```python
import numpy as np
import matplotlib.pyplot as plt
import random

class ThreeArmedBanditEnv:
    def __init__(self):
        self.probabilities = [0.4, 0.4, 0.4]
        self.n_arms = len(self.probabilities)
        self.best_action = np.argmax(self.probabilities)

    def step(self, action):
        if action < 0 or action >= self.n_arms:
            raise ValueError("Invalid action")
        reward = 1 if np.random.rand() < self.probabilities[action] else 0
        return None, reward, False, False, {}

    def reset(self):
        return None

def run_ucb(env, num_rounds=1000):
    n_arms = env.n_arms
    Q = np.zeros(n_arms)
    N = np.zeros(n_arms)
    rewards = []
    arm_selections = np.zeros(n_arms)

    for t in range(1, num_rounds + 1):
        ucb_values = np.zeros(n_arms)
        for a in range(n_arms):
            if N[a] == 0:
                ucb_values[a] = float('inf')
            else:
                ucb_values[a] = Q[a] + 2 * np.sqrt(np.log(t) / N[a])

        action = np.argmax(ucb_values)
        _, reward, _, _, _ = env.step(action)
        N[action] += 1
        Q[action] += (reward - Q[action]) / N[action]
        rewards.append(reward)
        arm_selections[action] += 1

    return Q, rewards, arm_selections


def run_thompson_sampling(env, num_rounds=1000):
    n_arms = env.n_arms
    alpha = np.ones(n_arms)
    beta = np.ones(n_arms)
    rewards = []
    arm_selections = np.zeros(n_arms)

    for _ in range(num_rounds):
        sampled_probabilities = [np.random.beta(alpha[a], beta[a]) for a in range(n_arms)]
        action = np.argmax(sampled_probabilities)
        _, reward, _, _, _ = env.step(action)
        if reward == 1:
```

```
        if reward == 1:
            alpha[action] += 1
        else:
            beta[action] += 1
        rewards.append(reward)
        arm_selections[action] += 1

    Q = alpha / (alpha + beta)
    return Q, rewards, arm_selections

env = ThreeArmedBanditEnv()

num_rounds = 1000
q_ucb, rewards_ucb, selections_ucb = run_ucb(env, num_rounds=num_rounds)

q_ts, rewards_ts, selections_ts = run_thompson_sampling(env, num_rounds=num_rounds)

print("UCB Results:")
print("  Estimated Q-values:", q_ucb)
print("  Arm Selections:", selections_ucb)
print("  Total Reward:", sum(rewards_ucb))

print("\nThompson Sampling Results:")
print("  Estimated Q-values:", q_ts)
print("  Arm Selections:", selections_ts)
print("  Total Reward:", sum(rewards_ts))
```

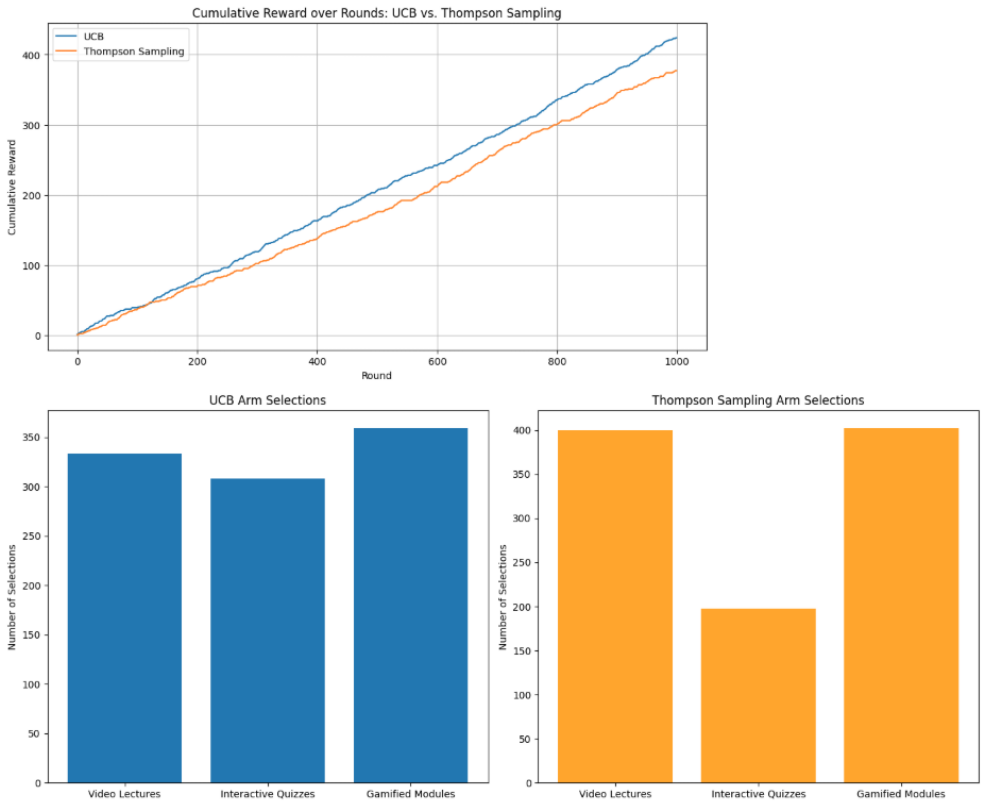
```
UCB Results:
  Estimated Q-values: [0.42342342 0.41233766 0.43454039]
  Arm Selections: [333. 308. 359.]
  Total Reward: 424

Thompson Sampling Results:
  Estimated Q-values: [0.38557214 0.38       0.36881188]
  Arm Selections: [400. 198. 402.]
  Total Reward: 377
```

[7]
```
import matplotlib.pyplot as plt
import numpy as np
plt.figure(figsize=(12, 6))
plt.plot(np.cumsum(rewards_ucb), label='UCB')
plt.plot(np.cumsum(rewards_ts), label='Thompson Sampling')
plt.xlabel('Round')
plt.ylabel('Cumulative Reward')
plt.title('Cumulative Reward over Rounds: UCB vs. Thompson Sampling')
plt.legend()
plt.grid(True)
plt.show()
arms = ['Video Lectures', 'Interactive Quizzes', 'Gamified Modules']
x = np.arange(len(arms))
fig, ax = plt.subplots(1, 2, figsize=(14, 6))
ax[0].bar(x, selections_ucb, tick_label=arms)
ax[0].set_ylabel('Number of Selections')
ax[0].set_title('UCB Arm Selections')
ax[1].bar(x, selections_ts, tick_label=arms, color='orange')
ax[1].set_ylabel('Number of Selections')
ax[1].set_title('Thompson Sampling Arm Selections')

plt.tight_layout()
plt.show()
```

[4]


Cumulative Reward over Rounds: UCB vs. Thompson Sampling
UCB
Thompson Sampling
Cumulative Reward
Round
0
100
200
300
400
0
200
400
600
800
1000

UCB Arm Selections
Number of Selections
Video Lectures
Interactive Quizzes
Gamified Modules
Thompson Sampling Arm Selections
Number of Selections
Video Lectures
Interactive Quizzes
Gamified Modules