Roulette

In this project, you will build a Simple Gambling Simulator. Specifically, you will revise the code in the roulette.py file to simulate 1000 successive bets on the outcomes (i.e., spins) of the American roulette wheel using the betting scheme outlined in the pseudo-code below. Each series of 1000 successive bets are called an "episode." You should test for the results of the betting events by making successive calls to the get_spin_result(win_prob) function.

Note that you will have to update the win_prob parameter according to the correct probability of winning which is 18/38

Here is the pseudocode of the strategy:

```
episode_winnings = $0

while episode_winnings < $80:

        won = False

        bet_amount = $1

        while not won

                wager bet_amount on black

                won = result of roulette wheel spin

                if won == True:

                        episode_winnings = episode_winnings + bet_amount

                else:

                        episode_winnings = episode_winnings - bet_amount

                        bet_amount = bet_amount * 2
```

Additional details regarding how roulette betting works: Betting on black (or red) is considered an "even money" bet. That means that if you bet N chips and win, you keep your N chips, and you win another N chips. If you bet N chips and you lose, then those N chips are lost. The odds of winning or losing depend on betting at an American wheel or a European wheel. For this project, we will be assuming an American wheel. You can learn more about roulette and betting here: https://en.wikipedia.org/wiki/Roulette.

Experiment 1

In this experiment, you will develop code that performs experiments using Professor Balch's original betting strategy. The approach we are going to take is called Monte Carlo simulation. The idea is to run a simulator repeatedly with randomized inputs and assess the results in aggregate. Your implementation will produce the following charts (i.e., figures):

- **Figure 1**: Run your simple simulator 10 episodes and track the winnings, starting from 0 each time. Plot all 10 episodes on one chart using Matplotlib functions. The horizontal (X) axis must range from 0 to 300, the vertical (Y) axis must range from –256 to +100. We will not be surprised if some of the plot lines are not visible because they exceed the vertical or horizontal scales.
- **Figure 2**: Run your simple simulator 1000 episodes. (Remember that 1000 successive bets are one episode.) Plot the mean value of winnings for each spin round using the same axis bounds as Figure 1. For example, you should take the mean of the first spin of all 1000 episodes. Add an additional line above and below the mean, at mean plus standard deviation, and mean minus standard deviation of the winnings at each point.
- **Figure 3**: Use the same data you used for Figure 2 but plot the median instead of the mean. Add an additional line above and below the median to represent the median plus standard deviation and median minus standard deviation of the winnings at each point.

For all the above figures and experiments, if the target of $80 winnings is reached, stop betting, and allow the $80 value to persist from spin to spin (e.g., fill the data forward with a value of $80).

All charts must be properly titled, have appropriate axis labels, use consistent axis ranges, and have legends.

Experiment 2

You may have noticed that the original strategy performed in experiment 1 works well, maybe better than you expected. One reason for this is that we were allowing the gambler to use an unlimited bankroll. In this experiment, we retain the upper limit of $80 in winning retained but make things more realistic by giving the gambler a $256 bankroll. This will require a modification to the original strategy since if he or she runs out of money: bzzt, that's it. Repeat the experiments, as above, with this new condition. Note that once the player has lost all their money (i.e., episode_winnings reach -256), stop betting and fill that number (-256) forward. An important corner case to handle is the situation where the next bet should be $N, but you only have $M (where M<N). Since you cannot bet more than you have, be sure you only bet $M. Here are the two charts to create:

- **Figure 4**: Run your realistic simulator 1000 episodes and track the winnings, starting from 0 each time. Plot the mean value of winnings for each spin using the same axis bounds as Figure 1. Add an additional line above and below the mean at mean plus standard deviation and mean minus standard deviation of the winnings at each point.
- **Figure 5**: Use the same data you used for Figure 4 but plot the median instead of the mean. Add an additional line above and below the median to represent the median plus standard deviation and median minus standard deviation of the winnings at each point.

All charts must be properly titled, have appropriate axis labels, use consistent axis ranges, and have legends. You should use python's Matplotlib library.

1. All winnings must be tracked by storing them in a NumPy array. You might call that array winnings where winnings[0] should be set to 0 (just before the first spin). The entry in winnings[1] should reflect the total winnings after the first spin and so on.
2. Use the population standard deviation. The standard deviation is plotted as two lines, the upper standard deviation (e.g., mean +stdev) and the lower standard deviation (e.g., mean –stdev).

3.5.1 Structuring the NumPy Array

		1 Spin [0]	2 Spin [1]	3 Spin [2]	4 Spin[3]	...	1001 Spin [1000]
1	Episode [0]	0	2	2	3	...	50
2	Episode [1]	0	4	5	6	...	30
3	Episode [2]	0	3	5	9	...	112
	Mean	0.0000	3.0000	4.0000	6.0000	...	64.0000
	StDev	0.0000	0.8165	1.4142	2.4495	...	34.9094

Hint: One way to think about structuring the NumPy array for holding winnings is illustrated below. Each episode consists of 1000 spins plus the initial value in the first column.