Problem Statement-News Value Maximizer Using K-Arm Bandit

Objective:

Design a recommendation system using the K-arm bandit approach to maximize the views of politically and commercially aligned news articles

Action Space (Arms):

Each arm represents a different article available for recommendation at a given time. The number of arms corresponds to the number of articles in the system (K).

Reward Structure:

The reward is defined as a binary or continuous value based on whether a user clicks on a recommended article. Aligned articles may have a higher weighted reward to encourage their selection.

Goal:

Maximize cumulative rewards by optimizing the selection of articles to be recommended, thereby increasing the views of aligned articles.

In [ ]:

In [1]:
```
import numpy as np
import random
```

In [28]:
```
class KArmBandit:
    def __init__(self, k, epsilon=0.1, aligned_weight=1.5):
        self.k = k
        self.epsilon = epsilon
        self.aligned_weight = aligned_weight
        self.q_values = np.zeros(k)
        self.arm_counts = np.zeros(k)

    def select_arm(self):

        if random.random() < self.epsilon:
            return random.randint(0, self.k - 1)
        else:
            return np.argmax(self.q_values)

    def update(self, arm, reward, aligned):
        if aligned:
            reward *= self.aligned_weight
        self.arm_counts[arm] += 1
        self.q_values[arm] += (reward - self.q_values[arm]) / self.arm
```

In [29]:
```
 def run_bandit(bandit, n_iterations, reward_function):
    rewards = np.zeros(n_iterations)
    for i in range(n_iterations):
        arm = bandit.select_arm()
        aligned = random.choice([True, False])
        reward = reward_function(arm, aligned)
        bandit.update(arm, reward, aligned)
        rewards[i] = reward
    return rewards
```

In [30]:
```
def reward_function(arm, aligned):
    base_reward = np.random.normal(arm + 1, 1)
    return base_reward + (2 if aligned else 0)
```

In [31]:
```
k = 10
epsilon = 0.1
n_iterations = 1000
```

In [32]:
```
bandit = KArmBandit(k=k, epsilon=epsilon)
```

In [33]:
```
rewards = run_bandit(bandit, n_iterations, reward_function)
```

In [34]:
```
print(f"Total Reward: {np.sum(rewards)}")
print(f"Average Reward: {np.mean(rewards)}")
print(f"Estimated Q-values: {bandit.q_values}")
print(f"Number of times each arm was selected: {bandit.arm_counts}")
```

```
Total Reward: 9973.16313075443
Average Reward: 9.97316313075443
Estimated Q-values: [ 2.92052616  4.58586329  5.05110961  6.63511437
6.49466306  9.17296806
  9.94450076  8.99316165  9.17514562 14.07151663]
Number of times each arm was selected: [ 40.   9.  34.  16.  16.
9.  32.   8.   6. 830.]
```