CS 106 Project 11: Monte Carlo Simulation of Five-card Stud

Dr. Victor Norman
Fall 2013

For this project, you will compute the probabilities of being dealt various significant hands in Five-Card Stud. Five-Card Stud is a variety of poker where you are dealt 5 cards only and you are not allowed to change any cards -- you bet on what you are dealt. We are not going to simulate the betting process. We are just going to do some pattern matching on the hand we are dealt to see if it matches the various significant "named" patterns in poker. These patterns are (listed in order of best hand to worst hand):

- royal flush (or royal straight flush): 10, J, Q, K, A of one suit.
- straight flush: 5 consecutive cards of one suit.
- four of a kind: (all) 4 cards of the same denomination
- full house: 3 of one denomination, and 2 of another.
- flush: all 5 cards have the same suit.
- straight: 5 cards with consecutively increasing denominations (but not all the same suit)
- three of a kind: 3 cards of one denomination
- two pair: 2 cards of one denomination and 2 cards of another denomination
- pair: 2 cards of one denomination.
- high card: (everything else)

The code should create a shuffled deck, and then deal 5 cards ("a hand") to the player, and then check the hand to see if it matches one of these patterns. The code increments a count of what pattern it matches. (Note that a hand can match multiple patterns -- e.g., a full house is always a three of a kind. We only want to match the hand once, to the highest matching pattern (so, we only want to count the hand as a full house in this example).)

Your code should simulate running five-card stud 10,000 times (by default) and count how many of each pattern has been matched. Then, the code must print out the probability of being dealt each of the patterns listed above. E.g., my (very pretty) output looks like this (after dealing one million hands):

Royal flushes: 2 (0.000%)
Straight flushes: 17 (0.002%)
4 of a kinds: 112 (0.011%)
Full houses: 1519 (0.152%)
Flushes: 1953 (0.195%)
Straights: 3556 (0.356%)
3 of a kinds: 21400 (2.140%)
2 pairs: 47536 (4.754%)
Pairs: 422829 (42.283%)
Nothings: 501076 (50.108%)
Total hands: 1000000

Implement your simulation by creating a new class called `Hand` (in `hand.py`) that has one attribute -- a list of cards (i.e., card instances). One needs to be able to add a card to the hand one-by-one, but the cards in the hand should always be sorted (by denomination). If the cards are sorted, then it becomes much easier to do the matching.

Make methods in `Hand` that check to see if the cards in the hand match each of the patterns. E.g., you should have this method:

```python
def is_royal_flush(self) -> boolean
```

Reuse as much code as possible. E.g., the implementation of `is_straight_flush()` should just call `is_straight()` and `is_flush()`. Note that one complicating factor is that an Ace has value 14, but must also be considered to have the value 1 for certain hands (like straights).

Each of your class files must have unit tests in them.

Next, make a new file called `pokercalc.py` that imports `deck`, `card`, and `hand`, and `OptionParser`. `pokercalc.py` should handle a `-v` command-line option and a `-n` (or `--num_hands`) command-line option. The latter allows the user to specify how many hands to deal (with a default value of 10000).

Here is the pseudo-code for `pokercalc.py`:

Create a variable `verbose`, set to `True`.
Ask the user how many hands should be dealt in the simulation. Store the result in `num_hand`.

Initialize variables to keep track of how many of each pattern (pair, flush, etc.) we see. All variables are initialized to 0.

In the following code, add many print statements to help with debugging. "Wrap" them all in `if verbose:` clauses.

A for loop which loops `num_hands / 10` times:
```
create a deck (that is full and shuffled)
a for loop, to be done 10 times (because we can deal out 10 hands of 5 cards each):
create hand and deal 5 cards to it (from the deck)
check if the hand has a royal flush. If so, increment the number of royal flushes seen.
```
else check if the hand is a straight flush, and if so, increment the count of straight flushes seen.
else ...

print out stats for each type of hand seen, as above.

Rubric:

<table>
<thead>
<tr>
<th>Category</th>
<th>Max points</th>
<th>Your points</th>
</tr>
</thead>
<tbody>
<tr>
<td>hand.py and pokercalc.py are clean and neat and follow coding standards (correct file, class, method, and variable names, docstrings, etc.).</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>Judicious use of if verbose: clauses to assist debugging.</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>Unit tests implemented for hand.py</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>Correct output</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td><strong>TOTAL</strong></td>
<td><strong>20</strong></td>
<td></td>
</tr>
</tbody>
</table>

Submit your hand.py, pokercalc.py, deck.py and card.py by copying them to /home/cs/106/current/<yourid>/proj11/.