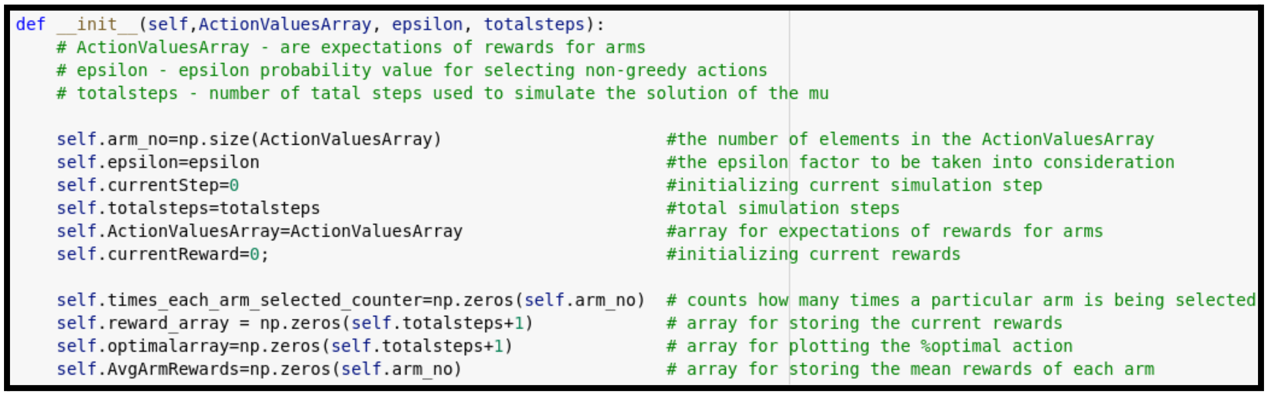
Reinforcement Learning (RBE595)
Programming Assignment 1: Implementation of Epsilon-Greedy Algorithm

Assignment By:
Swapneel Dhananjay Wagholikar (WPI ID: 257598983)
Bhaavin Jogeshwar (WPI ID: 888602054)
Chinmayee Prabhakar (WPI ID: 101177306)

The epsilon-greedy algorithm is a popular method used in reinforcement learning to balance exploration and exploitation. The basic idea behind the algorithm is to choose a random action with a small probability (epsilon) and choose the action that has the highest expected reward with a high probability (1-epsilon). This allows the agent to explore the state space and discover new and potentially better actions while still exploiting the actions that have a high expected reward.

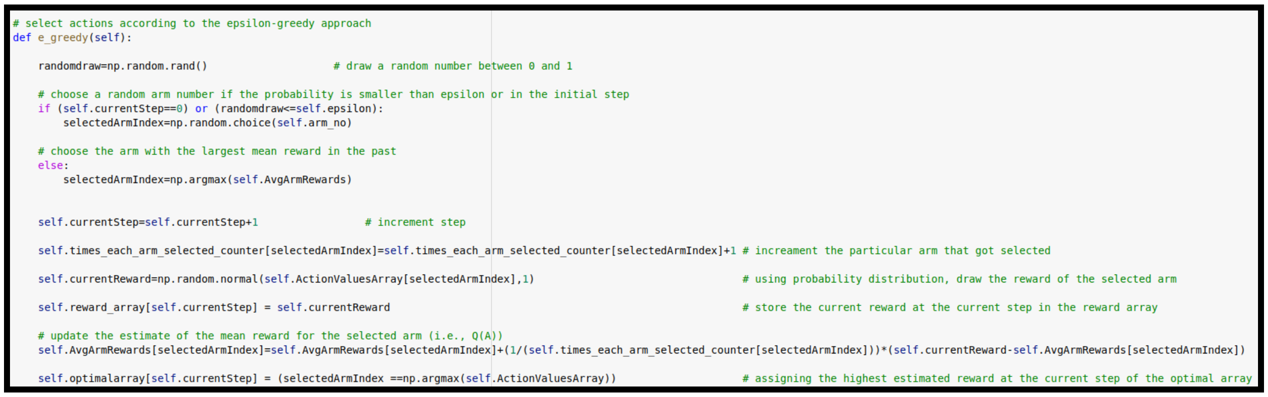
In this programming assignment, Bandit class is coded which is used for different example objects in further parts. The aim was to plot graphs of average reward and %optimal action with respect to simulation steps for three numbers of epsilon values. 1000 simulation steps were considered and the code was iterated for 2000 iterations.

The following factors were taken into account to implement the epsilon-greedy algorithm:



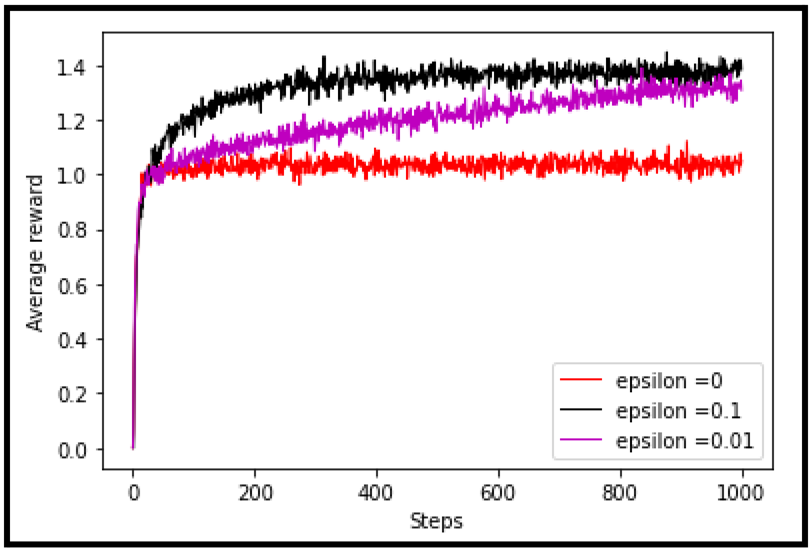
```
def __init__(self,ActionValuesArray, epsilon, totalsteps):
    # ActionValuesArray - are expectations of rewards for arms
    # epsilon - epsilon probability value for selecting non-greedy actions
    # totalsteps - number of tatal steps used to simulate the solution of the mu

    self.arm_no=np.size(ActionValuesArray)                  #the number of elements in the ActionValuesArray
    self.epsilon=epsilon                                    #the epsilon factor to be taken into consideration
    self.currentStep=0                                      #initializing current simulation step
    self.totalsteps=totalsteps                              #total simulation steps
    self.ActionValuesArray=ActionValuesArray                #array for expectations of rewards for arms
    self.currentReward=0;                                   #initializing current rewards

    self.times_each_arm_selected_counter=np.zeros(self.arm_no)  # counts how many times a particular arm is being selected
    self.reward_array = np.zeros(self.totalsteps+1)         # array for storing the current rewards
    self.optimalarray=np.zeros(self.totalsteps+1)           # array for plotting the %optimal action
    self.AvgArmRewards=np.zeros(self.arm_no)                # array for storing the mean rewards of each arm
```

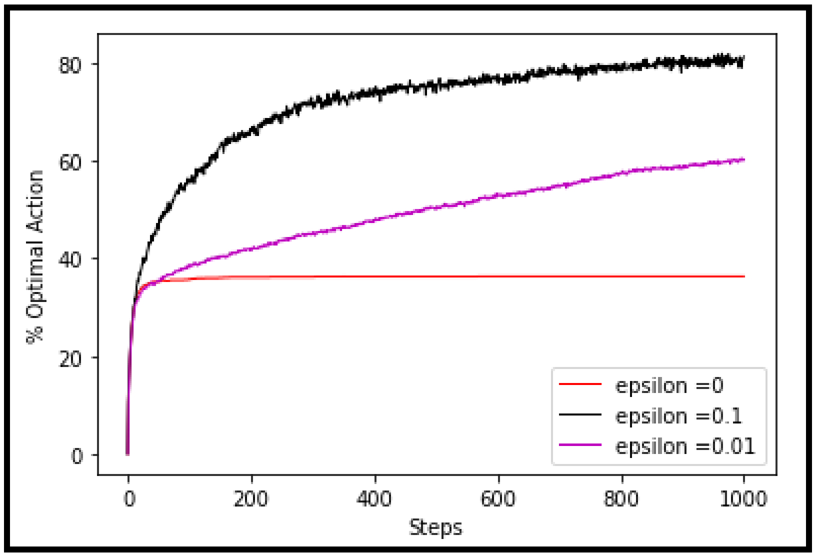
Then as per the above explanation, an epsilon-greedy algorithm is implemented and these factors are updated accordingly:



```
# select actions according to the epsilon-greedy approach
def e_greedy(self):

    randomdraw=np.random.rand()                    # draw a random number between 0 and 1

    # choose a random arm number if the probability is smaller than epsilon or in the initial step
    if (self.currentStep==0) or (randomdraw<=self.epsilon):
        selectedArmIndex=np.random.choice(self.arm_no)

    # choose the arm with the largest mean reward in the past
    else:
        selectedArmIndex=np.argmax(self.AvgArmRewards)


    self.currentStep=self.currentStep+1                 # increment step

    self.times_each_arm_selected_counter[selectedArmIndex]=self.times_each_arm_selected_counter[selectedArmIndex]+1 # increament the particular arm that got selected

    self.currentReward=np.random.normal(self.ActionValuesArray[selectedArmIndex],1)                               # using probability distribution, draw the reward of the selected arm

    self.reward_array[self.currentStep] = self.currentReward                                                      # store the current reward at the current step in the reward array

    # update the estimate of the mean reward for the selected arm (i.e., Q(A))
    self.AvgArmRewards[selectedArmIndex]=self.AvgArmRewards[selectedArmIndex]+(1/(self.times_each_arm_selected_counter[selectedArmIndex]))*(self.currentReward-self.AvgArmRewards[selectedArmIndex])

    self.optimalarray[self.currentStep] = (selectedArmIndex ==np.argmax(self.ActionValuesArray))                 # assigning the highest estimated reward at the current step of the optimal array
```

After the implementation is done, required graphs are plotted. Hereby attaching the results of our implementation:


Average reward
Steps
epsilon =0
epsilon =0.1
epsilon =0.01


% Optimal Action
Steps
epsilon =0
epsilon =0.1
epsilon =0.01