Reinforcement Learning Project

1. Project Overview

In this project, we will implement two multiarmed bandits (MAB) for the following problems. For each of them, you are given a bandit implementation in Python, and you have to integrate this bandit code we reviewed in class.

The project can be done in pairs.

For each problem add a Python file that implements and runs the MAB for it.

Submission:

you have to upload the two files to the Google form that will be provided. The files will be uploaded as a **single** zip file.

2. Advertisement Optimization Bandit

Filename: run_mab_advertisement.py

The Advertisement Optimization Bandit represents the challenge faced by news websites in deciding which ads to display to their visitors. Each ad placement (bandit) has an unknown probability of being clicked on by a user, based on factors like the ad's content, the user's interests, and the placement's visibility. The goal is to maximize the total number of ad clicks (rewards) by dynamically allocating ads to different placements.

This problem is a manifestation of the multi-armed bandit problem in the context of online advertising. Just as a gambler must decide which slot machines to pull in a casino, a news website must decide which ads to display to its users. The challenge lies in balancing the exploration of new ads (to discover potentially profitable ones) with the exploitation of known profitable ads to maximize revenue.

The Advertisement Optimization Bandit class simulates this decision-making process by generating rewards based on the probability of a user clicking on an ad. This allows for the exploration and exploitation strategies to be tested and optimized, similar to how multi-armed bandit algorithms are used in A/B testing and other optimization problems.

class AdvertisementBandit(Bandit):

```
    def __init__(self, n, probas=None):
        assert probas is None or len(probas) == n
        self.n = n # Number of ad placements
        # Set the bandits' rewards randomly
        if probas is None:
            np.random.seed(int(time.time()))
            self.probas = [np.random.random() for _ in range(self.n)]
        else:
            self.probas = probas
        # Maximal reward
        self.best_proba = max(self.probas)

    def generate_reward(self, i):
        # The player selected the i-th ad placement.
        if np.random.random() < self.probas[i]:
            return 1 # User clicks on the ad
        else:
            return 0 # User does not click on the ad
```

3. Content Personalization Bandit

Filename: run_mab_personalization**.py**

The Content Personalization Bandit represents the challenge faced by streaming platforms in deciding which content to recommend to their users. Each content category (bandit) has an unknown probability of engaging a user, based on factors like the user's past viewing history, the content's genre, and the user's current mood or interests. The goal is to maximize the total

number of content engagements (rewards) by dynamically allocating content recommendations to different categories.

This problem is a manifestation of the multi-armed bandit problem in the context of content personalization. Just as a gambler must decide which slot machines to pull in a casino, a streaming platform must decide which content to recommend to its users. The challenge lies in balancing the exploration of new content categories (to discover potentially engaging ones) with the exploitation of known engaging content categories to maximize user satisfaction.

The Content Personalization Bandit class simulates this decision-making process by generating rewards based on the probability of a user engaging with a content category. This allows for the exploration and exploitation strategies to be tested and optimized, similar to how multi-armed bandit algorithms are used in A/B testing and other optimization problems.

```
class ContentPersonalizationBandit(Bandit):
    def __init__(self, n, probas=None):
        assert probas is None or len(probas) == n
        self.n = n # Number of content categories
        # Set the bandits' rewards randomly
        if probas is None:
            np.random.seed(int(time.time()))
            self.probas = [np.random.random() for _ in range(self.n)]
        else:
            self.probas = probas

        # Maximal reward
        self.best_proba = max(self.probas)

    def generate_reward(self, i):
        # The player selected the i-th content category.
```

```
if np.random.random() < self.probas[i]:
    return 1 # User engages with the content
else:
    return 0 # User does not engage with the content
```