩家，從`this.AllCards`陣列第0張牌發到最後一張。

```
public void Deal(ref Player[] _players, int nCardsEachPerson = 2)
{
    Card[] cardsGivenToAPlayer = new Card[nCardsEachPerson];
    for (int i = 0; i < _players.GetLength(0); i++) {
        for (int j = 0; j < nCardsEachPerson; j++){
            cardsGivenToAPlayer[j] = this.AllCards[this.lastGivenCardIdx];
```

```
                this.lastGivenCardIdx++; // keeps track of which card in array
    this.AllCards has been given out
            }
            _players[i].ReceiveCards(cardsGivenToAPlayer);
        }
    }
```

這個發牌的方法在牌發完的時候會產生index out of range exception，玩家人數或每個人拿到的牌數量太多時會出問題。

自定義Player class

每個Player instance只有一個attribute，是`List<Card>`，代表該玩家的手牌。除此之外，Player class也定義了一些method，例如`ReceiveCard`、`ShowCard`等等，代表玩家可能做的事。還有一個static method `AllPlayersShowCards`，輸入一個玩家陣列，顯示所有玩家的手牌。

心得

C#確實比C++好寫很多。有了accessor的設計跟簡易的getter, setter，讀寫class內容的程式碼變得很簡單。

唯一比較想抱怨的，是C#不太讓我們把物件存在stack上，而且所有物件都需要一個個初始化。像是我的`Player`陣列：

```
Player players = new Player[3];
```

這樣寫只有初始化陣列本身，而沒有初始化到陣列裡的player instance。要走訪這個陣列，初始化一個個player instance，甚至不能用foreach loop。這樣寫行不通

```
foreach(Player p in players){
    p = new Player();
    // p is a foreach loop variable, cannot be reassigned
    // or initialized
}
```

必須用傳統的for loop，寫成這樣:

```
for(int i = 0; i < players.Count(); i++){
    players[i] = new Player();
}
```

創建instance的程式碼比C++ stack-allocate物件複雜，但這恐怕是在C#或Java都無法避免的。

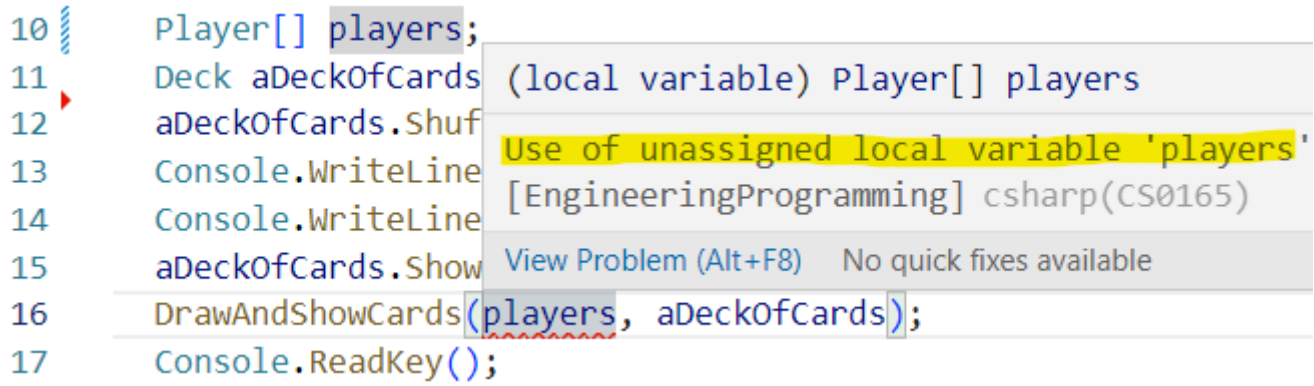
Q6

使用Q5的程式碼測試vscode intellisense跟debugging功能。使用dotnet sdk 7.0，建置vscode開發環境。

Compile time error

C#每個物件都需要用`new`關鍵字初始化。下圖是我創建了一個Player陣列，稱為`player`，卻沒有使用`new`初始化陣列本身。當我試圖把這個陣列拿來用，傳進別的method時，vscode intellicense在compile time就劃紅線顯示錯誤訊息，告訴我這個陣列尚未初始化。

雖然不太清楚為甚麼錯誤訊息是說*Use of unassigned local variable*而不是*uninitialized local variable*。

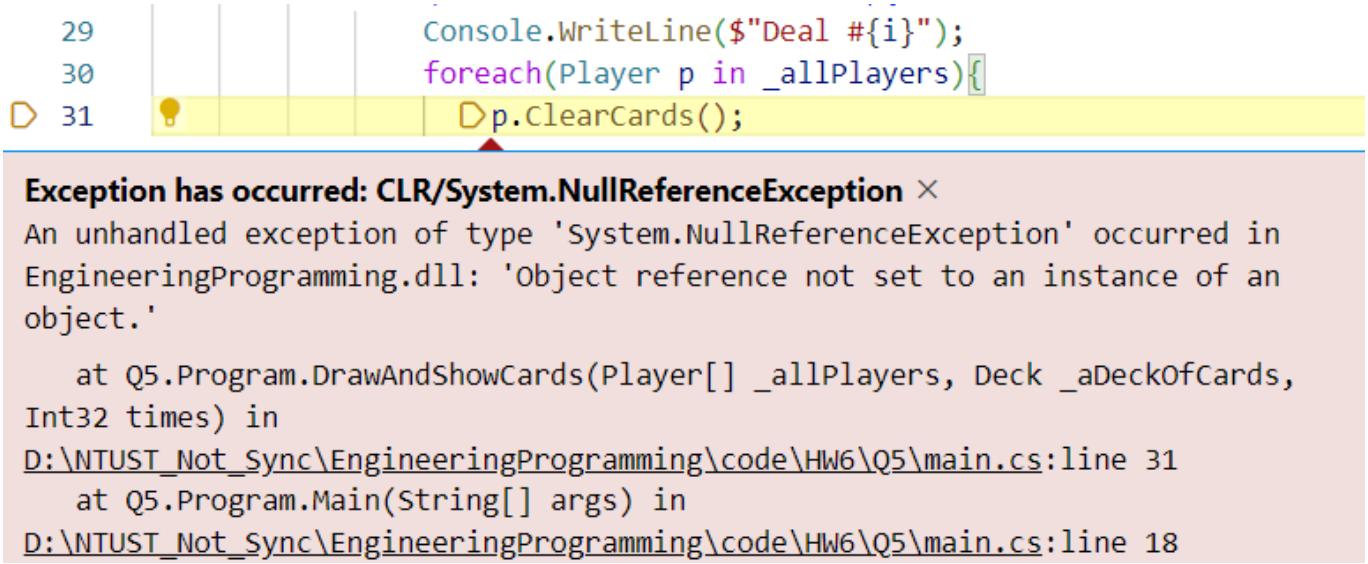


第10行加上`new`關鍵字後，紅線就消失，可以編譯了。

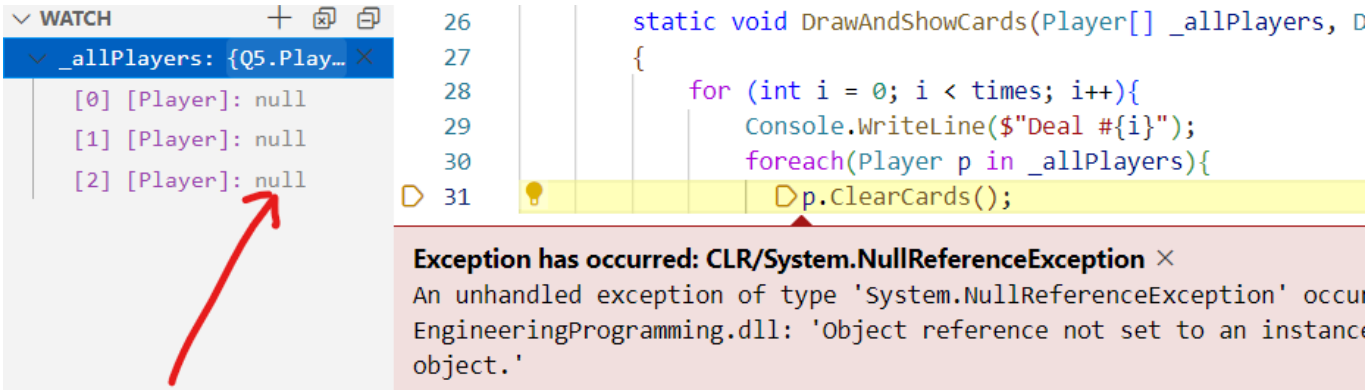
```
Player[] players = new Player[3];
```

Run time error

剛才的player陣列本身加上`new`關鍵字以後成功初始化了，但是裡面的元素，一個個Player instance沒有初始化，造成*NullReference Exception*



查看vscode debug工具列裡面的local variable watch視窗，可以看到陣列本身存在，但是裡面的三個元素還是`null`



在別處用for loop走訪這個陣列，初始化每個元素後就解決了這個run time error。

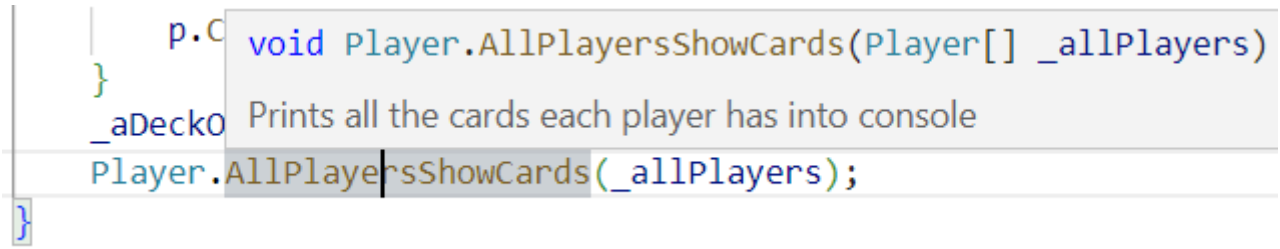
```
for(int i = 0; i < 3; i++){
    players[i] = new Player();
}
```

Xml-style comments

書裡有提到C# xml-style comment的功能，試著幫Q5的程式碼加上一些註解。

```
52      /// <summary>
53      /// Prints all the cards each player has into console
54      /// </summary>
55      /// <param name="_allPlayers">Array of players in the game</param>
        1 reference
56      public static void AllPlayersShowCards(Player[] _allPlayers)
57      {
58          int nPlayers = _allPlayers.GetLength(0);
59          for (int i = 0; i < nPlayers; i++){
60              Console.Write("Player {0:d}|: {1:S}", i, _allPlayers[i].ShowCards());
61          }
62          Console.WriteLine("");
63      }
64  }
```

同一個C# project使用到這個method的地方，只要把游標移到函式名稱上方，就會依summery, output, parameter自動顯示xml comment的內容。



但是有點疑惑的是，它只有顯示出<summery></summery>的內容，其他像<para name></para name>裡的，都沒有顯示出來。不知道我是哪裡做錯了，還是有什麼vscode套件的問題。

Break point

過去只知道break point可以讓程式執行到那裏就停下來，不知道還有conditional breakpoint這種東西。過去曾經遇到一個問題，走訪陣列的迴圈走到1000次的第894次時，總是發生runtime error。有conditional breakpoint，就可以在第893次的時候停下來，開始用step into功能單步執行，這樣更方便。

心得

vscode的intellisense非常人性化，可以自己用xml語法控制註解內容真是一大福音。加上精心設計`task.json`跟`launch.json`的內容，f5一按下去就自動編譯並開始偵錯程式，一切流程自動化太方便了。

更棒的是，vscode免費。