# Big Two Project Report

**One. Code:**

- card.cpp
  - Initialize the card suit, number, whether it is selected or not, and whether it is played
  - Initialize card length, width, number and suit strings, and create an image path (for printing images).
  - Get the suit
  - Corresponding numbers to A, J, Q, K, if they do not belong to 1~13, it is invalid number
  - Establish less than operator overloading
    - If the callback is less than true; greater than the callback false; Equal is the suit
  - Establish an equal sign operator overloading
  - Print out the picture
  - If you press the mouse to return true within the specified range and change whether it is selected, the position of the card will be changed according to its status; If not, false is returned.
  -
- cardcombonation.cpp
  - Initialize cards and card types
  - The operator overloading of the design judgment grade, the card type is the same as the size; Straight flush max (must return true); In the case of iron, the hand compared is not the highest when the straight flush is made.
  - Create a hand string
  - Print the cards

- single.cpp / pair.cpp / fullhouse.cpp / fourofkind.cpp / straight.cpp
  - Pass the cards to m_comparator for comparison

- player.cpp
  - Initialize the pass state, the number of cards left in the hand, and the score
  - Print the cards
  - Sort the cards
  - If the card has been selected and has not yet been played, it is passed to the vector, and the card is sent to the checker to determine whether the card type has passed
  - Move cards
  - If a card has already been played, remove it

- mainwindow.cpp
  - Initialize each parameter and design the initial page (background image, number of players, start button, etc.), and link the button and function.
  - Build a destructor
  - When it is the current player's turn, they can discard their cards

- Determine whether it can be passed: when the previous round is NULL (the rest of the players pass), the original player cannot pass; If the pass is successful, change the judgment parameter to true.
- At the end of the round, the table is cleared, the pass parameter is set back to false, and the score is calculated, if the cumulative score exceeds 50 points, the game ends, otherwise the next round continues.
- Design a checker (grouped by the number of cards):
  - Pick a card for single
  - Choose two cards of the same kind as a pair
  - When choosing five: four cards of the same number plus one random card as Iron Branch (the card of the higher size is the number of four cards); Three cards of the same number plus two other numbers are gourds (the cards of the same size are three numbers); Adjacent cards with a 1 interval are straights (cards of the same size are the highest numbers, and the same is the more suit), and five cards of the same suit are straight flushes.
  - If the card is greater than the previous card (according to the card type and number ratio), the card will be discarded instead of the previous card, the current player will become the last player, and the previous card will be removed.
  - If the player's remaining cards are not zero, the game continues, otherwise the score will be settled for the next round or the game will end.
- time
- Different objects are printed according to the state
  - Menu page to print the background
  - The play page prints the current player's hand and previous cards on the table (which have been released), as well as the player's remaining hand and scores; If it is in the pass state, the word pass will be printed ; Prints out the player's remaining time to discard.
  - The settlement page prints the remaining hand, the score, and the winning player of the round
- Use two random indicators to exchange to achieve the effect of shuffling, distribute the cards evenly to the players, and when there are three players, give the remaining cards to the player with a three.
- Toggle the current player, if the current player is the same as the last player, clear the previous card (replayable and unpassable), and display the number of cards and the score of the player's remaining cards.
- Determine which player has a three, and the player with a three, is the first player
- The round starts to clear the previous cards, then the cards are dealt according to the number of players, and the information of each player (player name, number of cards remaining, score) is added, and the game state is entered and the timer begins.
- When you return to the menu page, clear the card and player information, and stop the timer, at which point the state is
  menu status.

- If the time has not been played, the previous card is judged to be true then pass, otherwise the leftmost card will be automatically played
- Set the position of the button and the wording, and connect the button to the function
- When you click Exit, the screen closes
- Set the relationship between the screen and the buttons

- Main.cpp
  - The picture window is displayed

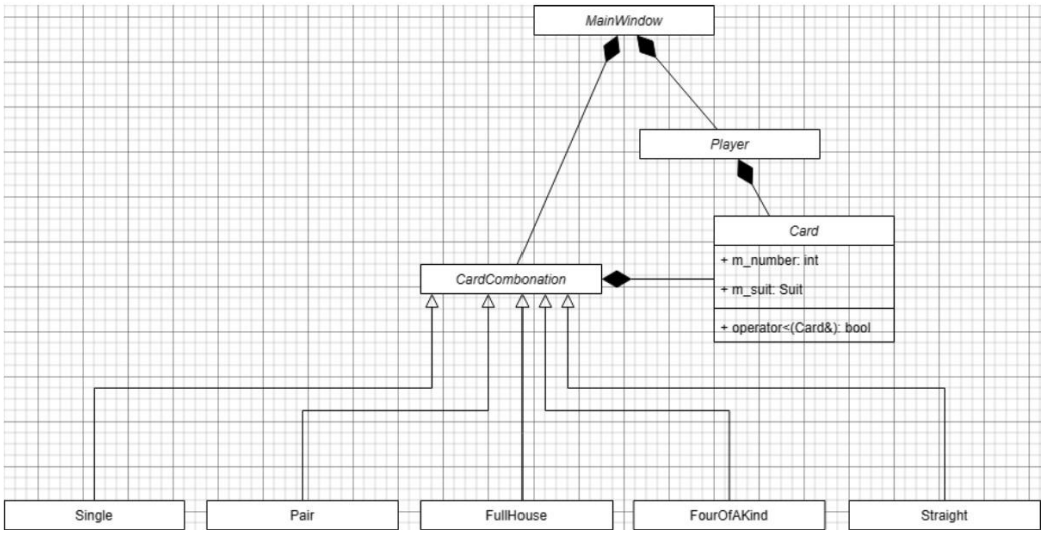
**Two Methods to design each function:**

1. Start page: Image display
    I. Save the image into the QPixmap variable
    II. Drawn via Mainwindow::p aintEvent
2. Start page: button
    I. Create a QPushButton *startGameBtn and use Object::connect to put QPushButton::clicked this signal with MainWindow::gameStart so that the gameStart will be executed when the button is pressed
3. Compare Card Rank :
    I. Overloading the Card::operator< the rule of size comparison is to judge the size of the number first and then the suit
4. Compare hand combination sizes:
    I. Overloading CardCombonation::operator< determines the card size according to the specified rules
5. Determine whether the card is legitimate:
    I. In MainWindow::checkCardCombination, first determine whether the number of cards is 1 or 2 or 5, if not, use QLabel *labelNotification to display invalidity
    II. If the number of cards is 1, it must be legally discarded
    III. If the number of cards is 2, then determine whether the two numbers are equal, equal is a legal card, and if not equal, it is not legal
    IV. If the number of cards is 5, it is determined whether it is FullHouse, FourOfAKind, Straight and Full Straight
6. Shuffling and dealing:
    I. In MainWindow::d istributeCards, shuffle the cards by taking two index1 and index2 at random, and then swapping the two indexes and repeating many times to achieve the shuffling effect
    II. The cards are dealt evenly, and when three players are dealt the cards to the player with a three
7. Pass Image:
    I. Use the Player::p assCurrentTurn variable to record/determine whether to pass the current turn
    II. Use QPainter::d rawPixmap to display the pass image
8. Calculate Score :
    I. According to the rules, go through everyone's hand in a round, add up the number of cards remaining, and multiply twice if there is 2

9. Player Rotation:
    I. After MainWindow::nextPlayer is executed, the player rotates
    II. Use int curPlayerIdx to record the current player's index in QVector<Player*>
    III. Use Player*curPlayer to point to the current player object

## Three Game Structure:

UML




Help:

MainWindow contains two cores: Cardcombonation and Player, of which Cardcombonation covers five major hand types, namely Single, Pair, FullHouse, FourOfAKind, and Straight; The basic settings of the card, etc., are linked to the player and the deck, and its content is provided to the player and the deck to use, and finally borrowed

MainWindow passes in main.cpp.

## Four Difficulties encountered and solutions:

A. When using std:: sort in the Player::sortCard method, you will encounter a segmentation fault
    I. Causes: At the beginning, the operator overloading of Card::operator< was not clearly considered, which may cause card1<card2 to be valid, but card2 < card1 is also true
    II. This will cause std::sort to be an undefined behavior
    segmentation fault, which was later found to have this problem and then corrected back

B. This project took me a lot of time to fine-tune the game screen and arrange the position of each object in the screen. Finally, in order to facilitate management, I wrote the common x, y coordinates of some objects to constants.h
, and errors are easier to correct.

C. At the beginning of the planning, we did not consider that if the number of players is not divisible by the number of cards (3 players), it will cause uneven splitting, until it was decided to implement "the number of players can be 2~4 players".

I thought of it when I thought of the bonus questions. Later, I wrote MainWindow::initFirstPlayer, to find the three of clubs in all players' hands, and put the extra card into the hand of the player with the three of clubs. When designing the "comparison of different discard combinations", I was troubled for a long time, and I originally thought that it was complicated to write a class for all possible discard combinations, and then compare each function, such as overloading the operation. Later, after thinking about it carefully, I found that in fact, each card combination has its own representative card, so I announced the Card m_comparator variable and CombType in CardCombonation to record the representative cards and hand types of each card combination, and in CardCombonation::operator< Discuss them separately according to the type of cards and decide whether to compare the representative cards and return true or false.

D. In the process of implementing the program, there have been two or three times when the program is halfway executed, it will automatically close, and the system does not display a special error message, so it took a lot of time to find the reason, because I added a new indicator member to the category, but I forgot to initialize this member into nullptr in the constructor, that is, use it. Causing program execution errors.

E. There are also many difficulties in the design of the game flow, for example, before I just started writing the program, I have already drawn a flowchart to make some preliminary planning, but after I start writing the program, I will find many edge cases that I didn't expect at the beginning, or the possibility that I didn't think would happen, resulting in the later stage, in order to meet these situations, I had to write something to make the code more legible if / else conditional judgments, such as MainWindow::checkCardCombination, which has about 150 lines, is a tragic case, but fortunately the problems with edge cases have been solved.