CSE 101: Introduction to Computational and Algorithmic Thinking

Stony Brook University

Homework Assignment #5

Spring 2018

Assignment Due: April 27, 2018 by 11:59 pm

Assignment Objectives

This homework assignment will give you additional practice classes and objects as you implement the child's card game known as *War*, along with two variants of the game detailed below. Review the simple rules at the following website if you are unfamiliar with the game: https://en.wikipedia.org/wiki/War_(card_game). Note that Aces have will have the highest rank in our versions of the game.

Getting Started

Visit Piazza and download the "bare bones" file `homework5.py` onto your computer, as well as `homework5_driver.py`. Open `homework5.py` in PyCharm and fill in the following information at the top:

1. your first and last name as they appear in Blackboard
2. your Net ID (e.g., jsmith)
3. your Stony Brook ID # (e.g., 111999999)
4. the course number (CSE 101)
5. the assignment name and number (Homework #5)

Do not, under any circumstances, change the names of the functions or their argument lists. The grading software will be looking for exactly those functions provided in `homework5.py`.

Submit your final `homework5.py` file to Blackboard by the due date and time. Late work will not be graded.

Code that crashes and cannot be graded will earn no credit. It is your responsibility to test your code by running it through `homework5.py` and by creating your own test cases.

Preliminaries

Throughout this assignment you will be using the following `Card` and `Player` classes, which are available in the file `war_classes.py`:

```
class Card:
    def __init__(self, rank, suit):
        self._rank = rank
        self._suit = suit

class Player:
```

```
def __init__(self, player_num, score, cards):
    self._player_num = player_num
    self._score = score
    self._cards = cards
```

We see that a `Card` object is defined by its rank and suit attributes, and a `Player` object is defined by its ID number, score and card list attributes.

Part I: Create Deck of Cards (10 points)

Write a function `create_deck()` that creates and returns a `list` of 52 `Card` objects in the order specified below. The function takes no arguments. A `Card` has a rank in the range 0 through 12 and a suite in the range 0 through 3. A card showing a 2 is represented by ID #0, 3 is represented by ID #1, ..., 10 is represented by ID #8; and the Jack, Queen, King and Ace are represented by 9, 10, 11 and 12, respectively:

Rank	Number Assigned
2	0
3	1
4	2
5	3
6	4
7	5
8	6
9	7
10	8
J	9
Q	10
K	11
A	12

Suits are assigned ID numbers as follows:

Suit	Number Assigned
`CLUBS`	0
`SPADES`	1
`DIAMONDS`	2
`HEARTS`	3

For example, a `Card` that represents the Queen of Clubs will have 0 for its `_suit` attribute and 10 for its `_rank` attribute.

The `create_deck()` constructs and returns a list of 52 `Card` objects. The cards are returned in the following order:

- all of the Clubs in the order 2 through Ace, followed by
- all of the Spades in the order 2 through Ace, followed by

- all of the Diamonds in the order 2 through Ace, followed by
- all of the Hearts in the order 2 through Ace.

Part II: Deal the Cards (10 points)

Complete the function `deal_cards()`, which takes one argument, `deck`, is a list `Cards` objects (but not necessarily exactly 52 cards). The objects in `deck` might appear in any random (shuffled) order. The function creates and returns (as a tuple) two `Player` objects with the following values:

- `_player_num`: `1` or `2`, as appropriate
- `_score`: `0`
- `_cards`: the empty list

The function appends alternating `Card` objects from `deck` to the `_cards` attribute of each `Player` object. More specifically,

1. Player 1 receives the `Card` object at `deck[0]`.
2. Player 2 receives the `Card` object at `deck[1]`.
3. Player 1 receives the `Card` object at `deck[2]`.
4. Player 2 receives the `Card` object at `deck[3]`.

and so on, for all the cards in `deck`.

Part III: Play One Round (20 points)

Complete the function `play_normal_round()`, which takes twos arguments, in this order:

1. `player1`: a `Player` object that represents Player #1
2. `player2`: a `Player` object that represents Player #2

Each `Player` object has equal number of `Card` objects in its `_cards` attribute. The function draws cards from each player's hand (use the `draw_card()` method in the `Player` class) until a player wins the round or the players run out of cards. Here is an example of how you might have Player #1 draw a card:

```
player1_card = player1.draw_card()
```

If, on the first draw, the rank of Player #1's card is greater than the rank of Player #2's card, then Player #1 wins the round and the function returns the tuple `(1, 2)`, where 1 indicates Player #1 and 2 indicates the points won by the player. The function also adds 2 to `player1._score`. As in the real *War* card game, Player #1 earns 2 points because he "wins" two cards: his own, as well as Player #2's card. The cards are discarded and are not added to either player's card list. Similarly, if the rank of Player #2's card is greater than the rank of Player #1's card, then Player #2 wins the round and the function returns the tuple `(2, 2)`. The function also adds 2 to `player2._score`. Here is an example:

```
Player 1 drew A♣
Player 2 drew 8♥
Player 1 won the round, scoring 2 points.
```

If the two cards have the same rank, then we have a war! The function draws another card from each player's hand (if any). Cards are continually drawn until the players run out of cards or one player wins the war by drawing a card of higher rank than the other player. Each draw of the cards earns the eventual winner 2 additional points. Here is an example:

```
Player 1 drew 10♥
Player 2 drew 10♦
WAR!
Player 1 drew 2♣
Player 2 drew 2♦
WAR!
Player 1 drew 5♣
Player 2 drew Q♥
Player 2 won the round, scoring 6 points.
```

For the above example, the function would return `(2, 6)` and would update the value of `player2._score` accordingly.

If the two players start a war and then run out of cards, the round is considered a tie. In this case the function returns the tuple `(0, 0)`.

Part IV: Determine the Winner (10 points)

Complete the function `check_game_winner()`, which takes the following arguments, in this order:

1. `player1`: a `Player` object that represents Player #1
2. `player2`: a `Player` object that represents Player #2

The function returns `1` if Player #1 has the higher score, `2` if Player #2 has the higher score, or `0` if the two players have the same score.

Part V: War Variant #1: Suit Rank (20 points)

Complete the function `play_with_suits()`, which takes two arguments, in this order:

1. `player1`: a `Player` object that represents Player #1
2. `player2`: a `Player` object that represents Player #2

This function provides an alternate form of gameplay to the rules implemented in the `play_normal_round()` function. Rather than decide the winner using card ranks, the winner is decided using suits:

1. Hearts beat Spades and Diamonds

2. Spades beat Diamonds and Clubs
3. Diamonds beat Clubs
4. Clubs beat Hearts

Here's an example:

```
Player 1 drew 6♣
Player 2 drew 2♠
Player 2 won the round, scoring 2 points.
```

Wars are now caused when two cards of the same suit are drawn, as in the example below:

```
Player 1 drew Q♣
Player 2 drew 8♣
WAR!
Player 1 drew K♣
Player 2 drew 9♣
WAR!
Player 1 drew A♠
Player 2 drew 8♦
Player 1 won the round, scoring 6 points.
```

If the two players start a war and then run out of cards, the round is considered a tie. In this case the function returns the tuple `(0, 0)`.

Hint: recall that each suit is represented using an ID number (see the `suits` dictionary in the `war_classes.py` file). You can these numbers to compare the `_suit` properties of two `Card` objects.

Part VI: War Variant #2: Scouting (30 points)

Complete the function `play_with_scouts()`, which takes two arguments, in this order:

1. `player1`: a `Player` object that represents Player #1
2. `player2`: a `Player` object that represents Player #2

This function provides an alternate form of gameplay to the rules implemented in the `play_normal_round()` function. Play proceeds as in normal gameplay except when a player draws a card of rank 5 or lower (i.e., 2, 3, 4 or 5). In such cases, if the player has at least one more card in his hand, the player *scouts* by drawing a second card from his hand. The combined value of the first card played and the scouted card is the value used in the battle (with J = 11, Q = 12, K = 13, A = 14). If the player is unable to draw a second card (because he has run out of cards), then the round simply continues as normal. Here is an example:

```
Player 1 drew 9♣
Player 2 drew 5♥
Player 2 drew K♠
Player 2 won the round, scoring 3 points.
```

Player #1's 9 of Clubs is valued at 9, whereas Player #2's 5 and King together are valued at 18 ($18 = 5 + 13$). Therefore, Player #2 wins the round. Since 3 total cards were draw this round, the winner earns 3 points.

Below's an example where scouting the next card did not help Player #1, and he lost an extra card:

```
Player 1 drew 2♦
Player 1 drew 7♦
Player 2 drew Q♦
Player 2 won the round, scoring 3 points.
```

It is possible that scouting a card can cause one or more Wars to break out. The Wars themselves might cause additional scouting to happen:

```
Player 1 drew Q♥
Player 2 drew 4♥
Player 2 drew 8♥
WAR!
Player 1 drew 7♣
Player 2 drew 3♠
Player 2 drew K♥
Player 2 won the round, scoring 6 points.
```

If the two players start a war and then one (or both) runs out of cards, the round is considered a tie. In this case the function returns the tuple `(0, 0)`.

How to Submit Your Work for Grading

To submit your `.py` file for grading:

1. Login to Blackboard and locate the course account for CSE 101.
2. Click on "Assignments" in the left-hand menu and find the link for this assignment.
3. Click on the link for this assignment.
4. Click the "Browse My Computer" button and locate the `.py` file you wish to submit. Submit only that one `.py` file.
5. Click the "Submit" button to submit your work for grading.

Oops, I messed up and I need to resubmit a file!

No worries! Just follow the above directions again. We will grade only your last submission.