FIT9133 - ASSIGNMENT 2

Monash University – Sep 2017

Full name: Hoang Thi Bich Ngoc

Student ID: 28496337

Tutor name: Md Samiullah

Assignment 2: Not Freecell

Table of content

I. Introduction

1. Free Cells game

The program is designed for developing a card game called "Free Cells" by Python Language. Free Cells was first developed by Microsoft on the Windows operating system. It was born in 1995 and until now, Free Cells is still one of the most well-known games among Microsoft users. In the recent years, along with the many different versions of Free Cells have been developed and tested in both Window operating system and other game websites. Basically, the game mentioned in this document follows the traditional game with 52 cards in a deck and the rules and mechanism found at https://en.wikipedia.org/wiki/FreeCell. This program support creating random deck, therefore, some decks are unsolvable.

2. Code ideas

As mentioned above, the game ideas based on the traditional game. Besides that, few parts of my program follow the coding style of my lecturer Gavin Kroeger at week 8 with the Chess Game example. I would like to say thank you to my lecturer Gavin and my tutor Samiullah for your knowledge and support during classes and consultation sessions. The report below will first explain more details about how the programs implemented based on classes and methods and give a guidance for user to play with Free Cells game.

II. Code structure

1. Code structure diagram

The game constitutes of 5 interrelated classes including card, deck, foundation, freecell and NotFreecell game. In general, the effect of each classes could be described as follows.

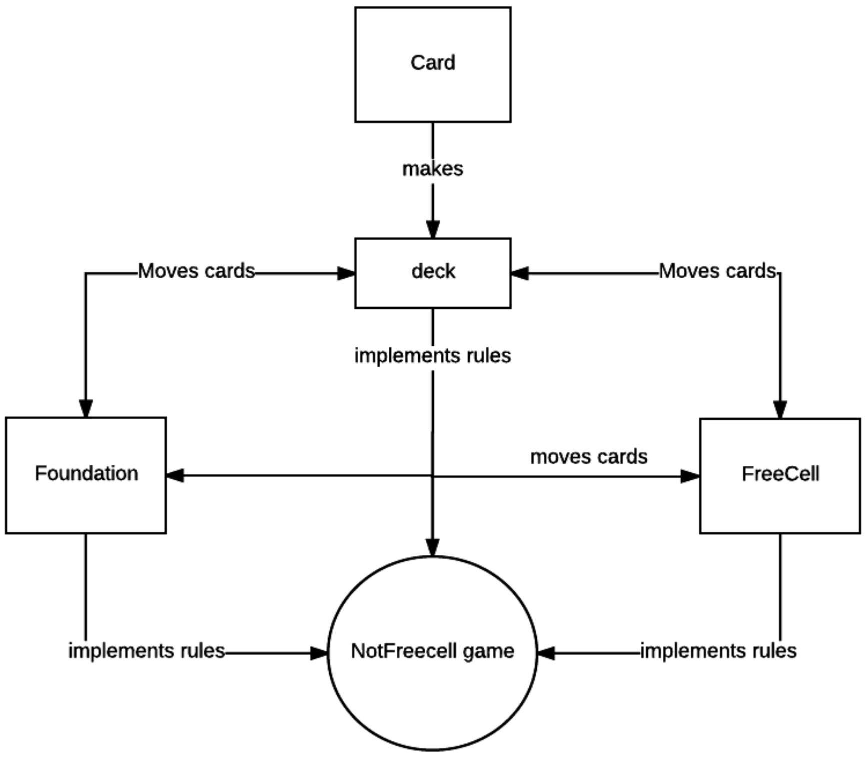

Image 1: Code structure of game Freecell

As can be seen, card is the most fundamental class. It is designed with 2 main attributes – card suit and card face. Card is used as unit for building deck game. Deck class is where the initial deck game is created. Besides the 52 card-deck, this class supports other customized decks with less cards inside. Also, Deck is created with shuffle method to ensure cards distributed randomly when a game started. In addition to Deck, Freecell and Foundation are places for moving cards in and out during game. As a result, 3 basics methods (remove card, add card, get card) are particularly designed for each class depended on the Free Cells' rules. To be likely to a real game, methods designed allow a card to move easily between the deck, 4 free cells and 4 foundations whenever a player calls the card location. NotFreecell class is written to implement the main game. It classifies card movement into 9 types based on 3 pickup locations (free cell slots, deck cascades) and 3 corresponding destination locations. Inside NotFreecell, the main function plays a role of running the whole game with requests for input from users and controllable loops. The loop breaks only when the user quit the game or win the game.

2. Class information

a. Card

Card is a fundamental unit for this game. Class Card represents a card in play. It contains 2 main attributes including the face value (face) and suit of the card (suit) respectively.

Attribute:

- *Face*: Card face (From Ace to King)
- *Suit*: Card suit (Heart(H), Club(C), Diamond(D), Spade(S))

***Method*:**

- *__init__(self, face, suit):* Initialize the card with face and suit attributes
- *__str__(self):* return card as string
- *get_face(self):* return a card's face as string
- *get_suit(self):* return a card's suit as string
- *__repr__(self):* Represent the card

b. Deck

A Deck is made up of cards. Each deck contains 8 cascades. Inside class Deck, a card set will be created based on user the start value, the end value and the number of suits. From this card set, a randomized deck will be initialized. Moreover, methods for adding card, removing card and get card from deck also written inside class Deck.

***Attribute*:**

- *value_start:* Start value
- *value_end:* end value
- *suit_number:* Number of suits

***Method*:**

- *__init__(self, start=1, end=13, n =4)*: Initialize the deck. Create an empty deck with 8 cascades. This function has default value (start=1, end=13, n =4)

- *Shuffle(self):* Create a shuffled card set with the same start value, end value and number of suits, then distribute this card set to that empty deck
- *__str__(self):* return a deck as string (a matrix)
- *get_card(self,y):* Get a card from a particular cascade. Only the last item of the required cascade could be gotten. If the cascade is empty, this function returns empty value ('--:-'). Variable y is the location of the cascade that a user wants to get a card in.
- *add(self,y, the_card):* Add a card to a particular cascade. If this cascade is empty, card can be added to the required. If it is not empty, only add to the required cascade when face and suit are suitable. Variable y is the location of the cascade that a user wants to add a card in. Variable the card is the card added by this user.
- *remove(self,y):* Remove a card from a particular cascade. Only the last item of the required cascade could be removed. Variable y is the location of the cascade that a user wants to remove a card from.

c. *Freecell*

Class Freecell is written for representing 4 free cells in an actual game. In fact, there are 4 free cells that cards can be moved in and out. Class Freecell creates a list with 4 empty items playing the role of free cells. Similarly, three methods for card movement (add, remove, get) also created inside Freecell.

***Method*:**

- *__init__(self):* Initialize the Freecell. Create a list with 4 empty slots.
- *__str__(self):* Return Freecell as string
- *get_card(self,y):* Get a card from a particular free cell. Variable y is the location of the free cell that a user wants to get a card in.
- *add(self,y,the_card):* Add a card to a particular free cell. If the required free cell is empty, a player can add any card to this free cell. Variable y is the location of the free cell that a user wants to add a card in. Variable the card is the card added by this user.
- *remove(self,y)*: Remove a card from a particular free cell. Variable y is the location of the free cell that a user wants to remove a card from.

d. *Foundation:*

Class Foundation used to generate 4 foundations and set its corresponding movement methods. In particular, each foundation is a list of same suit cards. Cards move into the foundation in the order from the smallest face (Ace) to the highest face (King).

Method:

- *__init__(self)*: Initialize the foundation, create a list of 4 lists called 'foundation'
- *__str__(self)*: Print foundation as a string. It only prints the last item of a foundation
- *get_card(self,y)*: Get a card from a particular foundation. When a player tries to get a card from a foundation, only the last card of this foundation is selected. Variable y is the location of the foundation that a user wants to get a card in.
- *add(self,y,the_card):* Add a card to a particular foundation. If the foundation is empty, only Ace can be added to this foundation. If the foundation is not empty, the next added card must have the same suit and the consecutive value. Variable y is the location of the foundation that a user wants to add a card in. Variable the card is the card added by this user.

- *remove(self,y):* Remove a card from a particular foundation. Only the last item of this foundation can be removed. Variable y is the location of the foundation that a user wants to remove a card from.
- *win(self)*: Check the game result. If the last item of 4 foundations is King, the player wins.

e. ***NotFreecell***:

Class NotFreecell inherits methods written in 4 mentioned classes. In other words, this class tells how a card move in and out from one place to another place. At the end, the main function is responsible for implement the whole game by the user input. Main function contains some 'while loop' to control user input and keep the game running until the player win or quit the game.

Method:

- *__init__(self)*: Initialize card deck, free cells, foundations
- *__str__(self)*: Display card deck, free cells, foundations
- *move_to_deck(self, from_y, to_y, pickup)*: Use to move to deck from deck, foundations, free cell. Variable from_y is the pickup location. Variable to_y is the destination location. Variable pickup is the pickup options (deck, freecell or foundation)
- *move_to_freecell(self, from_y, to_y, pickup)*: Use to move to a free cell from deck, foundations, free cells. Variable from_y is the pickup location. Variable to_y is the destination location. Variable pickup is the pickup options (deck, freecell or foundation)
- *move_to_foundation(self, from_y, to_y, pickup)*: Use to move to foundation from deck, foundations, free cells. Variable from_y is the pickup location. Variable to_y is the destination location. Variable pickup is the pickup options (deck, freecell or foundation)
- *result_check(self)*: Check the result. It based on the win function of Foundation class.

III. User guidance

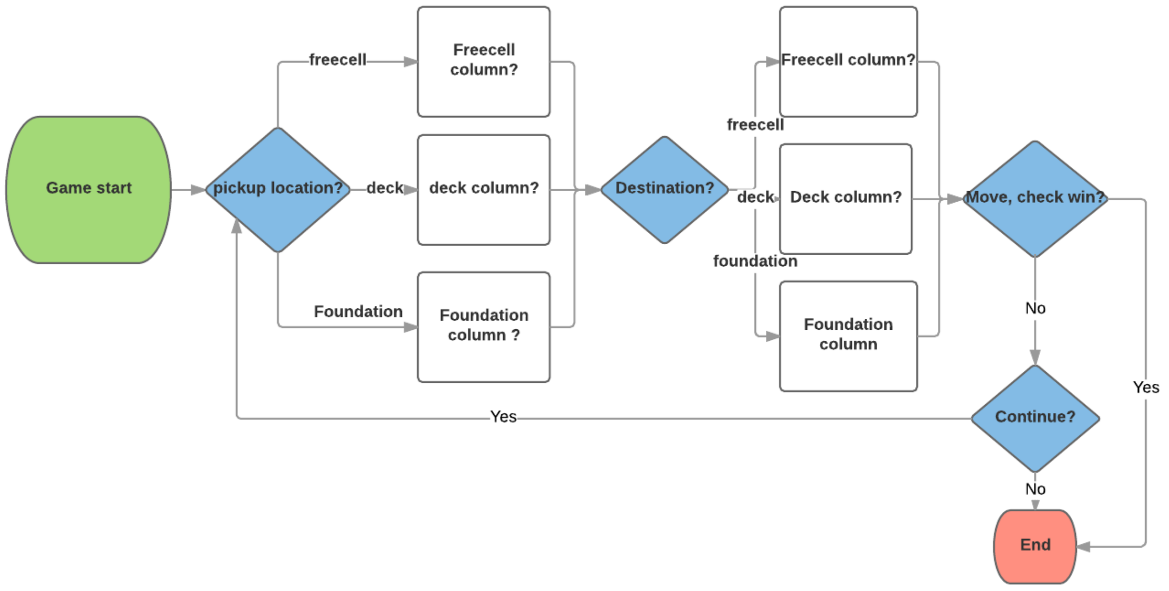

Image 2: Use case diagram

This part describes how a user plays the game. As can be seen from the diagram, a user give input by answering a series of question.

1. What is the pickup location? A user will choose among 3 pickup options (freecell, cascade, foundation).
2. What is the pickup column? A user will give a number as an input, start with 0 and end with 7 (for deck option) or 3 (for freecell and foundation options)
3. What is the destination location? A user will choose among 3 destination options (freecell, cascade, foundation).
4. What is the destination column? A user will give a number as an input, start with 0 and end with 7 (for deck option) or 3 (for freecell and foundation options)
5. Check win or not? It is not a question for the player. In the background, the code will check whether this user wins or not. If yes, the game finishes.
6. Do you want to continue? This question checks whether this user wants to quit. They can press ‘N’ to quit the game and ‘Y’ to continue the game.

IV. Conclusion

This program basically simulates the traditional Free Cells games with 52 cards. Unfortunately, within the time limitation in assignment 2, the program is lack of some auto movement function and the user interface. While the real game enables suitable cards to move to foundations automatically, this program requires user inputs to move cards and only a card could be moved each turn. The graphic user interface is also not emphasized when this game written in Python.

In the future, this game could be improved by adding user interface to provide a fascinating user experience as well as include the auto movement functions to reduce data input time.