Spring, 2015 CIT 590

Programming Languages and Techniques

Homework 7

First, I have to acknowledge this assignment is only a slightly modified version of one of the assignments given out at Michigan State University. While there are probably solutions floating around on the net, note that using those solutions is risky on 2 counts. a) it counts as cheating and b) you will probably get this assignment wrong, since I have made changes here and there. Also, they do not have mandatory TDD. I do!

This assignment uses classes, dictionaries, lists and a tiny element of files as well. There are lot of interesting cases and hence it should hopefully encourage TDD. It will be the final major Python assignment in this course. We will have one more HW, but that will be something more along the lines of a short exercise.

You get 2 weeks to solve this assignment. The same rules apply as HW5. Let me know about your potential pairing by 6pm on Sunday.

Goal of the assignment

The goal in this assignment is to get you to create a simplistic version of the following online game.

You will implement the card game

blackjack solitaire - http://www.solitairenetwork.com/Solitaire/blackjack-square-solitaire-game.html.

Familiarize yourself with the game by playing the online version before considering programming. Especially, make sure you know how to score. Our goal is to enforce the rules, allow the user to play 1 round of the game and then score the game.

The game is a solo game, so in that sense it is like solitaire, but all of the scoring comes from blackjack. In blackjack a hands 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 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.

How to score Blackjack Solitaire

Every card has a value. If it is from 2 to 10, the value is the number associated with the card. If it is a face card, that is, if you have king, jack or queen, then it is worth 10 points.

Finally, the trickiest card to score is the ace, which counts for 11 or 1, depending upon what gives you a higher score.

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 others	1	Hand totals 16 or less
BUST	0	Hand totals 22 or more

Provided classes

You have been provided with a cards.py file which has both a Cards class and a Deck class.

You will use those classes in your Blakckjack game.

The first step though is to complete those classes. You will notice that there are some very deliberate errors in both those classes.

You will find those classes useful and therefore we strongly recommend first trying to complete the code in cards.py.

Requirements

Once you have played the game for a bit and understand the way it works, here are the 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 shuffling the deck and dealing a card. 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.

3. At any given point, you have a concept of the state of the game, which is represented by one variable called **table**.

   The datastructure we will use is a dictionary where the keys are 'row1', 'row2', 'row3', 'row4'. The values associated with they keys are lists. Initially it looks like row1:[1,2,3,4,5], row2:[6,7,8,9,10], row3:[11,12,13], row4: [14,15,16].

   After the cards are placed it looks something like 'row1':[9C, 2, 3, 4, 5], 'row2':[10S, 9H, JS, 9, 10], 'row3':[11,12,13], 'row4':[2C, 4C, 10D]. That is to say, a number represents

a blank spot and a card number and a suit represents that particular spot being used by that particular card.

The 10S is the actual 10 of spades card object and not the string 10S. But when you display it you will use the string representation. Remember the __str__ function.

Aside from the table variable we also have the discarded cards. Those are just represented by a list of 4. Initially that list has the values [17,18,19,20]. When a user discards a card, it goes into this list. So let's say I discard the Queen of Hearts then the list looks like [QH, 18,19, 20]. Note that we again are not particular about case. We would like all 4 combinations 'qh, 'QH', 'qH' and finally 'Qh' to all represent the queen of hearts.

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:

   - If the user enters a non-number or an out-of-range number, you will print an error message and repeat the prompt.
   - If the user enters the number of a spot that already has a card, you will print an error message and repeat the prompt
   - Also, we would like to keep track of the highest score in the game. For that we will do something very very simple, where we just have a simple file called highScore.txt. Every time the game is played, at the end of the game you compare the score with the score in highScore.txt. If the score exceeds the score that is written in the file, then you print a nice congratulatory message to the user.

   To build your game, you will make one class called BlackJack inside a file called SoloBlackJack.py

   That class needs to have the dictionary called table as discussed above.

   It also needs to have a list variable which is the discardList.

   If you need any other properties for your class, make sure to put them in the __init__ function.

Writing the main playing method

The big method in this class is a method that we will call the **play** method. The goal is to write code such that at the end we can say the following two lines

```
bj_solitaire = Blackjack()
bj_solitaire.play()
```

which has to do a bunch of things. Let us split it up a bit

(a) Display initial state of the game.

(b) Shuffle deck.

(c) Deal a card.

(d) Allow user to make a move. How does the user make a move? We prompt the user to supply a position where they want to put the card and then we take that information and move the dealt card to the appropriate position. When we allow the user to make a move we also want to error check the move. Hint, do the error checking in a separate function/functions.

(e) Display current state of the game

(f) Repeat above 3 steps (deal card, place card, display game) over and over until the game is complete.

(g) At the point where all 16 points of the tableau are filled, print a message telling the user you are going to score the hands and then pass the state of the table to a scoring function.

(h) The final message to display is just the score of your table.

(i) If the current score is the highest score seen so far, then just print a nice congratulatory message.

(j) Print a message saying the game is done and give the option to the user to restart the game from the very beginning.

This assignment gives you freedom as far as how you want to write your functions. But remember things like modularity and allowing for sufficient testing.

For instance the scoring of the game probably should be broken up into scoring of a hand which scores just one hand. And a single hand can always be represented as a list of cards.

Hints

This next section consists of a series of hints. You are not supposed to view them as 'I will lose points if I do not do this ..' These exist merely to potentially help you. If you have a different idea, implement it that way.

(a) 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. That would seem like I am saying that you work on one while your partner should do the other. That might work, but frankly the scoring is what I found interesting when I took a crack at this, so I wouldn't want any of my students to miss out on the fun. And if you do TDD, you will see the beauty of it when you do your scoring function.

(b) For playing the game, begin by assuming perfect input. Get that working and add error checking later. Also, separate out your error checking functions. For instance write a function that says acceptUserMove. Within that function, call another function, checkErrorMove. Then call another function called processUserMove which actually then assumes that you have perfect input.

(c) 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 should only return a value of 1 for an ace, since that is the normal scoring of an ace. Think about how to deal with the 11.

Note that you are not allowed to touch card.py. You are allowed to do whatever you want in SoloBlackjack.py

(d) 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.

(e) It is also useful to have a function that scores the entire table, calling the score hand function 9 times for each of the hands.

(f) Your unit testing code will become more involved in this program. Remember that your unit tests are allowed to be set up before they assert anything. To clarify what I mean, consider this

```
row1 = [Card('K', 'D'), Card(7, 'h'), Card(2, 'd'), Card(6, 's'), Card('J', 'h')]
row2 = [Card('J', 'c'), Card(9, 's'), Card('Q','h'), Card(4, 'c'), Card(10,'d')]
row3 = [Card(10,'s'), Card(5, 's'), Card(6, 'c')]
row4 = [Card(4, 's'), Card('K', 's'), Card(5, 'c')]
table = {'row1': row1, 'row2': row2, 'row3': row3, 'row4': row4}
self.assertEqual(28, scoreGame([row1,row2,row3, row4]))
```

notice that I have to set up the tableau that I want to score. Sure, I could do all of this in one line, but then I increase my chances of getting it wrong. I strongly encourage creating unit tests before you write the scoring function.

(g) The game display function needs to display the entire dictionary, the discard lists and the current card that has been dealt. This function will be called after every move made by the user to display the current state of the game.

Style

The usual rules around docStrings, good variable names etc continue to apply. But the major difference from previous assignments is that you now just have some very basic specifications. You have to figure out what functions will be needed and you will correspondingly write unit tests.

Grading

You will be evaluated in the following manner

- Cards.py completion - 3 points
- Writing unit tests for Cards.py - 3 points
- Designing the SolBlackjack class, picking out valid functions - 12 points
- Unit testing SolBlackjack - 8 points. There are lots of cases for the scoring function in particular.
- Overall user experience when playing your game - 4 points.

The overall score is out of 30 this time because this is a longer assignment.

Please note that in this assignment it is near impossible for us to answer questions like 'will I lose points for ...' We would be happy to take a look at your code and make suggestions. But at the end of the day you will lose

What to submit

The final files to turn in are SolBlackjack.py, SolBlackjackTests.py, Cards.py and CardsTest.py.

So in SolBlackjack.py your first statement must say import cards
or from Cards import *.

Once you do this you can use the Cards and Deck classes in SolBlackjack.py