**Project1: Whist – Design Analysis**

**SWEN30006 Software Modelling and Design**

**Author 1(Wei Ge), Author 2(Chang Shen), Author 3(Xubin Zou)**

**Introduction:**

This report is focusing on the techniques we used and processed we had in the original Whist code and try to improve the code to satisfy the GRASP by using different patterns.

**Assumption:**

We assume the maximum number of all players is four.

**Solution Design:**

**Summary:**

In this project, we used 2 design patterns to finish the task. They are Strategy Pattern, Decorator Pattern. Also we recognized that there are 4 main elements in the project: card, human, npc and rule.

We ignore the card part while poker is the standard cards, there are few opportunities for change in the future.

For the human parts, we separated the player's collection out from the main class, which will allow us to add new player freely. We ignored the playing method's extension of human while we think there are few opportunities for changing its manipulation.

We used the Strategy pattern in our Card Selecting part and Decorator pattern in our Card Filtering part. We think these two patterns are the most fitted in this task.

**Operation Process:**

First of all, everything starts in the ***Whist*** class. In this class, we will read the property file first, there are all the information about each NPC. Then we will generate the initial score of each player includes NPC. After that, the Whist will create the NPC and normal player by the information we read from the property file. Here is the point that all the initial setting has been set up.

When the game start, the ***playRound()*** method will run.

**Patterns and Principles:**

As discussed above, we used the Strategy pattern on our card selecting part. By using this design pattern, we can use different card selecting algorithm easily and can add any new selecting algorithm in any time without a bunch of code. This implement the low coupling. In our code, if the people want to add any new selecting algorithms into the game, they can just create a new selecting algorithm strategy and extends from ***CardSelectingStrategy*** class.

In our task, we have 3 select algorithms. They are HighestSelect, RandomSelect and SmartSelect. We let them become the child class of ***CardSelectingStrategy.*** By this pattern, the NPC do not need to know how each selecting algorithm works but they can just use it. It achieves the*Low Coupling*, high cohesion and Information expert.

Also, this satisfied The Open-Closed principle. If you want to make any changes, you know need to change the code we already write it, you can add the new one in very simple way.

Secondly, we used the *Decorator Pattern* in our card Filtering part. The reason that we chose Decorator Pattern is the Filtering part is optional. We create a ***FilterNPC*** interface to connect all the filtering algorithms with the game. Then we have ***FilteringDecorator*** class which is the parent class of all the Filtering algorithms classes. This class is the base decorator in the *Decorator Pattern*. The ***TrumpSavingDecorator*** and ***NaiveLegalDecorator*** are the Filtering algorithms classes. They are all extends from the ***FilteringDecorator.*** By using this pattern, we can install any Filtering algorithm on NPC. Keeping Open-Closed Principle is another reason that we use this pattern.

Then, we have ***ManipulatePlayer*** class which is used to control all the NPC and normal players***.*** This class make our project with low couplingand Pure Fabrication. This class is also satisfied the Open-Closed Principle because it is easy to do any extension on players.

**Alternative Solution:**

For the alternative solution

1. Composite pattern, 操作所有的NPC 和 player。 Print不好在composite class里。