Introduction

Hydra is a card game that allowed two or more players to play. Each player has 54 cards at first: two jokers and 52 cards with value of 2-10 , J, Q, K, A, and suit of “S, C, H, D” At the beginning of the game, we mix all the cards and reallocate to each player. Players will play the game alternately. Each player has a draw pile, a discard pile and a reserve card.

In the first round, we will turn on player 1 ‘s card on the draw pile, to be a new head. Then for player 2, he has one card to play. If that card’s number is greater that that on the first card, then he will lose in this round and get that head and hand card into the discard pile. Also, 2 of his cards on his draw pile would be the next two heads. Then it is the round for the next player. If that card’s number is same as or less than that one, player 2 finish his round and put the hand card to the head, and let next player to play.

For each players’ round, they will play as much as the number of heads on the table moves in one round. For each move, if the number is same as the head newest card that player choose, this round would end. If the number is greater than the head newest card that player choose, this round would end, and he will take the oldest head and hand card into discard, and 2 of his cards on his draw pile would be the next two heads. For the number of heads on the table is greater than 1, they can put one of their hand cards into reserve. If there are already one card in reserve, then that card would be change with hand card. After each round, the reserve card would go back to the draw pile.

If the joker appears, it would be 2 if it would be a new header else player could determine the value of joker. If the card on the top of the head has value A, any hand card could be place on this head.

If any player that have 0 cards in draw pile, discard pile and reserve, then he will win the game. If the hand card is 0, and discard is not, firstly we check if there is a reserve. If there is a reserve, then it must be played before putting the discard into the draw pile. If there is not a reserve, we will change the sequence of the cards in the discard and put them into draw card. And continue playing. This game would end if one of the player win the game

Overview

I divide this program into 5 parts. (Cards, Heads, Players, Game, main)

For the first part (Cards), I have two classes: Card and Cards. Card mainly contains the value and type of a card. Value of a card is 0-13 with 0 for Jocker’s initial value, 1 for A, 11 for J, 12 for Q and 13 for K. Then, Cards contains a list of cards, with some method.

For the Second part (Heads), I have two classes: Head and Heads. Head is a Children of the Card with one more attribute of lose. If a head loses, it is not on the table. Then, Heads contains a list of Head, with some method.

For the third part (Players), I have two classes: Player and Players. Player mainly contains 3 piles: draw, discard, and reserve. Then, Players contains a list of Player, with some method.

For the fourth part (Game), it mainly contains a list of players and a list of heads. It will contain all the methods that will use in one move, and determine the actions.

For the last part (main). In main, we do the most job of input and output. And Game, Players and Heads do some of the outputs. It contains the whole structure, recursion for each round and each move, and determine who is the winner.

Design

For the list of Cards, Players, and Head, I use vector to produce the list. In the program, I use push_back() method to add element into the list. Then, I don't need to use delete, and reallocate the memory.

For outputting "player 1: 47 (12 draw, 35 discard) + 1 in hand, 2 remaining, 1 in reserve", As the remaining cards and reserve might goes back to the draw pile. I didn't divide the remaining card out of draw pile. Instead, I count the moves to show out how many cards are there waiting to play. If the round is over, there is no need to move the remaining back to the draw pile, as they are always in the draw pile.

For the Jocker, each time we meet jocker as hand card, we read in a value of it. And change the value of it. Since each time we use jocker, we change its value, we don't need to pay attention to the original value of the jocker. Instead, we only check if the type of the card is "J" to check if there is a jocker or not.

For heads, since each heads have a head number, and the oldest heads might move to the discard, it is hard to rewrite the sequence of the number. Then, I don't clear the head if this head moves to the discards of some players. Instead, I record whether this head is removed or not. If it is removed, I will not output it. Also, the input of move for the removed head is invalid.

For some special situations of when playing the game, I have several designs:
1, If we want to swap the reserve. We will record this movement so that we could check in the main function. Then, if it appears, we will finish this move and add one move to play the original reserve card.

2. When the draw card is not enough to play, we will record 2 values, the first one is the

number of draw cards, the second one is the number of moves that the players should play after playing out all the draw cards, which is the number of heads on the table minus the numbers of draw cards. Then, we will first change the times of recursion for moves to the number of draw cards, and finish the all the moves with them. If there is a reserve card when finishing moving the last draw cards, then we will add one moves to play this reserve as reserve should be played before moving the discard to the draw pile. After that, we move all the discard to the draw pile, and play the remain moves for this player.

3. To avoid memory leak, I use temp to copy the values of elements and push_back the temp to the lists. Then I delete the copied element in its list.

4. For which player to play, at each round beginning, I use "round number % number of players" to get the player in turn.

Resilience to Change

The basic cards are easy to change. Since I have a method to add the basic cards. If I want to add more cards that is super easy. Also, as the dividing parts for each player, I use the number of total cards divided by the number of players to get the number of cards each player will initial allocated. Thus, if we change the basic cards, nothing else would be change.

The rules of the game are easy to change. As all of my method is public, I can use all the methods and change every part of the game, for example:
1, If we let the last card in one round can not be reserve, then we could add "if the move is the last move in this round" to the part that determines whether the moving is valid.
2. If we want to change the numbers of new head players would add to the table if they lose, then we only need to change the times in the recursion.

I use many methods to avoids the whole program is too limited. For each method, it would not affect others. If we change the values in each methods, the program would run normally.

Answer to the Questions

Q1: What sort of class design … as possible?
A: I use the Builder Pattern. I write all attribute dividedly, thus changing of the rules only affect on small parts of program.
Q2: Jokers have a different … they are used?
A: Since every card have a num and a type, we could determine whether the type is "J" first, then we decide the value of Joker. Each time when we meet Joker, we change the value of it.
Q3: If you want to allow computer …structure your classes? Consider that different…to the computer payer?
A: I add a subclass of player, the computer could have everything the player has. Then we check the hand cards value and put on the oldest one it could be put on. If we cannot put

the card on all heads, we keep it as reserve. Then, if the next one is also cannot be put on heads, we lose in this round and get the oldest one as discard.
Q4: If a human player … to the computer player?
A: I will write a function play() that contains how this computer would act. If one of the player leave, we could just use play() to replace all the moves of this player.

Extra Credit Features

1. Gramma

For cards that have value A or 8, we use "an" to present the card.

2. House rules

The players could place their card to lose even if they can win if they putting this card onto another head. For example, we have heads 2: AC() and 3: 2H(). And our hand card is an 8S, we could place it on 2. However, we input 3, then we will lose in this move, and get the oldest heads with 8S to the discard and emit this round. This would be useful, since the player would place two cards to be the new heads. If the oldest head only have one card, with the hand card, the total number of cards would remain the same. Also, we could change our cards by doing so. For this example, we get an AC which is useful when playing in this game.

3. Reserve input and output

When inputting the move, I write "(input 0 if you want to reserve the card)". This is super convenience. If the move to reserve is invalid, it would output "YOUR MOVE IS INVALID< MOVE AGAIN. Your move is", and gives you a second chance to input the move. If there is nothing in the reserve before, after this move, it will output "You put a … into reserve." To tell you your move. If there is already a card in the reserve, then we swap this card with the hand card. After this move, it will output "You swap the reserve … with a/an …".

4. Testing mode

I finish all the requirement of testing mode. Moreover, I add a part to check if we want to use it for each move.

Final Questions

1. I worked alone, I notice that it important to make a UML graph before starting to write the program, all the stuff should be beginning with small attributes. At first, I write a lot of attributes that I could think of. Then, while writing the program, I found some attributes are useless, or some attributes are important. It is nearly impossible to get all the attributes right at first, but preparing all kind of attributes and method are good for the remaining programming. Also, planning for the time is also very important. I follow the plan I make, and finish the programming on time. However, I didn't plan for the design.pdf and the demo.pdf part. I will check the to-do list more careful next time. What is more, I think it

is important to run partial program, it is hard to debug when the program is large. If we debug while writing, the debug process would be easier. Furthermore, thinking the edge situations is hard. While writing the program, we will forget some edge situations. So, it is better to write down all the edge situations before writing. Then we could check if all the edge situations are considered in the program.

2. I would write all the explanations while writing the program instead of writing all the explanations after finish the whole program. Since it is a large program, we cannot finish the program in one day. When we review our own program in different day, we might forget what we write before. Next time, I will do better so that I can save my time from reviewing the job I've done before.

Conclusion

This program is interesting, and making a big program by myself is challenging. I enjoy the process of writing the code. I believe that I could do better next time.