```python
# Nihal Ranchod - 2427378
# Lisa Godwin - 2437980

import numpy as np
import matplotlib.pyplot as plt

class Bandit:
    def __init__(self, k):
        self.k = k
        self.means = np.random.normal(0, np.sqrt(3), k)

    def pull(self, arm):
        return np.random.normal(self.means[arm], 1)

def epsilon_greedy(bandit, epsilon, steps):
    k = bandit.k
    Q = np.zeros(k)
    N = np.zeros(k)
    rewards = np.zeros(steps)

    for t in range(steps):
        if np.random.rand() < epsilon:
            arm = np.random.choice(k)
        else:
            arm = np.argmax(Q)

        reward = bandit.pull(arm)
        N[arm] += 1
        Q[arm] += (reward - Q[arm]) / N[arm]
        rewards[t] = reward

    return rewards

def greedy_optimistic(bandit, Q1, steps):
    k = bandit.k
    Q = np.ones(k) * Q1
    N = np.zeros(k)
    rewards = np.zeros(steps)

    for t in range(steps):
        arm = np.argmax(Q)
        reward = bandit.pull(arm)
        N[arm] += 1
        Q[arm] += (reward - Q[arm]) / N[arm]
        rewards[t] = reward

    return rewards

def ucb(bandit, c, steps):
    k = bandit.k
    Q = np.zeros(k)
    N = np.zeros(k)
    rewards = np.zeros(steps)
```

```
54
55      for t in range(steps):
56          if t < k:
57              arm = t
58          else:
59              ucb_values = Q + c * np.sqrt(np.log(t + 1) / (N + 1e-5))
60              arm = np.argmax(ucb_values)
61
62          reward = bandit.pull(arm)
63          N[arm] += 1
64          Q[arm] += (reward - Q[arm]) / N[arm]
65          rewards[t] = reward
66
67      return rewards
68
69  def run_simulation(algorithm, bandit, param, steps, runs):
70      all_rewards = np.zeros((runs, steps))
71      for run in range(runs):
72          rewards = algorithm(bandit, param, steps)
73          all_rewards[run] = rewards
74      return np.mean(all_rewards, axis=0)
75
76  def main():
77      # Parameters
78      k = 10
79      steps = 1000
80      runs = 100
81
82      # Initialize bandit
83      bandit = Bandit(k)
84
85      # Run simulations
86      epsilon_rewards = run_simulation(epsilon_greedy, bandit, 0.1, steps,
    runs)
87      optimistic_rewards = run_simulation(greedy_optimistic, bandit, 5, steps,
    runs)
88      ucb_rewards = run_simulation(ucb, bandit, 2, steps, runs)
89
90      # Plot results
91      plt.figure(figsize=(10, 6))
92      plt.plot(epsilon_rewards, label='Epsilon-Greedy ($\epsilon=0.1$)',
    color='lightseagreen')
93      plt.plot(optimistic_rewards, label='Optimistic Greedy (Q1=5)',
    color='purple')
94      plt.plot(ucb_rewards, label='UCB (c=2)', color='deeppink')
95      plt.xlabel('Steps')
96      plt.ylabel('Average Reward')
97      plt.legend()
98      plt.title('Average Reward over Time')
99      plt.savefig('Avarege_Reward_over_Time.png')
100     plt.show()
101
102
103     # Part 2: Summary of Comparsion Plot with different hyperparameters
104
```

```
    # Hyperparameter values
    epsilon_values = [0.01, 0.1, 0.2, 0.3]
    Q1_values = [5, 3, 1, 0.5]
    c_values = [0.1, 0.5, 1, 2]

    # Average rewards for different hyperparameters
    epsilon_rewards = [np.mean(run_simulation(epsilon_greedy, bandit, epsilon, steps, runs)) for epsilon in epsilon_values]
    Q1_rewards = [np.mean(run_simulation(greedy_optimistic, bandit, Q1, steps, runs)) for Q1 in Q1_values]
    c_rewards = [np.mean(run_simulation(ucb, bandit, c, steps, runs)) for c in c_values]

    # Plot results
    plt.figure(figsize=(10, 6))
    plt.plot(epsilon_values, epsilon_rewards, label='$\epsilon$-greedy', color='lightseagreen')
    plt.plot(Q1_values, Q1_rewards, label='Optimistic Greedy ', color='purple')
    plt.plot(c_values, c_rewards, label='UCB', color='deeppink')
    plt.xscale('log', base=2)
    plt.xlabel('$\epsilon \quad / \quad c \quad / \quad Q_0$')
    plt.ylabel('Average reward over first 1000 steps')
    plt.legend()
    plt.title('Summary comparison of algorithms')
    plt.savefig('Summary_comparison_of_algorithms.png')
    plt.show()


if __name__ == '__main__':
    main()
```