Homework 4: Monte Carlo Methods
Due: Monday, October 11th 11:59 pm

The purpose of this project is to study different properties of Monte Carlo methods.

Problem
The objective of the popular casino card game of blackjack is to obtain cards that sum of whose numerical values is as great as possible without exceeding 21. All face cards count as 10, and an ace can count either as 1 or as 11. We consider the version in which each player competes independently against the dealer. The game begins with two cards dealt to both dealer and player. One of the dealer's cards is face up and the other is face down. If the player has 21 immediately (an ace and a 10-card), it is called a natural. He then wins unless the dealer also has a natural, in which case the game is a draw. If the player does not have a natural, then he can request additional cards, one by one (hits), until he either stops (sticks) or exceeds 21 (goes bust). If he goes bust, he loses; if he sticks, then it becomes the dealer's turn. The dealer hits or sticks according to a fixed strategy without choice: he sticks on any sum of 17 or greater, and hits otherwise. If the dealer goes bust, then the player wins; otherwise, the outcome—win, lose, or draw—is determined by whose final sum is closer to 21.

Playing blackjack is naturally formulated as an **episodic** finite MDP. Each game of blackjack is an episode. Rewards of +1, −1, and 0 are given for winning, losing, and drawing, respectively. All rewards within a game are zero, and we do not discount ($\gamma = 1$); therefore these terminal rewards are also the returns. The player's actions are to hit or to stick. The states depend on the player's cards and the dealer's showing card. We assume that cards are dealt from an infinite deck (i.e., with replacement) so that there is no advantage to keeping track of the cards already dealt. If the player holds an ace that he could count as 11 without going bust, then the ace is said to be usable. In this case it is always counted as 11 because counting it as 1 would make the sum 11 or less, in which case there is no decision to be made because, obviously, the player should always hit. Thus, the player makes decisions on the basis of three variables: his current sum (12–21), the dealer's one showing card (ace–10), and whether or not he holds a usable ace. This makes for a total of 200 states.

Note: for this homework you will be provided with an implementation of the Blackjack game and you can import it into your notebook as:

```
from blackjack import BlackJack
```

You do not need to submit the `blackjack.py` file. We recommend you do not make changes to the `blackjack.py` file, because we use our file while running and grading your submission.

Part I
1. Write a python function for the player policy that sticks if the player's sum is 19, 20, or 21, and hits otherwise. The input to this function should be states and the output should be the action. Then use this function to play Blackjack and print out 3 sample trajectories.
2. Implement the First-visit Monte Carlo prediction algorithm and consider the player policy you developed in the previous step. Estimate the state-value function for this policy using the first-visit Monte Carlo prediction algorithm by simulating 500,000 blackjack games

using the policy and average the returns following each state. Plot the 3D graph of state-value (z-axis), dealer showing (x-axis), player sum (y-axis) for both usable ace and not usable ace states (i.e., you should have two plots). Your implementation should use the incremental sample-average.

Part II

1. Use the previous policy (sticks if the player's sum is 19, 20, or 21, and otherwise hits) as the initial policy and find (i) the action-value estimates and (ii) the optimal policy using Monte Carlo with Exploring Starts. To make the starting states with uniform probabilities, pick the dealer's cards, the player's sum, and whether or not the player has a usable ace, according to a uniform sampling process.
2. Plot the 3D graph of state-value (z-axis), dealer showing (x-axis), player sum (y-axis) for both usable ace and not usable ace states (i.e., you should have two plots) for 500,000 simulated games. Also plot the optimal policy for both the usable ace and not usable ace states for all the player's sum and dealer showing (i.e., you should have two plots).

Part III (*)

1. Extend your algorithm to the off-policy Monte Carlo control with weighted importance sampling and estimate action-values and the target policy. You need to consider an arbitrary soft behavior policy. Plot the 3D graph of state-value (z-axis), dealer showing (x-axis), player sum (y-axis) for both usable ace and not usable ace states (i.e. you should have two plots) for 500,000 simulated games. Also plot the optimal policy for both the usable ace and not usable ace states for all the player's sum and dealer showing (i.e. you should have two plots).
2. Answer the following questions:
   a. What behavior policy did you select? Why does it make sense?
   b. How does the obtained target policy in this part compare to the target policies in parts I and II? Is it different? Why?

Evaluation: we will grade your submission according to the following table:

Item	COMP4600	COMP5500
Part I.1	10	5
Part I.2	40	25
Part II.1	30	15
Part II.2	20	15
Part III.1	10 (extra credit)	30
Part III.2	10 (extra credit)	10

Note 1: The parts marked with * are optional for COMP4600 (undergraduates) but mandatory for COMP5500 (graduates). Part III is only for COMP5500 and counts as extra credit for COMP4600.

Note 2: We do not accept code in any programming language other than Python 3 (do not use Python 2).

Note 3: For the implementation of the algorithms, you are not allowed to rely on any python library other than `numpy` and `matplotlib` (for plotting).

Note 4: All the code, plots, explanations should be included in a single Jupyter Notebook (`.ipynb`) file. Include your name as part of the filename and submit through Blackboard.

Submission: By 11:59pm on Monday, October 11th 2021, submit your `student_name.ipynb` files on Blackboard. Make sure everything is entirely contained within this file and it runs without any error.