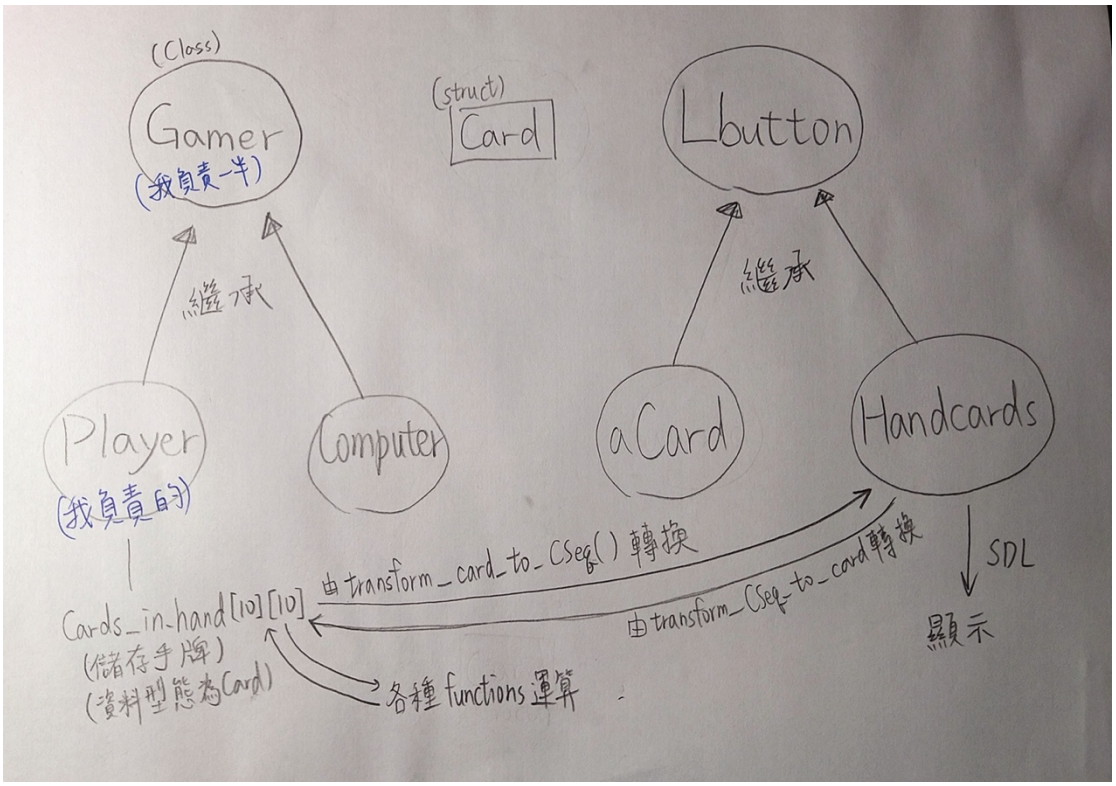
CPL 全班打麻將

【第一部分】

1. 主題發想：當初在設計題目時，大家先各自挑選有興趣的主題，或喜歡的遊戲，並發現大家都希望做益智遊戲類型，而紙牌類遊戲規則完善，不需多做複雜的設計（因為原本就夠複雜了），因此最後選擇**麻將**作為主題。

2. 程式架構：



在 $\mathcal{B}$ 部分有更詳細的說明 ~

3. 工作分配：

呂俐君：SDL 介面設計及程式實作（主選單畫面、出牌至牌桌等）以及美工（手繪遊戲主畫面、碰胡之底圖等）

夏良語：遊戲主要程式整合（main）、Gamer（遊戲主要類別）、Player（繼承自 Gamer 之針對玩家所設計之類別）

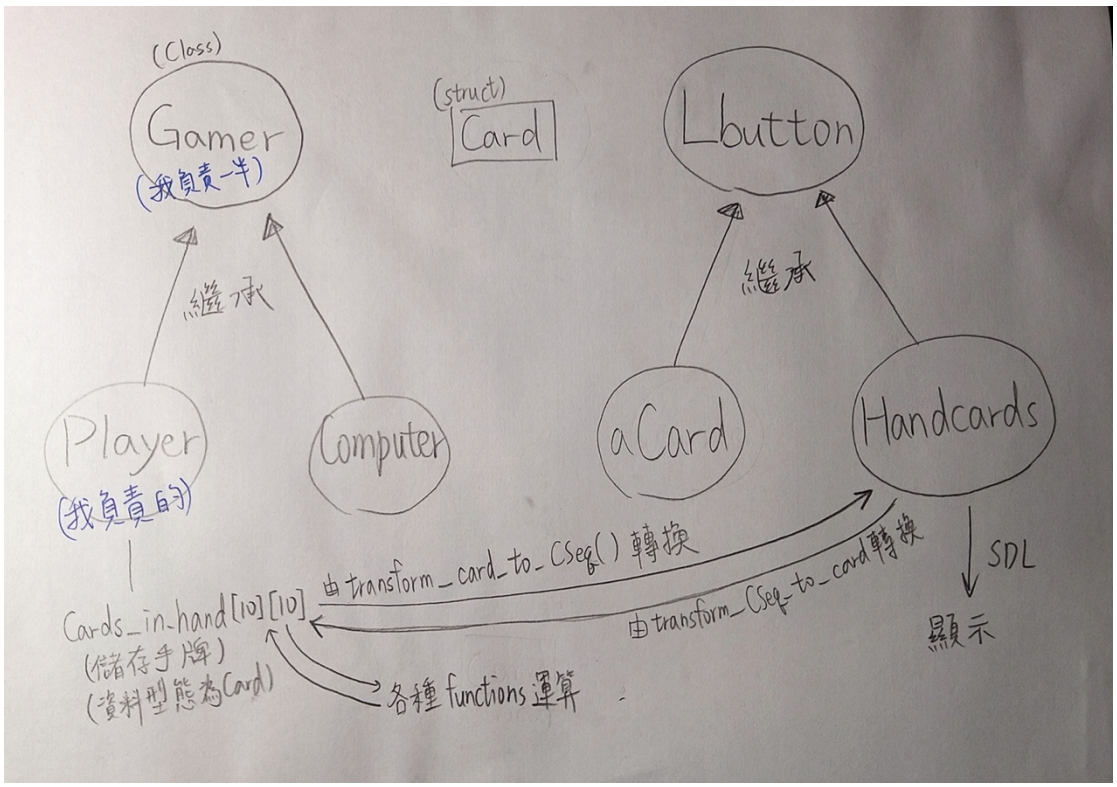
林靖容：遊戲主要程式整合（main）、Gamer（遊戲主要類別）、Computer（繼承自 Gamer 之針對電腦所設計之類別）

4. 心得：在專題中真的學到很多，程式撰寫的熟練度提升不少，尤其是在 class 和 inheritance，蠻深刻的體會到何為「物件導向」。在程式撰寫過程中，有時會有「要是一開始哪個陣列如何如何設計，就可以省下很多麻煩」等等的想法，如果之後有機會再做類似的工作，希望可以再開始前仔細規劃設計，避免重蹈覆徹（雖然沒有太大的影響啦）。另外，下次應該將檔案分成多個 header 檔，才不會幾千行的程式，很不方便閱讀及 debug。這真的是一個很特別的經驗，連續好幾天每晚熬夜和組員視訊討論，學會如何在意見相左又累得要死的時候，理性的思考與溝通，也很感謝組員們在這段時間的各種凱瑞，大家都辛苦了。

5. 註：第二部分所標示之行數皆出自 main.cpp 檔。

【第二部分】

A.



圖：圈圈代表 class，方形代表 struct。我負責的部分為 Gamer 部分 functions 及 player。Gamer 是玩家(含電腦)的 base class，被 player 和 Computer 兩個類別繼承。圖中下方的箭號為在各個 class 間傳遞資料的轉換方式，簡單來說，玩家的資料會儲存在 player 裡，因為手牌顯示介面等（Lbutton 那邊）為另一個人負責，為了將兩邊連結在一起，而設計了 transform_card_to CSeq 這個函數，將玩家手牌經過轉換，變成 Handcards 並顯示於牌桌；而針對手牌做運算及變動則還是在 player 這邊進行。以下為各個 class 的詳細內容，其中 Gamer 我負責的部分將以黃色螢光筆標示。

Gamer：因為玩遊戲的四個人（含三個電腦）有許多相似的功能，如發牌、判斷是否胡牌等等，因此設計此類別作為 base class。類別中包含：

protected—NUMBER_OF_CARDS（手牌總數，用來在過程中確認手牌數量正確）、canchi/canpeng（bool 型態變數，用來記錄特定狀況時能不能吃或碰）、haschi/haspeng（bool 型態變數，用來記錄玩家是否有選擇吃或碰）

public—constructor、cards_inhand（記錄手牌資訊的二維陣列）、deal（發牌函數）、checkchi/checkpeng/checkhu（檢查是否吃/碰/胡的函數）、printchi/printpeng/printhu（判斷可以吃/碰/胡後，顯示按鈕給玩家選擇的函數）、well_formed（和 checkhu 結合，用遞迴方式判斷是否胡牌，在 D. 小題中有更詳細的說明）、print_cards_inhand（cout 出玩家手牌，用來 debug）

會如此封裝，是因為都需要繼承給子類別，但有些不想被外面使用，因此設為 protected。

player：玩家的類別，繼承 Gamer，另外增加了：

private—max_card_num（為 const int，值等於 4，因爲相同的牌最多四張，又不想要被外面直接使用，故設為 private）

public—constructor、destructor、play（玩家回合出牌的函數）、operator++

（prefix 的 operator overloading，代表抽一張牌）、draw_card（和 operator++結合，抽牌並顯示在畫面上）、get_CARD_NUMBER（用來取得 Gamer 類別中 protected 的整數 NUMBER_OF_CARDS 的函數）、get_max_card_num（用來間接取得 private 中的 max_card_num）

以上函數不放在 Gamer 中是因為 computer 和 player 有需不需要顯示牌的差異，故將需要顯示牌的函數獨立出來放在 player 這個類別中。

B.

1. Gamer 建構子（第 698～710 行），建立一個二維動態陣列，存放手牌
2. player 建構子（第 1070～1079 行），並在其中使用到 member initializer（第 1070 行）
3. player 解構子（第 1080～1085 行），delete 二維動態陣列
4. 組裝關係： Gamer 中包含 card 型態的陣列（card 為 struct）（617、703~706 行）
5. 繼承關係：player 和 computer 繼承 Gamer，aCard 和 Handcards 繼承 Lbutton
6. get_CARD_NUMBER 函數（1065, 1087~1089 行），使得可以從外部透過 player 物件取得 Gamer 類別中 protected 的 NUMBER_OF_CARDS 的值。

C. Operator overloading:

i. Operator++ (prefix)

實作位置：宣告—第 1061 行、定義—第 1166～1171 行、使用函數 draw_card—第 1119~1164 行

呼叫位置：第 1555, 1580 行

使用方式：呼叫方式為++player_name，會在該玩家的手牌中，在不超出總共牌數的前提下（e.g.相同的牌每種只有 4 張），隨機增加一張，並回傳一個 player。

詳細運作方式：呼叫後（++player_name），該 operator 會呼叫 draw_card 函數，該函數會隨機選一張牌（如下圖），放入呼叫者的手牌。

```
1119    void player::draw_card(){
1120        card tem;
1121        std::random_device rd;
1122        std::mt19937 gen(rd());
1123        std::uniform_int_distribution<> dis(0, 33);
1124        int ran=0;
1125        for(int i=0 ; i<1 ; i++){
1126            ran = dis(gen);
```

使用原因：因為抽牌就是再多拿一張牌，用++這個運算子很直觀，又可以簡化程式。如果不使用該運算子，可以直接呼叫 draw_card 函數，但較不直觀也較冗長。

ii. Operator!

實作位置：宣告—第 1062 行、定義—第 1173～1183 行

呼叫位置：第 1540 行

使用方式：呼叫方式為 !player_name，會將該玩家的手牌數量歸零，並回傳一個 player。

使用原因：雖然遊戲本身沒有使用到這個 operator，但在 debug 的過程中常常需要將手牌歸零，每次都要用一個雙層回圈覺得很麻煩，就 overload 這個 operator 來簡化程式。而且「！」有「否」的含義，很直觀。如果不使用該運算子，會較不直觀且很冗長。例如第 1539~1552 行，功能為給玩家一副牌可由程式設計者自行設計的手牌，在 debug 時發揮很大的作用。方式是先將手牌歸零，再放入想要的牌。在歸零的過程中，如果不使用該 operator（1540 行），就要用一個雙層回圈，導致程式碼冗長且不易閱讀。

D. Advanced:

i. Static member（第 689, 696 行）：Gamer 類別中的 static int NUMBER_OF_CARDS 存放總牌數（用來確定發牌後一開始為 16 張）。

ii. 遞迴（第 835~886 行）：透過 well_formed 函式（835~886 行）的遞迴，判斷是否胡牌。邏輯：首先，看總共的牌數是否為兩張（837~858），若是，則判斷兩張牌是否相同，若亦是，則回傳

true，即胡牌。若總牌數不為兩張，則接著尋找是否有三張相同的牌（860~869），若有，則將這三張牌從手牌中移除，並呼叫自己（well_formed），進行遞迴。若無，則再尋找是否有三個連續數字的組合（870~883），若有則將這三張牌從手牌中移除，並呼叫自己（well_formed），進行遞迴。若無，則回傳 false，跳回上一層遞迴（若無上一層則代表沒有胡牌），繼續尋找下一種可能。

程式碼：

```
835 bool gamer::well_formed(card** cards_inhand, card** cards_copy){
836     cout << "here 2\n";
837     int total_cards = 0;
838     for(int i=0 ; i<10 ; i++){
839         for(int j=0 ; j<10 ; j++){
840             total_cards += cards_inhand[i][j].num;
841             //cout << cards_inhand[i][j].num << '\n';
842         }
843     }
844     cout << total_cards << '\n';
845 
846     if(total_cards==2){ //eyes
847         cout << "here 3\n";
848         print_cards_inhand();
849         for(int i=0 ; i<10 ; i++){
850             for(int j=0 ; j<10 ; j++){
851                 cout << "hi ::: " << cards_inhand[i][j].num << '\n';
852                 if(cards_inhand[i][j].num==2){
853                     cout << "here 4\n";
854                     return true;
855                 }
856             }
857         }
858     }
859 
```

```
860     for(int i=0 ; i<10 ;i++){
861         for(int j=0 ; j<10 ; j++){
862             if( cards_inhand[i][j].num >= 3 ){
863                 cards_inhand[i][j].num -= 3;
864                 cards_copy[i][j].num += 3;
865                 total_cards -= 3;
866                 if(well_formed(cards_inhand,cards_copy) == true) return true;
867             }
868         }
869     }
870     for(int i=0 ; i<3 ; i++){
871         for(int j=0 ; j<8 ; j++){
872             if( cards_inhand[i][j].num>=1 && cards_inhand[i][j+1].num>=1 && cards_inhand[i][j+2].num>=1 ){
873                 cards_inhand[i][j].num--;
874                 cards_inhand[i][j+1].num--;
875                 cards_inhand[i][j+2].num--;
876                 cards_copy[i][j].num++;
877                 cards_copy[i][j+1].num++;
878                 cards_copy[i][j+2].num++;
879                 total_cards -= 3;
880                 if( well_formed(cards_inhand,cards_copy) ) return true;
881             }
882         }
883     }
884     cout << "Hu Return False QQ~\n";
885     return false;
886 }
```

iii. static_cast<>（第 1073, 1176, 1209, 1375, 1452 行）：因為我們使用 enum 來存放牌的花色（第 16 行），在將整數轉為對應花色時，便使用到 static_cast<>來轉換，如下圖例子：

```
16  enum suit{Tong, Line, Wan, East, South, West, North, Cheng, Fa, Bai, NONE};
```

```
1173  player player::operator!(){
1174      for(int i=0 ; i<10 ; i++){
1175          for(int j=0 ; j<10 ; j++){
1176              cards_inhand[i][j].type = static_cast<suit>(i);
1177              cards_inhand[i][j].value = j;
1178              cards_inhand[i][j].num = 0;
1179              cards_inhand[i][j].picture = pic[i][j];
1180          }
1181      }
```

iv. random（第 722~756, 1119~1164 行）：在發牌及抽牌時，一開始使用 time(0)當作 random 的 seed，但發現因為取值的時間十分接近，導致出現一連串相同牌的狀況，上網查詢後找到下面這個方式，成功解決此問題，其中 ran 為程式所需要的隨機數值：

```
1119  void player::draw_card(){
1120      card tem;
1121      std::random_device rd;
1122      std::mt19937 gen(rd());
1123      std::uniform_int_distribution<> dis(0, 33);
1124      int ran=0;
1125      for(int i=0 ; i<1 ; i++){
1126          ran = dis(gen);
```

v. function overloading（第 758~799, 801~833 行）：在判斷胡牌時，分成兩種情況，一種是別人出牌時，要檢查他有沒有放槍，即那張牌若加入自己的手牌能不能胡。另一種是自己抽牌，檢查有沒有自摸，即此時的手牌能不能胡。兩種情況的差異為，前者的牌不屬於該玩家的手牌，因此需將它一併傳入檢查是否胡牌的函數

（checkhu）中；而後者的牌已放入該玩家的手牌中，故不需傳入該張牌。因此便設計了一個 function overloading，方便呼叫。