Peeking Blackjack

Now that we have written general-purpose MDP algorithms, let's use them to play (a modified version of) Blackjack. For this problem, you will be creating an MDP to describe a modified version of Blackjack.

For our version of Blackjack, the deck can contain an arbitrary collection of cards with different values, each with a given multiplicity. For example, a standard deck would have card values $[1, 2, \ldots, 13]$ and multiplicity 4. You could also have a deck with card values $[1, 5, 20]$. The deck is shuffled (each permutation of the cards is equally likely).

The game occurs in a sequence of rounds. Each round, the player either (i) takes the next card from the top of the deck (costing nothing), (ii) peeks at the top card (costing `peekCost`, in which case the next round, that card will be drawn), or (iii) quits the game. (Note: it is not possible to peek twice in a row; if the player peeks twice in a row, then `succAndProbReward()` should return `[]`.)

The game continues until one of the following conditions becomes true:

- The player quits, in which case her reward is the sum of the cards in her hand.
- The player takes a card, and this leaves her with a sum that is strictly greater than the threshold, in which case her reward is 0.
- The deck runs out of cards, in which case it is as if she quits, and she gets a reward which is the sum of the cards in her hand.

In this problem, your state s will be represented as a triple:

```
(totalCardValueInHand, nextCardIndexIfPeeked, deckCardCounts)
```

As an example, assume the deck has card values $[1, 2, 3]$ with multiplicity 1, and the threshold is 4. Initially, the player has no cards, so her total is 0; this corresponds to state `(0, None, (1, 1, 1))`. At this point, she can take, peek, or quit.

- If she takes, the three possible successor states (each has 1/3 probability) are

  ```
  (1, None, (0, 1, 1))
  (2, None, (1, 0, 1))
  (3, None, (1, 1, 0))
  ```

  She will receive reward 0 for reaching any of these states.

- If she instead peeks, the three possible successor states are

  ```
  (0, 0, (1, 1, 1))
  (0, 1, (1, 1, 1))
  (0, 2, (1, 1, 1))
  ```

  She will receive reward `-peekCost` to reach these states. From `(0, 0, (1, 1, 1))`, taking yields `(1, None, (0, 1, 1))` deterministically.

- If she quits, then the resulting state will be `(0, None, None)` (note setting the deck to `None` signifies the end of the game).

As another example, let's say her current state is `(3, None, (1, 1, 0))`.

- If she quits, the successor state will be `(3, None, None)`.
- If she takes, the successor states are `(3 + 1, None, (0, 1, 0))` or `(3 + 2, None, None)`. Note that in the second successor state, the deck is set to `None` to signify the game ended with a bust. You should also set the deck to `None` if the deck runs out of cards.