Final Design Document

Kyle Seto

Overview

In my project I have 4 c++ files, 3 header files and a make file. I used multiple files to make changes while not having to recompile all the code. This way I only had to recompile the files I made changes to, to test the program as the other files have already been compiled. Additionally I grouped the code in the files to maximize cohesion and minimalize coupling. All the implementation relating to cards was in a single file. The Card class, Hand class, and subclasses of hand like table and deck as well as their fields and methods were all in one file as these classes heavily relied on each other and will often work together. Classes like the player class and game class would rely on the card class, and hand class and would use the file as a module.

I implemented multiple classes including a game class, player classes, card classes, deck classes and more. The game class was used to hold all aspects of the game, it implemented the rules, different phases of the game, and turns. It contained other classes like player classes and deck classes that were necessary to have a game. The player class was used to hold data of a player including their hand, score, it also had a method to play its turn.

Within my classes I used c++ libraries like vector, algorithm, memory, random, and chrono. These libraries provided me with implementations of smart pointers, vectors, and random shuffling. Smart pointers were used to ensure no memory was leaked. Vectors were used to dynamically allocate memory while also not leaking memory.

My main file consisted of processing the command line argument that was used as the seed for the random and using a try and catch expression to play the game. This way the game could be quit from anywhere by throwing an exception. Therefore, I made all my functions with the exception safety level of basic guarantee.

The table class was a collection of cards that was sorted when a card was added so that it could be outputted in order. It also had vectors of integers for each suit that would be printed out instead of the cards.

Design (describe the specific techniques you used to solve the various design challenges in the project)

In the player of the game straights, it was possible that the player was a computer or a human. For each turn, I called a player method to play its turn. This method was virtual and was overloaded using computer and human classes that were subclasses of the player. This is an example of a template method design pattern. The player method was abstract with concrete implementation from the subclasses human and computer.

I also implemented the ability to quit the game from a player turn without leaking memory. This was done by using smart pointers and ensuring all functions have a basic guarantee, so that when the exception is thrown to quit the program it will not leak memory.

In the game class a vector of pointers to the player class was used so that we could have both players and computers. Since these were subclasses of players, we used polymorphism to allow the game to have both.

In the code, we also had many cases where there were many similarities among classes. For example the deck, the cards players had in their hand, the discards, and the table closely resembled each other but also had key differences. I used inheritance to reuse code for these classes. This way they all had similar implementations but I could also add their own methods and fields if need be.

Resilience to Change (describe how your design supports the possibility of various changes to the program specification)

My program is separated into multiple c++ files. This way when changes are made to the program, it is easier to recompile the files where the changes were made compared to recompiling all the code if it was in a single file. Furthermore, I used classes to encapsulate data. This way changes could be made to them without it affecting and causing the entire program to break. In some classes like the player class, the implementation and the abstraction were separated into different classes where the player class was the abstraction, and the human/computer class was the implementation. This way if we wanted to make changes to solely one of the abstraction or implementation, it could easily be done without having any effect on the other. I also leveraged polymorphism by using a vector pointer to the player class, this allowed the vector to be filled with those from a subclass. Therefore, adding additional subclasses to the player class such as a team class could be added without modifying the implementation of the game.

Answers to Questions

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.

Using a bridge design pattern would allow the developer to decouple the implementation and the abstraction of the game. By doing this, the developer can change the user interface (abstraction) without modifying any code from the implementation. This works the other way as well. The rules of the game can be changed (implementation) without having an impact on the code files for the user interface. In my case I used different class methods for the user interface and for the game rules in the same class. I decided not to use different classes as there were only a few methods relating to each. This design is less scalable and flexible to change, but I found it suitable for the size of the project.

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?

Creating subclasses of game players would allow for both computer and human players. Since computer players would need a strategy throughout the game we can implement a strategy design pattern. Implementing this design pattern would allow for the computer player to use different ways of playing and would also make it easier to add implementations of the computer player since the only addition would be the implementation and the attributes of the computer player would remain the same. In my structure, I did not implement a strategy pattern since I only had a single strategy for the computer. If I was to, I would need to add additional classes with the methods of their version of how they would play their turn.

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?

By using a player class that is a superclass of the human and computer classes, we can have a generalization structure. This allows us to use a template design pattern. By doing so, a player can easily take on attributes and methods and transfer information for the subclasses. This allows a smoother transition of a player going from computer to human and vice versa.

Extra Credit Features

No extra credit features were implemented.

Final Questions (the last two questions in this document).

1. What lessons did you learn about writing large programs?

I learned that large problems require a lot more planning than any smaller problem or assignment does. In large programs, there are lots of working parts that rely on each other for example, the player class relies on the implementation of a hand class and this relies on the implementation of a card class. For even larger projects, I would assume that the chain of dependencies is very long. Additionally to make changes to these dependencies and ensure that everything still works can be a difficult task. It would be smart to implement design patterns so that modifications are visible and easy to understand. This way it is easy to focus in on a single aspect of the project without being overwhelmed and worried that the changes would affect the rest of the project.

2. What would you have done differently if you had the chance to start over

Given the chance to start over, I would have used more design patterns. Utilizing factories, strategy, template design patterns would allow for a more scalable project. Although this project is not a large-scale project, I would be interested in learning more about how large-scale projects are designed and structured and working on my project as if it were one. This way I could add almost anything to the project without having to worry about how it would affect the other components. Some things I could add would be more players, larger decks, new cards, special rules, new scoring system, or multiple scoring systems.