CSE 231 **Spring 2017**

Project 10: Baker's Game

Overview:

This project is worth 55 points (5.5% of your course grade). It requires knowledge of both classes and exceptions, and is due by 11:59 on Monday, April 17th. This project will take time; I recommend starting it early.

Baker's Game is a Solitaire game similar to FreeCell, which differs in the fact that sequences are built by suit, instead of by alternate color.

Deliverables:

The deliverable for this assignment is the following file:

`Proj10.py` – the source code for your Python program

Be sure to use the specified file name and to submit it for grading via the handin system before the project deadline.

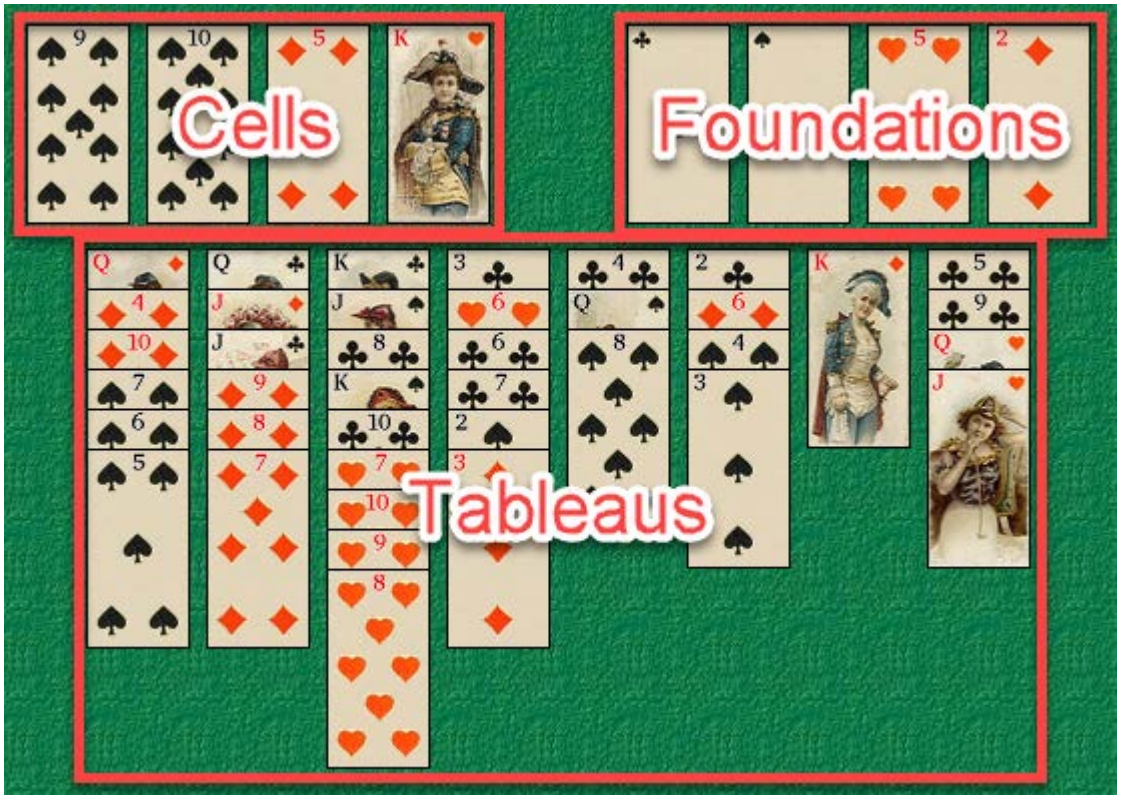
Understanding the Game:

The best way to understand any game is to play it. You can play Baker's Game online here:

http://www.247freecell.com/bakersGameFreecell.php. We suggest that you play around with this game for a little while so that you can understand what it is you need to implement. This is a very difficult game to solve; only 74.3% of solutions can even be solved. Thankfully, you won't have to worry about that. You only need to implement the game.

Important Terms:

Observe the image below from an in-progress game:



The top-left four locations are called Cells, the top-right four piles are called Foundations, and the 8 piles in the middle are called Tableaus. Cells allow only 1 card to be placed in them. Any card can be put in a cell. These act as maneuvering space. Foundations are built up by suit from Ace up to King. Tableaus are dealt out to at the start of the game. All cards start here. However, any cards that are moved to a tableau during the game must be built *down by suit*, e.g. if the 8 of hearts is at the bottom of a tableau column, you may only place a 7 of hearts on it ("placing 7 on 8" means that columns are built *down* and *by suit* means that you can only place a card on another card of the same suit).

Construction and Layout:

One standard 52-card deck is used.

There are four open cells and four open foundations.

The entire deck is dealt out left to right into eight tableaus, four of which comprise seven cards and four of which comprise six. Your program should deal one card to each tableau from left to right and deal this way until all 52 cards are dealt. *If you don't setup the tableaus this way, your project will fail all automatic test cases and you will get penalized*.

Building During Play:

Tableaus are dealt randomly from a shuffled deck, but during play tableaus must be built down by the same suit.

Foundations are built up by suit.

Moves:

Any cell card or top card of any tableau may be moved to build on a tableau, or moved to an empty cell, an empty tableau, or its suit's foundation.

Complete or partial tableaus may be moved to build on existing tableaus, or moved to empty cascades, by recursively placing and removing cards through intermediate locations.

Victory:

The game is won after all cards are moved in ascending rank by suit to their foundation piles.

Detailed Program Specifications:

You will develop a program that allows the user to play Baker's Game according to the rules outlined above. The program will use the instructor supplied cards.py module to model the cards and deck of cards. To help clarify the specifications, we provide sample output of a correctly implemented Baker's Game Python program below in the Sample Output section. We also provide a project10.py file that contains stubs of the required functions. The program will run as is, but will not do anything useful. You are encouraged to create additional functions as you see fit. Your program must use the import statement for cards.py; you are not allowed to copy the contents of cards.py into your proj10.py implementation. Also, you are not allowed to modify cards.py

Program commands:

The program will recognize the following commands regardless of case (i.e. TF, tf, Tf, and tF are all valid) -

TC x y	Move card from tableau x to cell y
TF x y	Move card from tableau x to foundation y
TT x y	Move card from tableau x to tableau y
CF x y	Move card from cell x to foundation y
CT x y	Move card from cell x to tableau y
R	Restart the game with a re-shuffle
H	Display this menu of commands
Q	Quit the game

Here, x and y denote column numbers. Tableau columns are numbered from 1 to 8, whereas

Cells and Foundation piles are numbered from 1 to 4.

The program will repeatedly display the current state of the game and prompt the user to enter a command until the user wins the game or enters q, whichever comes first.

The program will detect, report, and recover from invalid commands. None of the data structures representing the cells, tableaus, or foundations will be altered by an invalid command.

Function Descriptions:

We have already supplied stubs for functions that we require, but your implementation can include more functions if desired.

1. `setup_game() -> (cells, foundations, tableaus)`

setup_game takes no parameters. It creates and initializes the cells, foundations, and tableaus, then returns them as a tuple, in that order. The cells and foundations will be a lists of 4 empty lists, the Tableau will be a list of 8 lists, which will contain all of the cards dealt into 8 vertical columns from left to right as described before.

2. `display_game(cells, foundations, tableaus) -> None`

display_game takes four parameters, which should be the lists representing the cells, foundations, and tableaus. The cells and foundations should be displayed above the tableaus. A non-empty cell should be displayed as the card within it, whereas an empty cell should be displayed as [ ]. A non-empty foundation will be displayed as the top card in the pile (i.e. the last card moved to it), while an empty foundation will also be displayed as [ ]. The tableau is displayed below the cells and foundations, and each column of the tableau will be displayed downwards as shown in the sample below. An empty column will be displayed by whitespace. (Suggestions: (1) at first simply print each data structure, e.g. print(tableaus) in this function until you get the rest of the program working. Then come back and do the formatting last. (2) when you get to formatting, convert the card to a string using `str()`, e.g. `print("{:4s}".format(str(tableau[2][0])))`

3. `valid_fnd_move(src_card, dest_card) -> None`

This function is used to decide whether a movement from a tableau or a cell to a foundation is valid or not. It takes two parameters: src_card and dest_card. The card that you are trying to place into the foundation is the src_card and the card already in the foundation that you are trying to place onto is the dest_card. The function will raise a RuntimeError exception if the move is not valid.

Conditions for a valid foundation move:

If there is no dest_card and the src_card is an Ace.

If the src_card and dest_card have the same suit and the rank of the src_card is 1 greater than that of the dest_card.

4. `valid_tab_move(src_card, dest_card) -> None`

This function is used to decide whether a movement from a cell, foundation, or another tableau to a tableau is valid or not: It takes two parameters, src_card and dest_card. The card that you are trying to place into the tableau is the src_card and the card already in the tableau that you are trying to place onto is the dest_card. The function will raise a RuntimeError exception if the move is not valid.

Conditions for a valid tabeau move:

If there is no dest_card then any src_card is valid.

If the src_card and dest_card have the same suit and the rank of the src_card is 1 less than that of the dest_card.

5. `tableau_to_cell(tab, cell) -> None`

This function will implement a movement of a card from a tableau to a cell. It takes two parameters, both are lists. The first one is the source tableau column (i.e. from which you are moving a card) and the second parameter is the destination cell. This function has no return value. This function moves a single card and has to decide if a movement is valid or not. If there is an invalid move, it will raise a RuntimeError exception. When a user enters tc x y, this function will be used.

6. `tableau_to_foundation(tab, fnd) -> None`

This function will implement a movement of a card from a tableau to a foundation. It takes two parameters, both are lists. The first one is the source tableau column (i.e. from which you are moving a card) and the second parameter is the destination foundation pile. This function has no return value. This function moves a single card and calls valid_fnd_move() to decide if a movement is valid or not. If there is an invalid move, it will raise a RuntimeError exception. When a user enters tf x y, this function will be used.

7. `tableau_to_tableau(tab1, tab2) -> None`

This function will implement a movement of a card from one tableau column to another tableau column. It takes two lists as parameters. The first one is the tableau column from which you are moving a card and the second is the destination tableau column that you are moving a card to. This function has no return value. This function calls valid_tab_move() to decide if such a movement is valid or not. If there is an invalid move, it will raise a RuntimeError exception. When a user enters tt x y, this function will be used.

8. `cell_to_foundation(cell, fnd) -> None`

This function will implement a movement of a card from a cell to a foundation. It takes two parameters, both are lists. The first one is the source cell (i.e. from which you are moving a card) and the second parameter is the destination foundation pile. This function has no

return value. This function moves a single card and calls valid_fnd_move() to decide if a movement is valid or not. If there is an invalid move, it will raise a RuntimeError exception. When a user enters cf x y, this function will be used.

9. `cell_to_tableau(cell, tab) -> None`

This function will implement a movement of a card from a cell to a tableau column. It takes two lists as parameters. The first one is the cell from which you are moving a card and the second is the destination tableau column that you are moving a card to. This function has no return value. This function calls valid_tab_move() to decide if such a movement is valid or not. If there is an invalid move, it will raise a RuntimeError exception. When a user enters ct x y, this function will be used.

10. `is_winner(foundation) -> True/False`

This function will be used to decide if a game was won. A game has been won if all the foundation piles have 13 cards and the top card on each of the pile is a King. This function returns either True or False.

Some Mandatory Stuff:

1. The display_game() function output does not have to look exactly like the example, but it must generate legible output (it should look like you are playing the game).

2. You must use the try-except block to handle errors (the only try-except block needed is included in the sample), each error message must explain exactly what type of errors a user is making. There are many errors to consider and we will not list them for you. You should be able to think of them when analyzing gameplay. Please see the example.txt example for some example of errors.

3. You should use the RuntimeError's custom message option to throw and handle errors for all errors. The error message is placed in quotes and when the exception is caught in the main your message in quotes is passed as error_message in the except block and is then printed.

Example:

```
if everything looks fine:
   do whatever necessary
else:
   raise RuntimeError("Error: invalid command because of …")
```

Assignment Notes and Tips:

1. Before you begin to write any code, play with the game at http://www.247freecell.com/bakersGameFreecell.php and look over the sample interaction below to be sure you understand the rules of the game and how you will simulate the game.

2. We provide a module called cards.py that contains a Card class and a Deck class. Your program must use this module (by calling import cards). You should understand each line of this file and should not modify it.

3. Laboratory Exercise #11 demonstrates how to use the cards module. Understanding those programs should give you a good idea how you can use the module in your game.

4. We have provided a framework named proj10.py to get you started. Using this framework is mandatory. It runs as is, but does not do anything useful. Gradually replace the "stub" code (marked with pass and comments) with your own code. Modify and test functions one at a time—do not move on to a new function until existing functions are correct.

5. Displaying the tableaus in columns can be tricky. Start by implementing a very simple display function: You can easily label and display the cells and foundations on one line, and the tableau using seven lines, with each line displaying the cards in a row. Once you have the game logic working properly, modify your display function to display the current game state as described in the specification.

6. One thing that can help testing is to turn off shuffling when the game starts.

7. The coding standard for CSE 231 is posted on the course website:

https://www.cse.msu.edu/~cse231/Online/General/coding.standard.html. Items 1-9 of the Coding Standard will be enforced for this project.

8. Your program should not use any global variables inside of functions. That is, all variables used in a function body must belong to the function's local name space. The only global references will be to functions and constants.

9. Your program must contain the functions listed in the previous sections. You may develop additional functions as you feel is appropriate.

10. I recommend using None as a placeholder to send to the valid move functions when a card does not exist.

Sample Output:

Because the interactions are lengthy, one has been saved in a separate file named example.txt available in the project folder. It is a partial game, only meant to serve as an example of execution. It uses a modified cards file named cards1.py that has shuffling disabled so as for the execution to be the same every time.

Test cases:

Six test files and their outputs are provided in separate text files. Use cards1.py for tests 1-5 and cards6.py for test 6.

Scoring Rubric

```
General Requirements

 __0__ (5 pts)   Coding Standard
       (descriptive comments, mnemonic identifiers, format, etc...)


Tests

 __0__  (5 pts) Pass Test 1
                H, R, Q
 __0__  (5 pts) Pass Test 2
                TF
 __0__  (5 pts) Pass Test 3
                TT
 __0__  (15 pts)Pass Test 4
                TC, CT, CF
 __0__  (3 pts) Pass Test 5
                win
 __0__  (5 pts) Pass Test 6
                TT and CT to empty tableau
 __0__  (5 pts) Display in columns

 __0__  (7 pts) Handles invalid cases such invalid move, empty input, invalid
values for x and y, invalid range, missing one value, etc.

TA Comments:
```

Educational Research

When you have completed the project insert the 6-line comment specified below.

For each of the following statements, please respond with how much they apply to your experience completing the programming project, on the following scale:

1 = Strongly disagree / Not true of me at all

2

3

4 = Neither agree nor disagree / Somewhat true of me

5

6

7 = Strongly agree / Extremely true of me

****Please note that your responses to these questions will not affect your project grade, so please answer as honestly as possible.****

Q1: Upon completing the project, I felt proud/accomplished

Q2: While working on the project, I often felt frustrated/annoyed

Q3: While working on the project, I felt inadequate/stupid

Q4: Considering the difficulty of this course, the teacher, and my skills, I think I will do well in this course.

Q5: I ran the optional test cases (choose 7=Yes, 1=No)

Please insert your answers into the bottom of your project program as a comment, formatted exactly as follows (so we can write a program to extract them).

```
# Questions
# Q1: 5
# Q2: 3
# Q3: 4
# Q4: 6
# Q5: 7
```