Problem Set 2

Important:

- Write your name as well as your NU ID on your assignment. Please number your problems.
- Submit both results and your code.
- Give complete answers. Do not just give the final answer; instead show steps you went through to get there and explain what you are doing. Do not leave out critical intermediate steps.
- This assignment must be submitted electronically through Gradescope by February 13th 2025 (Thursday) by 11:59 PM.

1 Blackjack

In this problem, we will be playing a modified version of Blackjack. For this problem, you will be creating an MDP to describe states, actions, and rewards in this game. You will need to fill the remaining methods in the file *submission.py.*.

For our version of Blackjack, the deck can contain an arbitrary collection of cards with different face values. At the start of the game, the deck contains the same number of each cards of each face value; we call this number the *multiplicity*. For example, a standard deck of 52 cards would have face values $[1, 2, ..., 13]$ and multiplicity 4. You could also have a deck with face values $[1, 5, 20]$ with multiplicity 10 yielding a total of 30 cards in total: 10 each of 1s, 5s, and 20s). The deck is shuffled, meaning that each permutation of the cards is equally likely.

The game occurs in a sequence of rounds. Each round, the player can either:

- take the next card from the top of the deck, costing nothing.
- peek at the top card, costing *peekCost*. In the next round, that card will be drawn, if the player chooses to draw. It is not possible to peek twice in a row. If the player peeks twice in a row, then the method *succAndProbReward()* should return an empty list [ ].)
- end the game.

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

- The player ends the game, in which case their reward is the sum of the face values of the cards in their hand.
- The player takes a card and "goes bust". This means that the sum of the face values of the cards in their hand is strictly greater than the threshold specified at the start of the game. If this happens, their reward is 0.
- The deck runs out of cards, in which case it is as if they ended the game, and they get a reward which is the sum of the cards in their hand. Make sure that if you take the last card and go bust, then the reward becomes 0 not the sum of values of cards.

In this problem, a state s will be represented as a 3-element tuple:

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

Example

Consider a deck with card values $[1, 2, 3]$, multiplicity 1, and a threshold of 4. Initially, the player has no cards, so their total is 0 corresponding to state

```
(0, None, (1, 1, 1))
```

At this point, they can take a card, peek at the top card, or end the game.

- If they take a card, the three possible successor states, each of which has equal probability of 1/3, are:

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

  Even though the player now has a card in their hand for which they may receive a reward at the end of the game, the reward is not actually granted until the game ends. That is why, the states given above are associated with a reward of 0.

- If the player peeks at the top card, the three possible successor states are:

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

  The player will receive an immediate reward of *-peekCost* for reaching any of these states. Remember that after peeking, if the player chooses to take a card, the peeked card will be chosen. For instance, if the peeked card had a value of 1, the state will change from

  ```
  (0, 0, (1, 1, 1))    to     (1, None, (0, 1, 1))
  ```

  N.B. The second element of the state tuple is not the face value of the card that will be drawn next, but the index into the deck.

- If the player ends the game, then the resulting state will be

  ```
  (0, None, None)
  ```

  Setting the deck to None signifies the end of the game.

1.1 Practice with an Example

To make sure you understand the game, try out the example below and compare your answers to the provided ones. Consider a deck with card values $[1, 4, 7]$, multiplicity 2, and a threshold of 8. Assume the player's current state is

```
(4, None, (2, 1, 2)).
```

1. What are the possible successors states if the player takes a card and the associated reward with each successor? You should set the *deckCardCounts* element of the the tuple to *None* if the game ended with a bust. In the coding part in the next subsections, you should also set *deckCardCounts* to *None* if the deck runs out of cards.
2. What are the possible successors states if the player peeks at the top card and the associated reward with each successor?
3. What is the successor state if the player ends the game and the associated reward?

1.2 BlackjackMDP

1. Implement the general game of Blackjack as an MDP by filling out the *succAndProbReward()* method of the class *BlackjackMDP*.

2. Assume you're running a casino, and you're trying to design a deck to make people peek a lot. The threshold is fixed at 20, and the peek cost is 1. Design a deck where for at least 10% of states, the optimal policy is to peek. Fill out the function *peekingMDP()* to return an instance of *BlackjackMDP* where the optimal action is to peek in at least 10% of states.

   Hint: Before randomly assigning values, think of a case when you really want to peek instead of blindly taking a card.

You will work on part 2 in the next assignment in which you will be using reinforcement learning to play the game, and learn its rules and strategy at the same time.

2 Constraint Satisfactory Problems

Suppose you have n doors that you need to open. Initially, every door i, with $i = 1, ..., n$ is closed. There are m switches which control the doors. Each switch is linked with a subset of the doors. For each switch j, with $j = 1, ..., m$, you have access to the subset $D_j \subset \{1, ..., n\}$ of doors that the switch can open. When a switch j is toggled, the state of every door in D_j changes.

For example, consider that the switch 1 is linked with doors 1 and 2, and switch 2 is associated doors 2 and 3. Toggling switch 1 results in doors 1 and 2 opening. If we toggle switch 2 next, door 3 will open whilst door 2 will close.

In general, if k switches controlling the same door are toggled, then the door will be open if k is odd, and it will be closed if k is even.

Your goal is to open all the doors by pressing a subset of the buttons. Construct a CSP to solve this problem. Your CSP should have m variables and n constraints. You can use constraints that can be functions of up to m variables. Describe your CSP precisely and concisely. You need to specify the variables with their domain, and the constraints with their scope and expression. The constraints will need to include D_j.