Python Project BlackJack

Assignment Overview

This assignment will give you more experience on the use of classes.

Background

You will implement the card game
http://www.solitairenetwork.com/Solitaire/blackjack-square-solitaire-game.html
Familiarize yourself with the game by playing the online version before considering programming. Our goal is to enforce the rules and, after playing, score the game. We do not write a player program to play the game.

The game combines solitaire with blackjack: it is a one-person game, but scoring comes from blackjack. In blackjack a hand's score should stay at or below a value of 21. In this game, blackjack hands are scored from nine hands formed by each of the four rows and five columns of the grid of cards laid out. To play the game you draw cards one at a time from the deck (called the 'stock') and place them on the grid. Once placed, a card cannot be moved. The four discard spots allow one to ignore four cards by placing them in the discard spots rather than on the grid. Once all sixteen spots in the grid have cards, a score is calculated.

Each hand (row or column of the grid) is scored as shown in the table below. Face cards count as 10. Aces can count as either 1 or 11.

Hand	Points	Explanation
Blackjack	10	Blackjack is two cards that total 21
21	7	3, 4, or 5 cards total 21
20	5	Hand totals 20
19	4	Hand totals 19
18	3	Hand totals 18
17	2	Hand totals 17
16 and other	1	Hand totals 16 or less
BUST	0	Hand totals 22 or more

Requirements

Implement the blackjack-square solitaire game in Python with the following requirements.

1. Your game will play one hand and then exit. That is, the sixteen spots on the grid will be filled and a score calculated; then the game will quit.
2. The game begins by dealing a card from the deck onto the playing surface face up. The sixteen grid spots and four discard spots are numbered 1-20. You will repeatedly prompt for a number that indicates where to place the dealt card. Here is an example layout for the game (we're playing in text, remember)

```
Choose location (1 - 20) to move card to (q to quit): 1
Deal:   4S
Tableau:
  9H    2    3    4    5
   6    7    8    9   10
       11   12   13
       14   15   16

Discards:    17   18   19   20
```

The game shows the 4 rows and 5 columns, showing either the position (if the position is unfilled) or the card at that position. Shown also are the Deal, the Tableau and the Discards.

3. The blackjack hands are in the following grid spots
    a. Row hands: [1,2,3,4,5], [6,7,8,9,10], [11,12,13], [14,15,16]
    b. Column hands: [1,6], [2,7,11,14], [3,8,12,15], [4,9,13,16], [5,10]
4. After each card is placed a new card is automatically dealt.
5. When the sixteenth card is placed in the grid, the game ends and the score is calculated.
6. Error checking:
    a. If the user enters a non-number or an out-of-range number, you will print an error message and repeat the prompt.
    b. If the user enters the number of a spot that already has a card, you will print an error message and repeat the prompt.

Card and Deck Classes plus Display function

We provide a module named ***cards*** that contains a card class, a deck class and a display function. We also provide a sample piece of code that demonstrates how to use the cards module. The card and deck classes are general purpose for developing card games so they contain many methods that may not be used in any particular implementation. You are welcome to use all of them, but do not be surprised if there are many that you do not need for this project. The most useful methods are:

- the *Card()* and *Deck()* constructers
- the *get_value()* method of Card
- the *shuffle()* and *deal()* methods of Deck

The *get_value()* method returns the value of the card: 1 for ace, 10 for face cards, and face value for the rest.

Other good information

Notes:

1. When using class methods remember the parenthesis—no error is generated for missing parenthesis, but results will not be what you expect.
2. There are two parts to this game: one is the playing of the game; the other is the scoring of the game. The scoring is the harder of the two. The two parts can be implemented completely separately from each other (divide-and-conquer!).
3. For playing the game, begin by assuming perfect input. Get that working and add error checking later.

4. The hardest part of scoring is the handling of Aces. For blackjack scoring an ace can have a value of 1 or 11. The *get_value()* method in the card class only returns a value of 1 for an ace. You must handle a value of 11 as a special case in your program. The best strategy is to assume that an ace has a value of 11, and if that value will 'bust' you (generate a score greater than 21), change its value to 1. That sounds easy, but what if there are two aces in the hand (or three, or four!)? Also, an ace may be the first card you count, and it is a later card that puts the score over 21 (bust!). For example, an ace, a jack, and a ten would be a 'bust', if the ace has a value of 11, but it is a perfect 21, if the ace has a value of 1.

   As with most things, begin by assuming that an ace always has a value of 1 and once your scoring is working add in the complication of the 11 value.

5. It is useful to have a function that scores a hand (row or column in the grid). Since a hand can have 2, 3, 4 or 5 cards it is useful to pass a list of cards as a parameter. The function can work using the list, extracting the value of the cards, and then calculating the score according to the scoring table. As mentioned above, get this function working using aces with a fixed value (1 or 11) and then adding the complication of differing values for ace after the rest is working.
6. It is also useful to have a function that scores the entire tableau, calling the score hand function 9 times for each of the hands
7. When testing the scoring function it is useful to create specific hands to test out various scoring possibilities. Since the Card class takes parameters when creating a new card that feature is useful for testing. For example, `card1 = Card(8,'S')` will create an eight-of-spades card. You can make a few cards, put them in a list, and call your hand scoring function (item #5). Also, after you run your program you can test the various possibilities within the Python shell.
8. The game display function might be the most difficult functions to write. You have a decision to make as to the data structures used (probably lists) and what they contain as elements. The function must display either the card value (if there is a card there) or the position if no card is yet there. Setting this up might take some work and should probably be left for later in development.