Set War

Let's take our Set cards (from last assignment) and play a different game with them. The game is called Set War and is a variant of the popular game War (except there are no actual wars). The rules are as follows.

- To Start, the cards are dealt to the players one at a time starting with Player 0 until all cards are dealt. (Note that some players may get an extra card. Tough luck.) Players don't look at their cards and they keep them in a pile face down.
- Player 0 goes first, followed by Player 1, etc. and then back to Player 0.
- On a player's turn, they take the top card from their pile and place it on the table. The cards on the table are laid out in a line so it's clear what their order is. These cards are called *the line*.
- If the card is played that forms a set with two other cards in the line, the player who put that last card down wins all the cards in the line. The cards from the line are stacked so they maintain their order (first card played is on top) and they are placed on the bottom of that player's pile.
- The player to win the round is the first to put down in the start of the next round.
- If a player plays their last card, and that card does not form a set (i.e. they don't win the round on that play), then that player is out of the game. If that leaves only one player, then the round is over and the last player wins the pile.
- If there is only one player left, that player is the winner.

An ADT for a deck of cards

Many of the basic operations needed to implement a deck of set cards apply equally well to other card games. So, in writing the code, you will first implement a class `DeckOfCards` which handles decks of cards more generally.

The `DeckOfCards` ADT should support the following operations:

- `__init__(L)` an initializer that takes either a string with space separated cards or a list of strings, where each is a card.
- `dealTop()` removes and returns the card at the top of the deck.
- `dealBottom()` removes and returns the card at the bottom of the deck.
- `addTop(card)` adds `card` to the top of the deck.
- `addBottom(card)` adds `card` to the bottom of the deck.

- `addPileTop(pile)` adds `pile` to the top of the deck. The `pile` should itself be a deck of cards. After this operation, the bottom card of `pile` will be on top of the (formerly) top card of the deck.
- `addPileBottom(pile)` adds `pile` to the bottom of the deck. The `pile` should itself be a deck of cards. After this operation, the top card of `pile` will be below the (formerly) bottom card of the deck.
- `deal(nplayers, ncards)` deal out `ncards` cards to `nplayers` players. If `ncards` is `None` or omitted, then it should deal out all of the cards. It should return a list or a tuple of decks corresponding to the hands of the different players. Dealing is assumed to follow the standard conventions that one card is given at a time, and each new card given is added to the top of the hand that receives it.
- `__len__()` return the number of cards in the deck.

For the specific case of a deck of Set cards, you will implement a subclass of `DeckOfCards`, i.e., a class that inherits from `DeckOfCards` as shown below.

```
class SetDeck(DeckOfCards):
    def __init__(self, cards):
        DeckOfCards.__init__(self, cards)
        # you can add more stuff here.
```

Submitting your DeckOfCards

Your submission should include files called `deckofcards.py` containing your `DeckOfCards` class. It should also include a file called `setdeck.py` that includes your `SetDeck` class. You can include any other files that you want. For example, if you will be using a doubly-linked list, you will want to include that file. However, please don't include test files with your submission. This has a tendency to mess up Mimir's tests.

Implementing Set War

Your submission should contain the following.

- A file called `playsetwar.py`.
- This script should read two strings from stdin. The first is the number of players. The second is the string of cards.
- The game should be played and the output should be a string that says which player won and

how many rounds it took. For example, a possible output string would be

```
Player 0 won in 18 rounds.
```

You can use the following code to print this.

```
print("Player %d won in %d rounds." % (winner, rounds))
```

If there is was only one round, you should not print the \emph{s} in `1 rounds`.

- If there is no winner after 10000 rounds, the script should print the following.

```
Draw after 10000 rounds.
```

- You should include unit tests for all of the methods in the DeckOfCards ADT.
- You should also include unit tests for your SetDeck and game playing code.
- Ideally, all deck operations other than dealing should be constant time ($O(1)$). (*This probably requires a linked list implementation.*)
- Do not use the Python `collections.deque` class. Write your own data structures.