CSE 231 **Spring 2022**

Computer Project #10

Assignment Overview

This assignment focuses on the design, implementation and testing of a Python program which uses an instructor-supplied module to play a card game, as described below.

It is worth 55 points (5.5% of course grade) and must be completed no later than 11:59 PM on Monday, April 18, 2022.
A 3 pts extra credit will be awarded if submitted by 11:59PM on Sunday April 17, 2022.

Assignment 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 **Mimir system** before the project deadline. Do **not** submit the `cards.py` file.

Assignment Background

Streets and Alleys is a solitaire card game which is played by one person with a standard 52-card deck of cards. The rules and a tutorial video are available at:

http://worldofsolitaire.com/

Click on "Choose Game" at the top of the screen and click on "Streets and Alleys".

Your program will allow the user to play a simplified version of Streets and Alleys, with the program managing the game. The game rules are given below, but you should play the game online to understand it before you try to code it.

Game Rules

1. Start: The game is played with one standard deck of 52 cards. The deck is shuffled and becomes the initial *stock* (pile of cards).

All fifty-two cards are then dealt from the stock to the *tableau*, left to right, starting in the upper left. The tableau has four rows of two piles each numbered 0-7. Each row has two piles of seven cards on the left (even indexed piles 0,2,4,6) and six cards on the right (odd indexed piles 1,3,5,7). Working left-to-right from the top left corner, deal seven cards to the first pile on the left and then six cards to the pile on the right, left-to-right, in the first row and then working left-to-right fill up the remaining rows. The order of the initial deal is important to match tests. All cards are visible, a.k.a. face up.

The game also has a *foundation*, which has four piles of cards with cards face up. The foundation starts out empty and is filled during play. Each foundation pile is in order and contains only one suit starting with the ace at the bottom and ending with the king on top. Only the top card of each foundation pile is visible. The foundation is placed in the middle between the even and odd indexed piles tableau.

Note the initial layout on the worldofsolitaire.com site.

2. Goal: The game is won when the foundation is full, i.e. all cards are in the foundation.

3. Moves: A player can move one card at a time.
A tableau card can be moved to one of two places:

- To the tableau:
  - A card, the *source card*, can be moved to a collection if the rightmost card in the collection, the *destination card*, is one larger in rank than the source card. Suits do not matter—they may be the same or different. Also, any card can be moved to an empty spot.
- To the foundation:
  - A card, the source card, can be moved to a foundation, if the destination (top) foundation card is the same suit and has a rank one lower than the source card. An ace can only move to an empty foundation pile; no other rank card may move to an empty foundation pile.

A foundation card can be moved only:

- To the tableau:
  - A card, the *source card*, can be moved to a collection if the rightmost card in the collection, the *destination card*, is one larger in rank than the source card. Suits do not matter—they may be the same or different. Also, any card can be moved to an empty spot.

No other moves are permitted (except undo).

Assignment Specifications

You will develop a program that allows the user to play Streets and Alleys according to the rules given 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 a sample interaction with a program satisfying the specifications at the end of this document.

1. The program will recognize the following commands (upper or lower case):

```
MTT s d: Move card from Tableau pile s to Tableau pile d.
MTF s d: Move card from Tableau pile s to Foundation d.
MFT s d: Move card from Foundation s to Tableau pile d.
U: Undo the last valid move.
R: Restart the game (after shuffling)
H: Display this menu of choices
```

Q: Quit the game

where **s** and **d** denote pile numbers (standing for **s**ource and **d**estination) where pile numbers are 0 to 7 inclusive.

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. Neither of the data structures representing the tableau or foundation will be altered by an invalid command.

2. The program will use the following function to initialize a game:

initialize() → (tableau, foundation)

That function has no parameters. It creates and initializes the tableau and foundation, and then returns them as a tuple, in that order. This corresponds to rule 1 in the game rules:

- `foundation` is an empty list of four lists, i.e. `[[],[],[],[]]`
- `tableau` is a list of eight lists, even indexed lists have seven cards, the others have six cards.

The deck is shuffled and becomes the initial `stock` (pile of cards). All fifty-two cards are then dealt from the stock as specified above into the `tableau`. The order of dealing is important to match tests.

4. The program will use the following function to display the current state of the game:

display(tableau, foundation) → None

Provided.
The challenge was to create a display function that was dynamic, adjusting the vertical bars setting off the foundation depending on the maximum number of cards in an even-indexed pile. The field width of the formatting needed to be a variable.

5. The program will use the following function to prompt the user to enter an option and return a representation of the option designed to facilitate subsequent processing.

get_option()→ list

That function takes no parameters. It prompts the user for an option and checks that the input supplied by the user is of the form requested in the menu. Valid inputs for options are described in item (bullet) 1 of the specifications. If the input is not of the required form, the function prints an error message `"Error in option:"` followed by the option entered by the user and returns `None`.

Note that input with the incorrect number of arguments, e.g. M 2, or incorrect types, e.g. F D, will return `None.` Remember to ensure that indices are valid. If the indices are not valid, the function should return `None.` Valid indices are integers within the corresponding range (between 0 and 7 inclusive for the tableau, between 0 and 3 inclusive for the foundation).
if the index for the source is not valid an error in source message `"Error in Source."` should be printed and the function should return `None`.
If the index for the destination is not valid an error in destination message `"Error in Destination."` should be printed and the function should return `None`.

The function returns a list as follows:

- **`None`**, if the input is not of the required form and the indices are not valid.
- **`['Mxx', s, d]`**, where **`s`** and **`d`** are **`int`**'s, for moving a card from a source tableau pile to a destination tableau or foundation. Where `Mxx` is in `['MTT','MTF','MFT']`
- **`['U']`**, for undo
- **`['R']`**, for restart
- **`['H']`**, for displaying the menu
- **`['Q']`**, for quit

5. The program will use the following functions to determine if a requested move is valid:

```
valid_tableau_to_tableau(tableau,src,dst) → bool
valid_move_tableau_to_foundation(tableau,foundation,src,dst) → bool
valid_move_foundation_to_tableau(tableau,foundation,src,dst) → bool
```

These similar functions have three or four parameters: the data structure representing the tableau or foundation and one **`int`** indicating source and the other **`int`** for the destination where the source card should be moved. The rules are stated above in the Game Rules. The function will return **`True,`** if the move is valid; and **`False`**, otherwise. Note that this function should only check if the move is valid. No changes to the tableau or the foundation is done.

Some things to consider:

- An empty foundation can only have an ace moved to it.
- A card can only be moved to a tableau card whose rank is one greater than the source card's rank.
- A card can only be moved to a foundation card if the suits agree, and the source's rank is one larger.
- Any card can be moved to an empty tableau spot.
- Remember to ensure that a source card exists.

When checking that the source card exists at the specified pile, e.g. the index may be out of range; `try-except` works well for that but you do not have to use it.

6. The program will use the following functions to move a card within the tableau:

```
move_tableau_to_tableau(tableau,src,dst) → bool
move_tableau_to_foundation(tableau,foundation,src,dst) → bool
move_foundation_to_tableau(tableau,foundation,src,dst) → bool
```

These functions have three or four parameters: the data structure representing the tableau or foundation, the source, and the destination. If the move is valid (determined by calling the corresponding `validate` function), the function will update the data structure and return `True`; otherwise, it will do nothing to it and return `False`. Because a lot of work is done in the corresponding validate function, these functions are easier to implement.

7. The program will use the following function to check if the game has been won:

```
check_for_win( foundation ) → Bool
```

That function checks to see if the foundation is full. It returns **`True`**, if the foundation is full and **`False`**, otherwise. (Hint: you can make your life easier by assuming that your move (and validate) functions were correct, i.e. the foundation was built correctly.) (Challenge: write this function in one line.)

8. Once you write all your functions, it is time to write your **`main`** function:

a) Start by printing the Welcome message (check the string.txt file)
b) Your program should start by initializing the board (the tableau and the foundation).
c) Display the board first (use the `display()` function) and then the menu.
d) Prompt for an option and check the validity of the input.
e) If `'Mxx s d'`, move a card:
   - If a move is successful, check to see if the user won; if so print,
     `"You won!"`
     display the winning board
     restart then print
     `"\n- - - - New Game. - - - -\n"`
     and display the board. Finally display the MENU again. If the user did not win and move was successful just display the board.
   - If the move was a failure, you should print an error message:
   - `"Error in move: {} , {} , {}"` followed by command, the source and destination in this order.

f) If 'U', undo the last valid move. Successive undo's can eventually undo every valid move. Therefore, you have to save all valid moves (until they are undone) since the start of the game. What data structure works best for this? That is, dictionary, set, list, tuple, or string? Note that this operation is fully specified, but the details are left to you. Hint: start by making it work only with the one last, valid move; work on the history of moves after you get one working. If there is no move to undo just print `"No moves to undo."`. Otherwise, print `"Undo:"` followed by the command that you are undoing. Then display the new board after undo.

g) If `'R'`, restart the game by initializing the board (after shuffling). Display the board.
h) If `'H'`, display the menu of choices
i) If `'Q'`, quit the game
j) If none of these options, the program should display an error message.
k) The program should repeat until the user won or quit the game. Display a goodbye message that is `"Thank you for playing."`.

If the user won,

```
print("You won!")
# display the winning game
print("\n- - - - New Game. - - - -\n")
# restart the game
# display the new game
# display the menu
```

Assignment Notes

1. Before you begin to write any code, play with the provided demo program and look over the sample interaction supplied on the project website to be sure you understand the rules of the game and how you will simulate the demo program. The demo program is at http://worldofsolitaire.com/: Click on "Choose Game" at the top of the screen and click on "Streets and Alleys."

2. We provide a module called **`cards.py`** that contains a Card class and a Deck class. Your program must use this module (**`import cards`**). *Do not modify this file!* This is a generic module for any card game. Your program must implement "Streets and Alleys" *without modifying the* **`cards`** *module***.** Do **not** submit the cards.py to Mimir.

3. Laboratory Exercise #10 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. Begin by downloading **`proj10.py`**. Check that it runs. Gradually replace the "stub" code (marked with comments) with your own code. (Delete the stub code.)

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

http://www.cse.msu.edu/~cse231/General/coding.standard.html

Items 1-9 of the Coding Standard will be enforced for this project.

6. Your program may 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.

7. Your program must contain the functions listed above; you may develop additional functions, as appropriate.

8. Hard-coding to pass one or more tests will earn a zero for the whole project.

TEST 1-4
See the files `input`**`X`**`.txt output`**`X`**`.txt`

Grading Rubric

```
Computer Project #10
Scoring Summary

General Requirements:
(3 pts)          Coding Standard 1-9
(descriptive comments, mnemonic identifiers, format, etc…)


Implementations:

(5 pts) initialize
(4 pts) MTT
(5 pts) valid_MTT
(4 pts) MTF
(5 pts) valid_MTF
(4 pts) move_MFT
(4 pts) valid_MFT
(4 pts) get_option (no Mimir test)
(3 pts) check_for_win
(4 pts) Test 1
(4 pts) Test 2 (winning case)
(3 pts) Test 3 (undo)
(3 pts) Test 4 (errors)
```