Whist Game Design Report

By Edward Marozzi (910193), Fei Yuan (913503), Jake Hum (914666) - Team 1

Introduction

Whist is a 4 player Graphical User Interface card game developed and released by NERD Games Inc. The game was developed for human players to play against Non-Playable Cardplayers (NPC). NPCs essentially have to select cards automatically to partake in the game. However, the current implementation by NERD Games Inc simply has this done by random selection, which may prove to be ineffective in playing with human players and will likely violate the rules of the game. Our team has been brought in to extend the system to include a new NPC player that has a card filtering stage and selection stage with the aim to improve its overall card selection for the game. Our secondary focus for this engagement is to improve the games configurability and extensibility to enable future changes. Through a process of **iterative modelling** and **refactoring**, while considering established design principles and patterns, our team has successfully achieved these goals. This report will outline our final solution, the patterns and principles that we applied, as well as the various alternatives approaches that were considered throughout this process.

Description of Design

Summary

The main changes to the existing software are as follows:

- Created an abstract player class to handle player functionality of the game, which the human player and NPC player extend.
- Created two interfaces for NPC card filtering and selection strategies. Congruently, we also created the associated strategies to enable the new desired behaviour for the NPC player.
- Created a properties file system to allow properties of the game to be customised and read into the game.
- Refactored some logic into methods and documented the code properly

The rationale for these changes will be described in the following section, along with the patterns and design principles that have guided these design choices.

Patterns and Principles

Abstracted player class

The original implementation of whist handles the entire game within the whist class, which includes all the functionality of both the Human and NPC players. This resulted in a set up that had extremely low cohesion, repeated code within the class and stood to benefit from **GRASP** and **Gang of Four software design principles**. During the early stages in our domain modelling, we identified the player as a stand alone entity within this system, as seen in our **Static Domain Model diagram**. This served as the catalyst for our team to look into extracting the player functionality out of the whist class to be in line with this.

In our initial implementation of the system, we had set up just one concrete player class that had both the functionality of the human class and the NPC class. However, this reduced the cohesion of the player class as it needed to check the type of the player before performing certain actions and contained functionality for both types of players. Furthermore, this limited the extensibility of the system in terms of adding new kinds of players as the player class would need to be greatly modified.

In our final implementation, we decided that an abstract class Player with a Human class and NPC (AI) class **inheriting** from it would best suit NERD Game Inc's current and future needs. This implementation improved the **cohesion** among the classes and allows for further **extensibility** in the future by allowing new types of players to be easily introduced into the system through the introduction of new classes that inherit from the player class. The use of **polymorphism** in this implementation also allowed us to remove the check for the type of player and further improve the **cohesion** of the player class.

As seen in our design class diagram, we not only took out the functionality of choosing cards to the player, but we also moved out the storing of relevant player information, such as score and hand, into the player. This decision was guided by the **information expert principle** as the player should hold the information that is relevant to itself, instead of the whist class as in the original implementation.

Interface for NPC Card filtering and selection

For the functionality of choosing a card we broke the process into filtering and selection parts. We used two **strategy patterns** to implement the filtering and selection processes and a **singleton factory pattern** to produce both strategies based on a **configurable properties** file.

This design enables the client to add new ways of filtering and selecting cards in the future by simply adding a new class that implements the IFilterStrat and ISelectStrat classes.

The **singleton factory** class **encapsulates** the **strategy creation** for the NPC players, with different NPC's being able to be configured to use different strategies for a more variable user experience. By using the **factory** and **strategy** design patterns we utilised the GRASP principle of **Pure Fabrication**. We decided this class needed to be fabricated as once when we completed our **Static Design Model** diagram it became apparent that the class would be necessary to simplify strategy creation despite not appearing in the **Static Domain Model** diagram.

The use of the **Strategy Pattern** interfaces improves the extensibility of NPC behaviour and provides a **low coupling** link to the Whist class via the interface. Furthermore, the interfaces utilises **polymorphism** to enable the each NPC to choose cards with a different strategy via the same method call reducing repeated code.

Implemented properties system

The game has been designed so that it functions with many settings such as board height/width, number of players (2-4), different configurations of Human and NPC players, different AI strategies, different random seeds, number of cards in hand, score to win and more. To easily configure these various properties they are read in via properties files. This means that to change the settings of the game the client can create and modify settings to their desired configuration or choose between several presets provided.

These parameters are set before the game is constructed and then used throughout the program ensuring **flexible** and **reusable code** structure, avoiding hard coded parameters where possible.

Changes we didn't implement

We contemplated abstracting out all of the graphics related code such as Hand, Trick and score locations to a separate class. Upon beginning to implement this change we found that the Whist, UI and JCard library classes were highly coupled and added no improvements to the system as a whole and thus could not be justified. Furthermore as the Whist class extends the JCard CardGame class, some features had to remain in the Whist class and thus we decided it should be the information expert in the game and game layout.

Another change that didn't make it into the final design was the abstraction of cards to a separate class called WhistCard. This class contained the Rank and Suit enums and a few methods relating to the cards. The aim of this was that if the client wanted to use a different type of card it would be easily implemented in class and the main class would not have to be modified. However we found that this class was confusing as the static WhistCard class and JCard's Card class had no clear distinction and ultimately made code more verbose (e.g Rank to Whist.Rank). These problems could potentially be fixed but we decided that it would be unlikely for the game to be played with a different deck of cards than the standard card deck and the negatives outweigh the positives.

Smart Selection Approach

The smart selection approach taken, is a simple algorithm that decides if the NPC can play a card that would make them winning for that round, if not the algorithm will choose the lowest rank card available so as to not waste a high ranking card.

First the algorithm will check if the current winning card is not the same suit as the trump suit. If not, the algorithm will see if it can play a card of the lead/winning card suit of greater rank and thus be in the winning position. If it doesn't have a card that satisfies that condition, it knows it can take the lead by playing any trump card, thus if it has a trump card it will play the lowest rank trump card in the hand.

If however, the winning card is of the same suit as the trump card then we can only take the lead by playing a trump card of greater rank than the current winning card. If the NPC has no such card it knows it is not possible to take the lead this turn and thus the lowest rank card in the hand is selected as it is the least valuable.