Kiet Do

Hydra Design Document

I. Introduction

This program is an implementation of the board game Hydra, a board game between players with the objective to get rid all the cards the fastest by playing them on heads.

II. Overview

For this program, there are 4 main classes: GameDriver, Player, Head, Card.

I made changes to the original UML Diagram. At first, I let a string represent a card. However, for the purpose of the game, we need to compare the values of the card, which will not be possible when using strings. We also

The essence of the game is the playing cards that will be used. Since each playing cards are different and with different properties during the game, therefore a Card class is created.

The Card class contains the following attributes:

- **name** – name of the card
- **value** – the value of the card from 1 to 13 (and Joker)
- **joker** – if the card is a joker or not

Each a game of Hydra, there need to be 2 or more players to compete each other, with a draw pile, a discard pile, a reserve for a card, therefore, the Player class contains the following attributes:

- **draw_pile** – the vector of cards, representing the draw pile of a player
- **discard_pile** – the vector of cards, representing the discard pile of a player
- **current_card** – the current card is the top card of the draw pile
- **reserve** – the location where the player can store a card of their choice
- **bool testing_mode** – whether the players are in testing mode or not

The final objective of the game is to have no cards left in the draw pile, the discard pile, and the reserve. Each player will have to play a certain number of cards, accordingly to the number of heads on the table. A head is a stack of cards, and the card on top is the only thing the players will

care about when it is their turn. Therefore, the Head class is created and contains the following attributes:

- **pile** – a vector of cards
- **head_index** – the head index of the Head class

We have the Cards class, the Players class, and the Heads class. However, we still need to put all of these into one single game, under a class call GameDriver, with the following attributes:

- **no_players** – the number of players in a game
- **players** – a vector of Players
- **heads** – a vector of Heads
- **testing_mode** – whether the game is in testing mode or not

GameDriver class is responsible for creating a game of Hydra, which has a certain number of players, a changing number of heads. To store the information of each player, a vector of the class Player is created as an attribute of the GameDriver class, in other words, the GameDriver class owns the Player class.

Then, we have the Utils class, which contains the function custom_cin(), which will handle exceptions and EOF. The enumeration GAME_STATE is also used to make the code easier to read.

I will generally describe about the flow and logic of this structure.

In the main class, we will ask the user the number of the players in a game and whether we want the testing mode to be on or not. Using the aforementioned information, we will then create a new game, and then invoke the public function start_game to start playing the game.

The function distribute_to_player() will then be called, and it will create a deck of cards with the necessary number of cards, shuffle them and distribute them to each player equally. Then, we will go into a loop, and start with the first player, and go in a while loop and call the function start_turn() on that player. That function will return 1 if the player win on that turn, and the while loop will be break, and we will print out the winner of the game.

III. Design

There are multiple design challenges in the program. At first, when planning the first UML diagram and coding the first part of the code, I realized that even though a string can represent a card, but it would be difficult to compare them to fit the logic of the game. Therefore, I created a class call card, and assign each card a value. During initialization, we will assign the value for all the cards based on its name (Value of 3 for 3D, Value of 11 for JD, Value of 2 for Joker, …). We can then use the value to compare easily with the heads.

I also tried making my program that allow the user to end their game whenever they want. As there are too many loops and logics to make that, so I created an additional class called Utils, and implement a function where it will handle the invalid input and EOF cases, so the user can exit the game whenever they want.

IV. Resilience to Change

If the rules of the game changed, the function check_rules of GameDriver can be changed to better fit the rule requirements of the game. If the card properties change, we can easily change the value of the cards or the name of the card easily during initialization.

The classes have a lot of small functions, meaning that they have high cohesion, and therefore it can be reused or repurpose depending on the purpose of the game.

V. Answers to Questions

Question: Jokers have a different behavior from any other card. How should you structure your card type to support this without special-casing jokers everywhere they are used?

I created a class called Card that represents each card in a deck. The Card class will have a name, in this case, the name “Joker”. We will add a value and joker attribute. During the initialization of a card, if the name is “Joker”, we will set the value of said card to 2, and the Boolean joker to true. Whenever we want to change the value of the Joker, we will only change its value to a set number, without altering its nature (being a joker) and its name. We will freely be able to compare them cards of other values, without worrying about any edge cases.

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

We will implement all classes and functions in as much detail as possible, meaning that each action of each class will have its own function, and they will be as simple as possible, so that we can reuse them if we add a new rule or a new interface.

In our class design, we have three different classes that are the main part of the game: the GameDriver class that controls the game, the Player class that represents each player of the game, and Heads class that represents the heads in the game. Each classes have specific functions that will allow us to reuse them in case we need to add a new rule or change the interface.

Question: 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?

We will create 2 new classes called HumanPlayer and ComputerPlayer, and they both inherit the Player class. If the player(s) want a Player to be a computer player, they can change it from the HumanPlayer to ComputerPlayer class. Then when the playing of the games are happening, we can check whether the player is a ComputerPlayer or a HumanPlayer class.

Question: 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?

We will create 2 new classes called HumanPlayer and ComputerPlayer, and they both inherit the Player class. At the start of the game, all players will be HumanPlayer class. During the game, if a player decides to stop playing, we will call a function in GameDriver class that switches the HumanPlayer to ComputerPlayer class, and the ComputerPlayer class will automatically play the game.

VI. Extra Credit Features

For the extra credit features, I was able to correctly implement the grammar correctly, with additional information that provide much more details about what the input, the output, the game was instead of just displaying them.

I also implemented a feature that allows the user to exit whenever possible with the command Ctrl + D or EOF, using in the function custom_cin in the class Utils. That class will take in a type of data and a question, and with that, we will check if the input is of the right format using the if statement with condition cin.fail() and cin.eof(), all in a while loop, so that if the user type in an incorrect input, they can try again.

In the testing mode, I also allow the user to input the number of cards each player can have. This will make testing more flexible and easier for the testers and users.

VII. Final Questions

1. 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?

For writing large piece of program, the most important thing to look out for is the length of the program, and if the program is just a single main class, which would be extremely difficult to read, understand and implement or fix the program. Also, I will also try to split the logic of the program into smaller classes that are easier to understand. That way, we have a lower coupling and higher cohesion in our code. Lastly, each important function when created need to be tested, instead of finishing the whole program before testing. This will make testing much more difficult and waste a lot of time.

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

I will divide the GameDriver class into even smaller functions, so that we can have even higher cohesion for future change or implementation. Otherwise, I will not change a lot about my program or change the structure of my program drastically.

VIII. Conclusion (if needed)

Overall, this program was quite challenging as the game itself is quite hard to comprehend for me. So, I took a whole day to understand it, watched the video to understand more about Hydra. Then, I planned the structure and the UML Diagram of this project. However, during the implementation, I realized that my implementation with cards as string will need a lot of edge cases and logic to handle (if, switch statements), therefore, I changed a card to a class called Card, with all the necessary attributes (values, name, joker), making handling logic of the game rules much simpler.