Signature Assignment for CSC 520

Prepared by

Hiral Jasani – ?????

Xiaofang Yu – 93842

Xiaoya Luo – ?????

International Technological University

April 2018

1 Introduction

2 Product survey

3 Specifications and models

4 Design and Implementation

4.1 Design Overview

This program will work as a poker analyzer. In particular, the program will take a random pick of five cards and determine which category describes the hand.

To be clear, the categories are predefined. Even though most of them are mapped to the categories of the porker card game, such as Texas Hold'em, these given categories form a variant of porker hands of Texas Hold'em. So, we are not developing a complete poker game. In particular, we are not going to follow the porker competition rules, mimic the players' actions, use gaming strategies, or calculate results.

Since this signature assignment is aimed at testing proficiency in various offering of Course CSC 520, such as software engineering, function designing, Python programming, and so on. While the project is not aiming to develop a complete game, the design will use the Object Oriented Programming technology and plan for future development. That is to say, we will build the components organically as extendable classes so that the program can easily be extended and developed in the future.

4.2 The Categories Involved

While different versions of poker can involve different hand categories, the categories we'll discuss in this section is a variant of the standard category set in many games, including Texas Hold'em.

Poker is generally played with a standard deck of fifty-two cards, and a poker hand typically consists of five of these cards. The number of possible hands is

$$\binom{52}{5} = 2{,}598{,}960$$

The following Table 1 and Table 2 list standard categories and categories involved in this project respectively. In the tables, we also list the number of ways that various poker hands can be obtained and the probability of getting these hands, which is simply the number of ways it can be obtained divided by $\binom{52}{5}$. In a standard game, the less probable a hand is, the higher we rank it, so a straight flush beats four of a kind, which in turn beats a full house etc. In standard versions of poker no preference is assigned to any particular suit, and so ties are possible (later, we will explain suits in detail).

Category Name	Number possible	Probability
Straight flush	40	.000015
Four of a kind	624	.000240
Full house	3744	.001441
Flush	5108	.001965
Straight	10200	.003925
Three of a kind	54912	.021128
Two pairs	123552	.047539
One pair	1098240	.422569
Bupkis	1302540	.501177

Table 1: Standard categories of poker hands

Category Name	Number possible	Probability
Four of a kind	624	.000240
Full house	3744	.001441
Three of a kind	54912	.021128
Two pairs	123552	.047539
One pair	1098240	.422569
Ranks-all-different	1317888	.501177

Table 2: Categories of poker hands involved in this project

As you can see from the above tables, the difference between Table 1 and Table 2 is that the Ranks-all-different category of the variant version combines Straight flush, Flush, and Straight of the standard version. This greatly reduces the difficulty of the implementation, not just because of the smaller collection of categories, but also because of the elimination of the suit factor.

As we all know, the four cards suits used primarily in the English-speaking world are diamonds (♦), clubs (♣), hearts (♥) and spades (♠). Normally, the suit is a key factor to determine the rank of a hand. While in this project, the suit factor will be ignored since categories related to the suit factor are all merged into the Ranks-all-different category and thus eliminated.

To be clear, Table 3 lists some examples for each categories involved in this project.

Category Name	Example
Four of a kind	A♦, A♣, A♥, A♠, K♠
Full house	Q♠, Q♣, Q♥, J♦, J♠
Three of a kind	10♥, 10♣, 10♥, 9♦, 9♠
Two pairs	8♠, 8♣, 7♥, 7♠, 6♥
One pair	5♥, 5♣, 4♣, 3♠, 2♥
Ranks-all-different	J♣, 4♦, 9♣, 3♠, 7♥

Table 3: Examples of involved Categories

4.3 Implementation overview

The goal of this project is very clear: given a random pick of five cards from 52 standard cards, take it as your hand and determine which of the above six categories describes your hand.

4.3.1 Input

The file *DeckOfCardsList.dat* is a pickled binary file containing a list of the 52 cards in an ordinary deck of playing cards. The following program is given and it randomly selects five cards from the deck and displays them.

```
import random

import pickle

…

infile = open("DeckOfCardsList.dat", 'rb')

deckOfCards = pickle.load(infile)

infile.close()

pokerHand = random.sample(deckOfCards, 5)

print(pokerHand)
```

The above code will be used directly to randomly select five cards from the deck of 52 cards. These five cards, the *pokerHand* above, will be displayed and then be taken as the input of our program.

For example, a random input would be string collection and be displayed as below:

['A♥', '7♦', 'Q♠', 'A♣', '7♥']

4.3.2 Output

The output will be very straightforward and just display the name of the category that can describe the given hand.

Following the Object Oriented Programming paradigm, the category class will have a *categoryStr* method which will be called to generate the output string.

4.3.3 Method overview

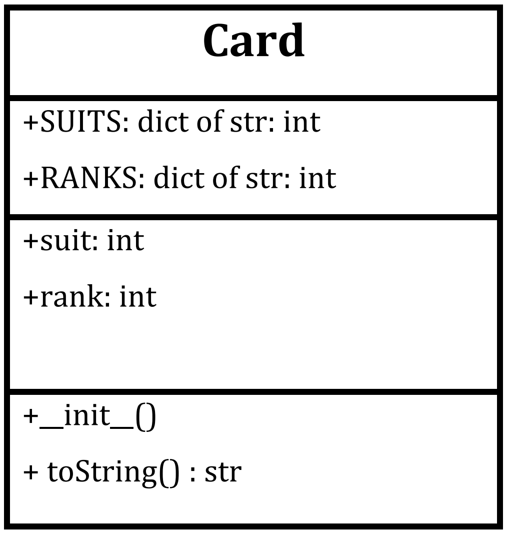
In short, the program will firstly filter out the suit info and generate a collection of the card ranks in reverse order, and then it will determine the number of each rank in the hand and all possible cases.

1). Create a Card class to store information of each card. There will be 5 Card instances in total.

2). For each Card object, extract the rank (not the suit) and save the rank into an array.

3). Iterate the array to find out the number of different ranks and the number of cards of each rank, and determine its category.

4.4 The Card class



The above is the UML diagram for the Card class. First, we will create two static dictionaries for ranks and suits respectively.

Suits:

Clubs	(♣)	- 1
Diamonds	(♦)	- 2
Hearts	(♥)	- 3
Spades	(♠)	- 4

```
SUITS = {'♣': 1, '♦': 2, '♥': 3, '♠': 4}
```

Ranks:

Numbers	(2-10)	- 2-10
Jack	(J)	- 11
Queen	(Q)	- 12
King	(K)	- 13
Ace	(A)	- 14

```
RANKS = {'2': 2, '3': 3, '4': 4, '5': 5, '6': 6, '7': 7, '8': 8,
'9': 9, '10': 10, 'J': 11, 'Q': 12, 'K': 13, 'A': 14}
```

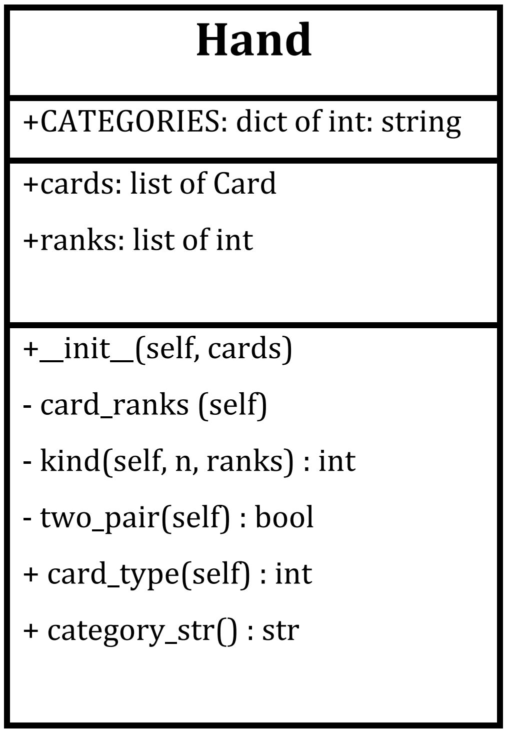
The Card class provides an efficient way of representing all 52 cards by two member properties: one is for the rank and the other is for the suit.

```
class Card:

    def __init__(self, cardInfo):

        self.suit = self.SUITS[cardInfo[:-1]]

        self.rank = self.RANKS[cardInfo[-1]]
```

To solve the problem assigned to this project, further member functions are not needed. But, for future development, we can create a function to transform a card back to a string for displaying purpose.

```
def __toString__(self):

    text = ""

    if self.rank < 0:

        return "Joker";

    elif self.rank == 11:

        text = "J"
    elif self.rank == 12:

        text = "Q"

    elif self.rank == 13:

        text = "K"

    elif self.rank == 14:

        text = "A"

    else:

        text = str(self.rank)




    if self.suit == 0:    #D-Diamonds

        text += "♦"

    elif self.suit == 1:  #H-Hearts

        text += "♥"

    elif self.suit == 2:  #S-Spade

        text += "♠"

    else:                 #C-Clubs

        text += "♣"


    return text
```

4.5 The Hand class



The above is the UML diagram for the Hand class.

First, we will create a static dictionary for categories involved:

```
CATEGORIES = {0: "Ranks-all-different",
              1: "One pair",
              2: "Two pairs",
              3: "Three of a kind",
              4: "Full house",
              5: "Four of a kind"}
```

The Hand class will hold a list of five cards as well as the category that describes those cards.

```
class Hand:

    def __init__(self, cards):

        self.cards = cards

        self.ranks = []
```

The private function *card_type ()* will be the key component for this project since the function will analyze and determine which type the Hand belongs to.

```
def card_ranks(self):

        """Return the card ranks in reverse order

        without the suit"""

        for card in self.cards:

            self.ranks.append(card.rank)

        self.ranks.sort(reverse=True)



    def kind(self, n, ranks):

        """ Return the n of a kind

        Return None if not found"""
        for r in ranks:
            if ranks.count(r) == n:
                return r
        return None

    def two_pair(self):
        """Return True if it is two pair, False otherwise"""
        pair = self.kind(2, self.ranks)
        another_pair =self.kind(2,list(reversed(self.ranks)))
        if pair and another_pair != pair:
            return True
        return False

    def card_type(self):
        self.card_ranks()
        if self.kind(4, self.ranks):
            return 5
        elif self.kind(3, self.ranks) and self.kind(2, self.ranks
):
            return 4
        elif self.kind(3, self.ranks):
            return 3

        elif self.two_pair():

            return 2

        elif self.kind(2, self.ranks):

            return 1

        else:

            return 0
```

For reporting purpose, we will need to print a readable category name by calling the *category_str()* function. This function is easily transform the category value, an integer, to a string using the *CATEGORIES* dictionary.

```
def category_str (self,category):

        return self.CATEGORIES[category]
```

4.6 Main workflow

4.6.1 Get input

As mentioned before, a pickled binary file called *DeckOfCardsList.dat* will be used to generate a random pick of five cards as the input. In particular, the *DeckOfCardsList.dat* file contains a list of the 52 cards in an ordinary deck of playing cards. We will use the following code to randomly select five cards from the deck and displays them. These five cards' info will be stored in a string array, the *pokerHand* below, which will then be used as the input of our program.

```
infile = open("DeckOfCardsList.dat", 'rb')

deckOfCards = pickle.load(infile)

infile.close()

pokerHand = random.sample(deckOfCards, 5)

print(pokerHand)
```

4.6.2 Analyze data and create objects

Traverse the input string array and build a collection of five Cards from it. Use the Cards collection to initialize an instance of Hand.

In total, there will be five instances of the Card class and one instance of the Hand class. And the category will be calculated as the result we will display later.

```
hand = Hand([Card(hand) for hand in pokerHand])
```

We also included a test every time we run the program to ensure accuracy:

```
def test():

    “Test cases for poker game"

    assert(Hand([Card('2♦'), Card('3♣'), Card('K♥'), Card('5♠'),
Card('7♣')]).card_type() == 0)

    assert(Hand([Card('2♦'), Card('K♣'), Card('K♥'), Card('3♠'),
Card('7♣')]).card_type() == 1)
```

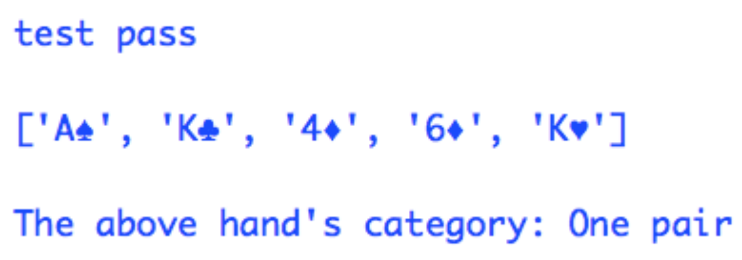
```
    assert(Hand([Card('2♦'), Card('2♣'), Card('3♥'), Card('3♠'), Card('7♣')]).card_type() == 2)
    assert(Hand([Card('2♦'), Card('3♣'), Card('3♥'), Card('3♠'), Card('7♣')]).card_type() == 3)
    assert(Hand([Card('2♦'), Card('3♣'), Card('3♥'), Card('3♠'), Card('2♣')]).card_type() == 4)
    assert(Hand([Card('2♦'), Card('3♣'), Card('3♥'), Card('3♠'), Card('3♣')]).card_type() == 5)
    return '\ntest pass'
```

4.6.3 Output result

Easily call the public *category_str()* function to get the output string and print it to the screen.

```
category = hand.card_type()
print(hand.category_str(category))
```

Below is a screenshot of an output example:



```
test pass
['A♠', 'K♣', '4♦', '6♦', 'K♥']
The above hand's category: One pair
```

5 Testing

6 Conclusion

6.1 Project overview

This project is mainly to solve the Signature Assignment which is aimed at testing student proficiency in various offerings of the course CSC 520 – Python Programming.

With this purpose, the problem is pretty straightforward and could be a sub-problem of the Poker game, such as Texas Hold'em. In particular, the signature assignment predefines a collection of categories and requires the program to determine the category of a random hand of five cards.

To solve this problem, we firstly used an easy way without creating any class. That is, directly analyze the input to find out the number of ranks and the frequency of each rank, and then determine which category fits in.

To some extent, this signature assignment reflects an important step of the poker game, so we considered more and decided to make it more extendable. Specifically, we used the object oriented programming technology and built Card class and Hand class. By doing this, the program is more flexible and can be extend to more categories and even a complete poker game.

In short, this project is fully tested and can successfully provide all required functionalities. More than that, the project is built using the XXX software engineering model and object oriented programming paradigm, so we can further develop this extendable project to more complex games.

6.2 Ideas about future development

As mentioned above, we intended to make this project extendable. Below is some ideas that we can develop in the future.

- Take the suit factor into consideration and extend the category collection to a standard set of category (Table 1).
- Extend the Card and Hand class by adding new members to handle the new categories and game rules.
- Adding a player class that represents the game players (could be 2 to 10). We can also develop a derived class called Dealer who is a host and special player.
- Create a Deck class so that we can track and manage all 52 cards.
- Finally, a Game class should be built to run the game, control the workflow, manager all the players, and determine the results.

7 Reference

http://www.math.tamu.edu/~phoward/m442/pokerproj.pdf

http://www.cs.nott.ac.uk/~pszgxk/ugyear2/projects/2001-02/dissertations/gp-gxk2.pdf

https://en.wikipedia.org/wiki/Texas_hold_%27em