Design

Project: Straight

Group Member: Yuqi Bai (20849507, y78bai)

Due Date: February 21, 2021

Overview

This program contains 4 main classes: Card, Board, Player, and Game.

Card is the 52 poker cards. Each card has a suit: space(S), club(C), heart(D), or diamond(D), and a rank from 1 to 13.

Board acts like the actual board in a game. It has the deck and 4 straights on the table. In addition, it is the board's responsibility to shuffle the cards.

Player represents the 4 players in a game. They share an observer Display (composition, explained in the next section). In my plan, player is an abstract class and has two derived classes Human and Computer. However, I realized it’s more convenient to have one concrete class Player and use a bool field to tell if this player is a human or computer.

Game has a board, a display, and 4 players (aggregation). It initializes a game and interacts with the main function.

There is a class name InvalidCommand. It is thrown when a invalid command is made.

Design

In order to generate the 52 cards, I used the factory design pattern. This is unnecessary because doing so is a little complicated and hardcoding also works, but it is a good practice of using the factory design pattern.

I also used the observer design pattern for printing messages. Player is the subject and the display is the observer. A player notifies the display when he/she/it plays or discards a card, and the display will print “Player<x> plays(discards) <card>.”

Resilience to Change

The classes have low coupling and high cohesion so have great resilience to change. Here are two possible changes that I can think of.

1. Short Deck. To make each round shorter, the game may use a short deck. For Instance, Js, Qs, Ks are removed and there are 40 cards in total. To make this change, I will add two new int fields to the class Board, called `first_rank` and `last_rank`. And there are 3 functions to be changed: `init()` and `showDeck()` in class Board and `deliverCards()` in class Game. More specifically, the loops.

2. Peek the Deck. The command “deck” is not part of the straights game since we don’t want the players to see what are other player's hands. However, we might want to enable some players to cheat. To make this change, I will add a new bool field called `can_cheat` to the class Player. When inviting players (is this player a human or a computer), if a special code is received from input, then this player’s `can_cheat` is true, otherwise it’s false. Then I will modify the main function. When the command is “deck”, check current player’s `can_cheat`. If it’s true, show him/her the deck. Otherwise, ignore this command.

3. More Levels. In my program, computer players have 3 levels, and there can be more. For example, I want a new level where computers play or discard hearts first. To do this, I will implement a helper function called `find_heart`. It takes a vector of cards and returns the first heart or a random one if there’s no heart. With `find_heart`, these computers know how to play or discard.

Answers to questions

Question 1: What sort of class design or design pattern should you use to structure your game classes so that changing the user interface or changing the game rules would have as little impact on the code as possible? Explain how your classes fit this framework.

The classes have low coupling and high cohesion, so changing the user interface or changing the game rules would only have a little impact on the code. Three detailed examples are given in the above section (Resilience to Change).

Question 2: If you want to allow computer players, in addition to human players, how might you structure your classes? Consider that different types of computer players might also have differing play strategies, and that strategies might change as the game progresses i.e. dynamically during the play of the game. How would that affect your structure?

In my program, class Player represents both human players and computer players. There is a bool field named `is_human` to indicate if a particular player is human or computer. There are 3 levels for computer players and each level has its own strategy to play the game. Details about this are in section “Extra Credit Feature”. To implement dynamic dispatch, it’s a good idea to use strategy design. I would try it if I had the chance to start over.

Question 3: If a human player wanted to stop playing, but the other players wished to continue, it would be reasonable to replace them with a computer player. How might you structure your classes to allow an easy transfer of the information associated with the human player to the computer player?

As I mentioned, class Player represents both human players and computer players. There is a bool field named `is_human` to indicate if a particular player is human or computer. Other than this, computer players and human players are the same. When a human player leaves, simply switch `is_human` to false and a computer player will take over. No need to transfer any information.

Extra Credit Feature

I implemented smart computer players. Computer players have three levels and they use different strategies. Level 1 computers always play or discard the card that has the highest rank. Level 2 computers make the first choice of legal moves, and the first choice of discards. Level 3 computers always play or discard the card that has the lowest rank. Discarding the card with the highest / lowest rank seems to be the worst / best strategy, but I'm not sure about the strategy of playing cards.

The difficulty is finding the card that has the highest or lowest rank. I wrote a helper function to compare the rank of two cards and use std::sort in <algorithm> to find the card that has the highest or lowest rank in a vector of cards.

Final Questions

Q1: What lessons did this project teach you about developing software in teams? If you worked alone, what lessons did you learn about writing large programs?

Read the requirements thoroughly before getting started. I missed some details mentioned in starights.pdf and I had to put unnecessary effort into fixing it.

Write comments and update the uml when writing the program. You might forget the purpose of a method or what you should do next if you don't do so.

Make good use of functions provided by the IDE. I use vscode and there are many useful functions. For example, if you want to change the name of a function, you can replace all occurrences of that name across all files using search. Another instance I learned is that if you want to change the indentation of a big part of your code, you select this part and press "tab" or "shift + tab".

Test the program carefully. I copied a function and didn't change every "heart" to "spade". The program could compile and never crashed, so I assumed everything was fine. I didn't spot this problem until the day before the deadline.

Q2: What would you have done differently if you had the chance to start over?

I would use some different strategy to implement. For example, in my plan I mentioned I wanted to use the strategy design pattern to implement function `play()` and `discard()` of computer players. For simplicity, I didn't actually use this design pattern. I would use the strategy design pattern if I have the chance to start over, it's a good practice.

In addition, I would add more extra features. In particular, I would implement a GUI so that the display is better. I didn't do the bonus question of A4 so I know little about GUI. I would like to give it a try if I have enough time.

I also would like to choose a group project. I want to experience working on a program with other people.