Project 2: Freecell

Due: 16 October 2019

In this project, you'll use OO principles and Python builtins to design and build a game that implements the Freecell variant of solitaire.[1]

Basics of the game

The game requires a standard deck of 52 unique cards. Both suits and ranks are important. Ace is low only.

There are three groups of piles. The cards in all piles are face-up. These piles are:

- 8 cascade piles: all cards are visible.
- 4 freecell piles: hold at most 1 card each. Initially empty.
- 4 homecell piles: holds cards of a given suit in order by rank from lowest to highest. Initially empty.

A game begins with all cards being dealt from a randomly-shuffled deck to the 8 cascade piles in round-robin fashion until there are no more cards. The first four cascade piles should have 7 cards each; the remaining four piles should have 6 cards each. The objective of the game is to move all the cards from the cascade piles to the homecell piles, using the freecells as scratch space.

Game play

Actions in the game consist of moving single cards (or, in Full Freecell, groups of cards) from one pile to another. Each kind of pile has slightly different rules for when a card can legally be taken from it and when a card can be placed there; in all cases, though, the card(s) that are moved come from the top of one pile and are placed on the top of another.

[1] Adapted from an assignment by Robert Noonan at William and Mary.

Freecell pile. A freecell pile holds at most 1 card.

Taking Rule: Any card can be taken from the top of a freecell pile.
Placing Rule: Any card can be placed onto an empty pile.

Homecell pile. A homecell pile holds cards of the same suit, in increasing order by rank.

Taking Rule: A card cannot be taken from a homecell pile.
Placing Rule: Only an Ace can be placed onto an empty homecell pile. The suit of the Ace determines the suit of the homecell pile. A card can be placed onto a nonempty homecell pile provided the card is the same suit and the next higher rank than the top card of the homecell pile.

Cascade pile. In a cascade pile, all cards are visible.

Taking Rule: Any card can be taken from the top of a cascade pile.
Placing Rule: Any card may be placed onto an empty cascade pile. Cards may be placed onto the top of a nonempty cascade pile if they form a *tableau* with the current top card in the pile: a tableau is a sequence of two or more cards where the cards are in decreasing order with no gaps, and of alternating colours. Exact suit is irrelevant.
Basic/Full Freecell: The only difference between Basic Freecell and Full Freecell is that in Basic all moves involve single cards, while in Full, multiple cards can be taken and placed in a single action, iff the entire card group that is moved forms a single tableau.

The game terminates when either all cards have been successfully moved to the homecell piles (a win) or the player quits (a loss).

Dealing the cards

For a variety of reasons, it will be convenient if we all generate our decks in the same way, at least when we explicitly seed our random-number generators. For this reason, I've provided a `deck` module in the `/home/shared/280` directory; use it, and don't modify it. Your `Card` class will need to have a constructor that takes two `int` arguments in the ranges $[1, 13]$ and $[0, 3]$ respectively, representing rank and suit (but once you get them into the constructor you can convert them into a more convenient format if you like). Then if you call

```
shuffled = deck.shuffle_deck(Card, seed)
```

where `seed` is some numeric value, you can either

```
for card in shuffled:
```

to iterate through a whole deck, or make 52 explicit calls to `next(deck)`, in order to deal out the cards as shuffled according to that seed. If you use the same seed again, you'll get exactly the same sequence. If you omit the seed (or use `None`) it'll give you a different shuffle each time you run it.

Expected input and output

I should be able to start the program by typing

```
python3 freecell.py 234897
```

or any other number, and get a deck that is seeded by that number.

While the program is running, it should print out a summary of the contents of the board (all the different kinds of piles), and then accept input from the user indicating where to take a card from and where to move it to. If you do the Full Freecell version, the user needs to be able to specify not just where the cards come from but how many to move.

The user should be able to quit without Ctrl-C'ing the program, and if the user wins, the program should end gracefully.

Design homework

For your initial pass at this project, *by Friday* you should come to class with some plans for how you'll design this. It should be on paper (so we can talk about it), and should include at least:

- Draft layout of how this will look on the (text) screen
- Examples of legal and illegal moves
- Ideas for what classes will be required, what data each one will hold, and what behaviours each will support

Checkpoint

For the checkpoint, due next Wednesday (9 Oct) at 4pm, the user should be able to run the program with a specific seed (as specified above), and the program should then print out at least the first eight cards dealt from the deck generated by that seed. (These eight cards will eventually be the bottom/first card in the eight cascade piles, so if you are further in your implementation and have dealt out the full board, that's fine too.)

Each card should display using its standard two-character abbreviation: A for ace, TJQK for ten through king, or a digit for the numbered cards; and the suit's initial. So, 7H is the seven of hearts, TC is the ten of clubs, and so on.

Final version

In a full-credit final version, a user will be able to play a complete game of Freecell. Your documentation should include evidence that will demonstrate to me that it accomplishes what you claim it does; for the movement restrictions this will almost certainly involve test cases, and for the game-play stuff you'll certainly need to use the explicit random seeds to help you show me what to type for your demonstration to work.

Credit is on the following scale:

0: Doesn't compile, or compiles but immediately crashes when it's run.

10: Compiles, runs, does no more than specified for the checkpoint.

20: Deals a complete board and displays representations of all 16 piles and their full contents.

25: As for 20, plus user can specify moves from/to any pile, and program executes the move (at least when the move is permitted, and maybe even if it isn't). If the "from" is an empty pile, responds gracefully without crashing.

35: As for 25, plus program implements at least two of the card movement restrictions.

45: As for 25, plus program implements all five of the card movement restrictions.

50: Complete implementation of Basic Freecell (including win detection).

55: Complete implementation of Full Freecell (including win detection).

+10: Program is otherwise at the 35-point level or above and implements the piles and their move validation using good object-oriented design and appropriate use of builtin collection types. (Note that this rubric item is required for a full-credit solution!)

Again, to get the points you have to not just tell me but *show* me that you've done those things. Document your work and tell me what to look for! Again, I don't plan to debug your code or read it in much detail, although I will look at it to verify the +10 rubric item.

Handing in

Hand it in as `proj2` using the handin script. The checkpoint is due 9 October at 4pm; the final version is due 16 October at 4pm.