Simulation Notebook

In the real world, we don't know the value of each action that we are going to take
This is also the reason why we can't use Kelly's criteria in real life because we don't priorly know the winning or losing probability

This is where Reinforcement learning comes to the rescue. We don't know the correct action for every state beforehand. Here, we receive rewards based on a series of actions. It is only after running the iteration or cycle multiple times. That you are able to get some optimal results.

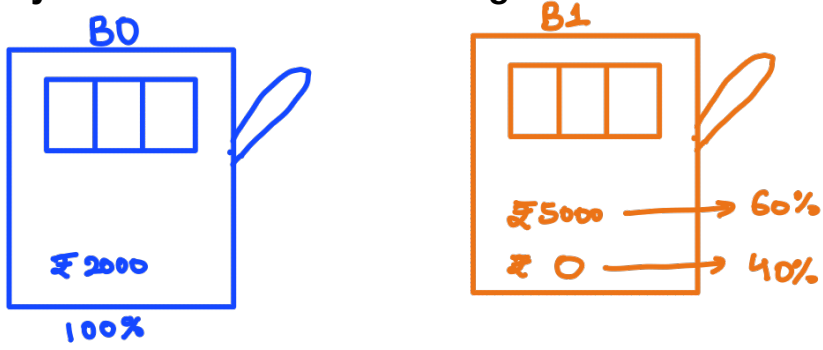
- **The idea of K-armed bandit problem is to maximise reward**
- **But to achieve this goal we need to take best actions**
- **How to find the best actions?**
  - **We need to explore or exploit.**

The need to balance exploration and explotation is a distinctive challenge that arises in reinforcement learning.

PseudoCode of the simulation

```
fix the value of epsilon
initialise allMachineQValueMatrix matrix to be 0

for range in runs:

    :-create an empty array (counterArray) to keep track the number of times an action was taken.

    for trials in trial:

        Step 1: generating a random number using random number generator between 0 and 1

        Step 2: use that value for Action selection
                if random Number is less than or equal to 1 - epsilon
                    then
                    (exploit)
                        :select the action which has got the max expected value
                else
                    (explore)
                        :choose any action with equal probability

                :-store the action that you took above
                :-increment the counter Array for that particular action

        Step 3: reward Allocation
                :-store the reward for the particular action that was taken

        Step 4: Action Value Updation:
            :- Update the allMachineQValueMatrix values for the next trial using the below incremental implementation technique
            NewEstimate <- OldEstimate + stepSize[Target - Old Estimate]

Step 5: plot the allMachineQValueMatrix by taking mean column wise for each bandit machine
```

Our very first reinforcement learning Code



In [204]:

```
"Discrete reward example"

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


def epsilonGreedyMethod(e,    numberOfBanditArms     ,  runs   ,  trials):

    allMachineQValueMatrix = np.zeros((numberOfBanditArms,runs,trials))

    for run in range(runs):

        #starting a fresh game
        actionSelectedCounter = np.zeros(numberOfBanditArms)

        for trial in range(trials):

            #The np.random.uniform() function generates a random float number between a and b, including
            #a but excluding b.
            rand_num = np.random.uniform(0, 1)

            '''
            Step 1: Action selection:
                    e = epsilon = e.g. 0.1
                    probability of exploration = e = e.g. 0.1
                    probability of exploitation = 1 - e = e.g. 0.9

            R is the reward
            A is the action
            '''


            if rand_num <= 1 - e:
                '''
                exploit

                '''



                #storing the expected Q value for each arm for the current trial
                QvalueOfBanditArms = np.zeros(numberOfBanditArms)

                for arm in range(numberOfBanditArms):
                    QvalueOfBanditArms[arm] = allMachineQValueMatrix[arm][run][trial]
```

You will solely focus on increasing your reward

```
                #finding the max expected value of an action (Qvalue) in the QvalueArrayForAllBanditArms
                maxExpectedValue = max(QvalueOfBanditArms)

                #finding which actions are giving me this maxExpectedValue
                shortListedActions = [action for action, estimateValue in enumerate(QvalueOfBanditArms)
                                      if estimateValue == maxExpectedValue]

                #breaking the tie randomly if there are more than one selected actions

                actionLength = len(shortListedActions                                    )
```

```
            if actionLength > 1:

                #choosing an action randomly from the shortListedActions
                finalAction    = random.choice(shortListedActions                    )

            else:
                finalAction = shortListedActions[0]

            #incrementing the counter Array for that particular action
            actionSelectedCounter[finalAction] += 1

                [1, 0, 0]



            A = finalAction [0]

        else:
            '''
            explore
            '''
            in this block you will try to discover unexplored states.

            #selecting randomly from among all the actions with equal probability.
            randActionIndex = np.random.randint(0,numberOfBanditArms)

            #incrementing the counter Array for that particular action
            actionSelectedCounter[randActionIndex] += 1

            A = randActionIndex

        '''
        Step 2: Reward Allocation
        '''

        rand_num = np.random.uniform(0, 1)
        if A == 0:
            R = 2000




        else:
            if rand_num <= 0.6:
                R = 5000
            else:
                R = 0




        '''
        Step 3:
        Action Value Updation:
        NewEstimate <- OldEstimate + stepSize[Target - Old Estimate]

        Matrix Updation For Performance Measurement
        A is actual arm lever that you pulled dowm
        '''
        #if condition to prevent index out of bound error.
        if trial+1 != trials:
            for arm in range(numberOfBanditArms           ):

                if arm == A    :

                    #updation of the expected value for Q for the arm that we pulled down

                    update =  (R - allMachineQValueMatrix[arm][run][trial])/actionSelectedCounter[A]
                    allMachineQValueMatrix[arm][run][trial+1] =allMachineQValueMatrix[arm][run][trial]+update
                else:
                    #updation of the expected Q value for the other arms that we left in this trial
                    if trial != 0:
                        #updating the Q value of the arm of the next trial with the previous trial's value
                        #if this arm was not pulled down
                        allMachineQValueMatrix[arm][run][trial + 1] = allMachineQValueMatrix[arm][run][trial]

    return allMachineQValueMatrix
```

$$Q_{n+1} = Q_n + \frac{1}{n}[R_n - Q_n]$$

Setup

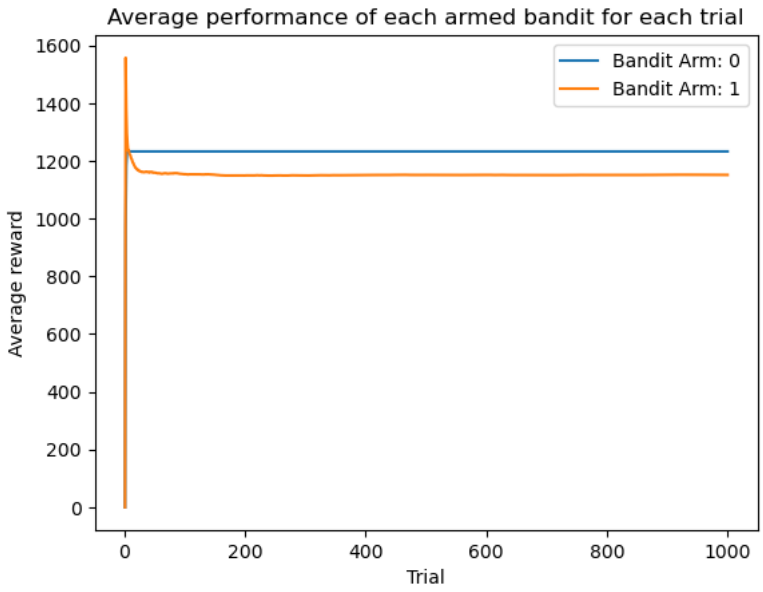
In [205]:
```
runs = 2000
trials = 1000

#1 run = 1000 trials
# we are running this for 2000 * 1000 trials!!!!
```

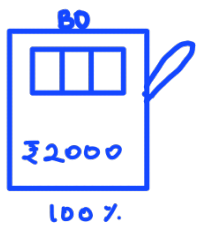
Discrete reward example

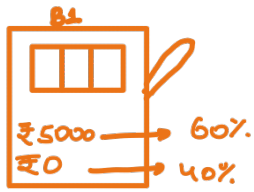
running discrete reward example with epsilon = 0

In [206]:
```
numberOfBanditArms = 2
allMachineQValueMatrix = epsilonGreedyMethod(e = 0, numberOfBanditArms = numberOfBanditArms
                                             , runs = runs, trials = trials)

#print(allMachineQValueMatrix)

column_wise_mean = []
for banditArmIndex in range(numberOfBanditArms):
    column_wise_mean.append(np.mean(allMachineQValueMatrix[banditArmIndex], axis=0))

timeStepArray = np.arange(1,trials + 1)


for i in range(numberOfBanditArms):
    plt.plot(timeStepArray, column_wise_mean[i], label = "Bandit Arm: " + str(i))

plt.legend(loc = "best")
plt.title("Average performance of each armed bandit for each trial")
plt.xlabel("Trial")
plt.ylabel("Average reward")

```

Out[206]: Text(0, 0.5, 'Average reward')





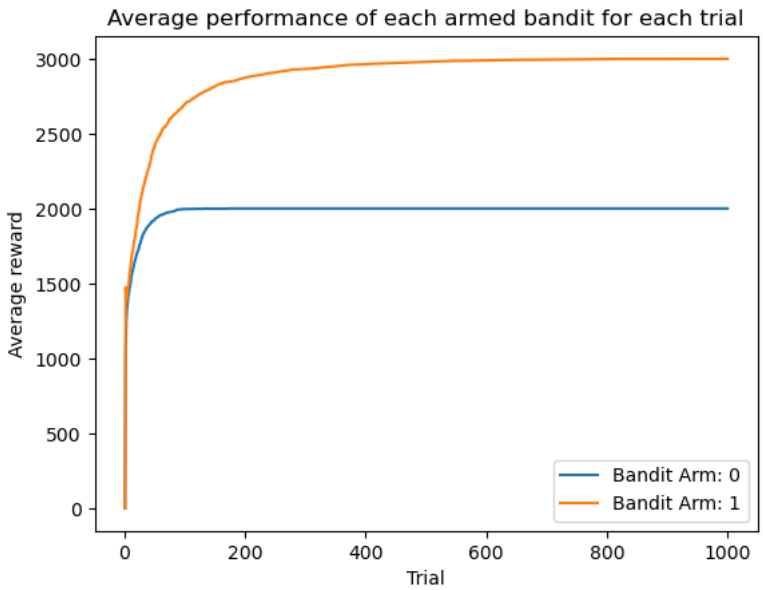


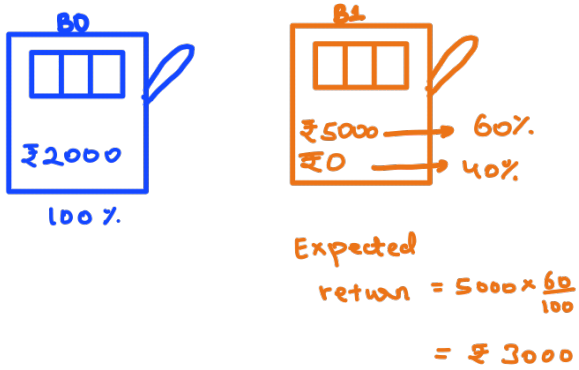
running discrete reward example with epsilon = 0.1

In [207]:

```
allMachineQValueMatrix = epsilonGreedyMethod(e = 0.1, numberOfBanditArms = numberOfBanditArms
                                             , runs = runs, trials = trials)

#print(allMachineQValueMatrix)

column_wise_mean = []
for banditArmIndex in range(numberOfBanditArms):
    column_wise_mean.append(np.mean(allMachineQValueMatrix[banditArmIndex], axis=0))

timeStepArray = np.arange(1,trials + 1)


for i in range(numberOfBanditArms):
    plt.plot(timeStepArray, column_wise_mean[i], label = "Bandit Arm: " + str(i))

plt.legend(loc = "best")
plt.title("Average performance of each armed bandit for each trial")
plt.xlabel("Trial")
plt.ylabel("Average reward")

```

Out[207]: Text(0, 0.5, 'Average reward')





Lets Level up the game

Continuous Probability Reward Distribution

In [ ]:

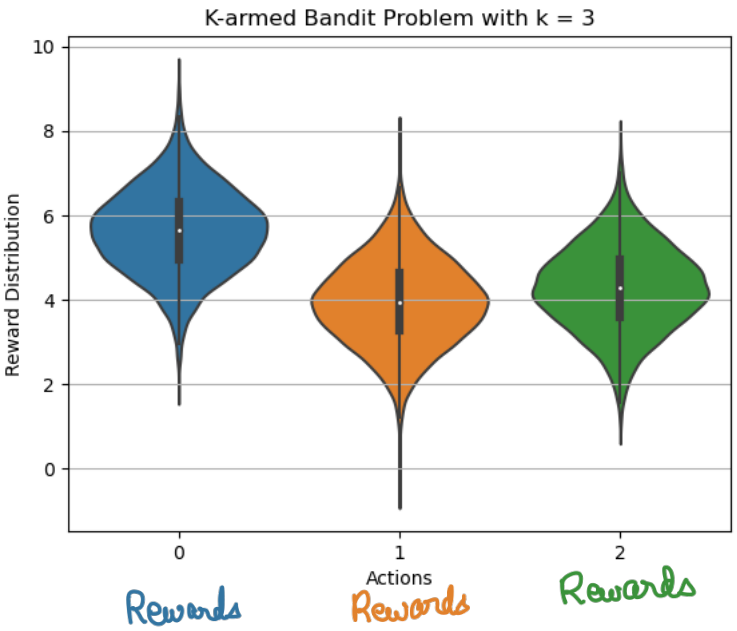
```

```

In [215]:
```
'''
Prerequisite:

Know the terms:
(1) Action = A (pulling the arm of a bandit machine)
(2) Reward = Rt (actual reward selected at time t)
(3) Value:
  (a)True Value = q*(a) of an action (it is the actual value)
  (b)Expected Value = Q*(a) of an action (it is a value based on stochastic method)
'''

'''
Step 1: For each bandit problem, the true action value q*(a), a = 1,...,10 were selected according to
        a normal distribution with mean 0 and variance 1.
(in book the value of k is 10. This times number of actions is 10. This means that we need 10 data points.)
'''
#setting up parameters
runs = 2000
trials = 500
numberOfBanditArms = 3

#setting the true values of each Bandit Arm that is q*(a)
qValuesOfBanditArms = np.random.normal(5, 1, numberOfBanditArms )
qValuesOfBanditArms
```

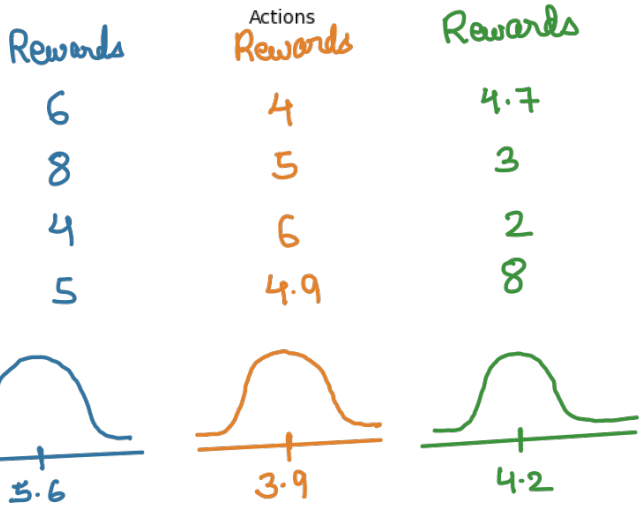
Out[215]:
```
array([5.65257546, 3.9338925 , 4.27232989])
```

Visualising the Continuous Reward Distribution

In [216]:
```
dataPoint = []
arms = len(qValuesOfBanditArms)
for x in range(arms):
    dataPoint.append(np.random.normal(qValuesOfBanditArms[x], 1, 10000))

plt.title("K-armed Bandit Problem with k = " + str(arms)+ "")
plt.grid(True)
sns.violinplot(data=dataPoint, split=True, orient='v', width=0.8, scale='count')
plt.xlabel("Actions")
plt.ylabel("Reward Distribution")
plt.show()


```



Code



In [217]:

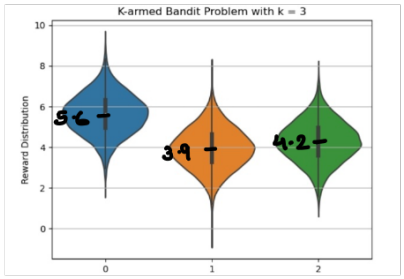
```
import numpy as np
import matplotlib.pyplot as plt
import random



def epsilonGreedyMethod(e, numberOfBanditArms, qValuesOfBanditArms, runs, trials):

    allMachineQValueMatrix = np.zeros((numberOfBanditArms,runs,trials))

    for run in range(runs):

        #starting a fresh game
        actionSelectedCounter = np.zeros(numberOfBanditArms)

        for trial in range(trials):


            rand_num = np.random.uniform(0, 1)

            '''
            Step 1: Action selection:
                    e = epsilon
                    probability of exploration = e
                    probability of exploitation = 1 - e

                    R is the reward
                    A is the action
            '''


            if rand_num <= 1 - e:
                '''
                exploit

                '''

                #storing the expected Q value for each arm for the current trial
                QvalueOfBanditArms = np.zeros(numberOfBanditArms)
                for arm in range(numberOfBanditArms):
                    QvalueOfBanditArms[arm] = allMachineQValueMatrix[arm][run][trial]


                #finding the max expected value of an action (Qvalue) in the QvalueArrayForAllBanditArms
                maxExpectedValue = max(QvalueOfBanditArms)


                #finding which actions are giving me this maxExpectedValue
                shortListedActions = [action for action, estimateValue in enumerate(QvalueOfBanditArms)
                                      if estimateValue == maxExpectedValue]

                #breaking the tie randomly if there are more than one selected actions
                actionLength = len(shortListedActions)
                if actionLength > 1:

                    #choosing an action randomly from the shortListedActions
                    finalAction = random.choice(shortListedActions)

                else:
                    finalAction = shortListedActions[0]

                #print("the agent pulled the arm of bandit machine number ", finalAction)
                actionSelectedCounter[finalAction] += 1

                A = finalAction
                '''
                reward added for action
                '''

            else:
                '''
                explore
                '''

                #selecting randomly from among all the actions with equal probability.
                randActionIndex = np.random.randint(0,numberOfBanditArms)
                #rewardOfRandAction = rewardListForEachBanditArmsAction[randActionIndex]
                actionSelectedCounter[randActionIndex] += 1
                #print("actionSelectedCounter: ", actionSelectedCounter)

                A = randActionIndex


            '''
            Step 2: Reward Allocation
            When learning method applied to that k bandit problem selected action A at time step t,
            the actual reward R, is being selected from a normal distribution with q(A at time t) and variance 1
            '''

            R = np.random.normal(qValuesOfBanditArms[A], 1, 1)
```

mean

variance

no. of data points

[5.6, 3.9, 4.2]

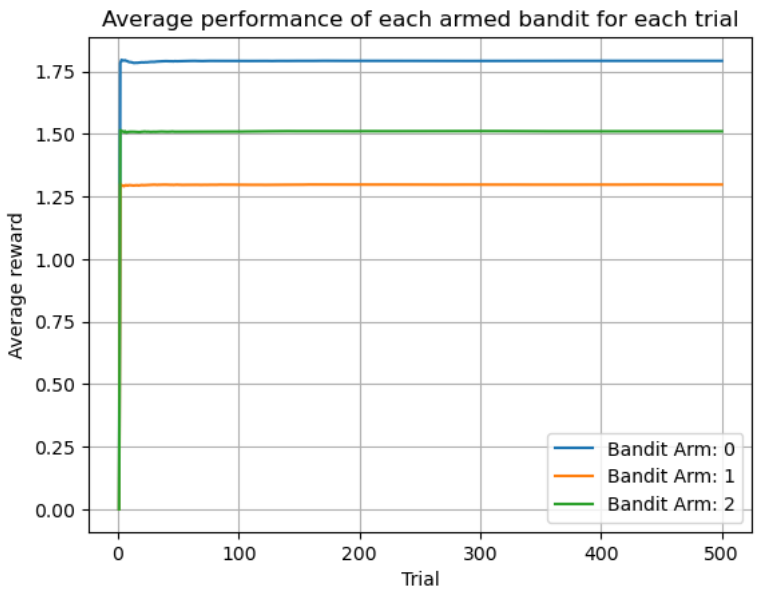



```
            '''
            Step 3:
            Action Value Updation:
            NewEstimate <- OldEstimate + stepSize[Target - Old Estimate]
            '''
            '''
            Matrix Updation For Performance Measurement
            A is actual arm lever that you pulled dowm
            '''
            #if condition to prevent index out of bound error.
            if trial+1 != trials:
                for arm in range(numberOfBanditArms):
                    if arm == A :
                        #updation of the expected value for Q for the arm that we pulled down
                        update = (R - allMachineQValueMatrix[arm][run][trial])/actionSelectedCounter[A]
                        allMachineQValueMatrix[arm][run][trial+1]=allMachineQValueMatrix[arm][run][trial] + update
                    else:
                        #updation of the expected Q value for the other arms that we left in this trial
                        if trial != 0:
                            #basically we are updating the Q value of the arm of the previous trial
                            #if this arm was not pulled down
                            allMachineQValueMatrix[arm][run][trial + 1] = allMachineQValueMatrix[arm][run][trial]

    return allMachineQValueMatrix
```

running continuous probability reward example with epsilon = 0

In [218]:

```
allMachineQValueMatrix = epsilonGreedyMethod(e = 0, numberOfBanditArms = numberOfBanditArms
                                             ,qValuesOfBanditArms=qValuesOfBanditArms,runs = runs,trials = trials)
#print(allMachineQValueMatrix)


column_wise_mean = []
for banditArmIndex in range(numberOfBanditArms):
    column_wise_mean.append(np.mean(allMachineQValueMatrix[banditArmIndex], axis=0))

timeStepArray = np.arange(1,trials + 1)
#print(column_wise_mean)
timeStepArray

for i in range(numberOfBanditArms):
    plt.plot(timeStepArray, column_wise_mean[i], label = "Bandit Arm: " + str(i))

plt.legend(loc = "best")
plt.title("Average performance of each armed bandit for each trial")
plt.xlabel("Trial")
plt.ylabel("Average reward")
plt.grid(True)
plt.show()
```

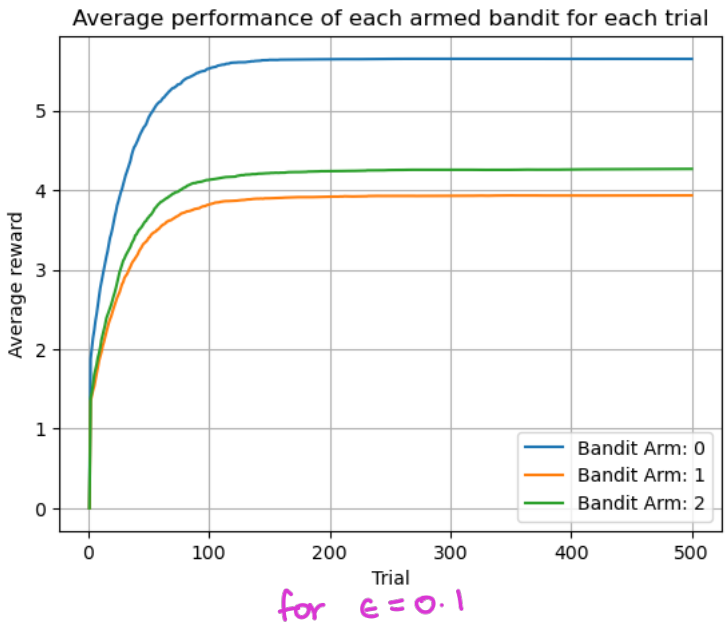


[5.6, 3.9, 4.2]

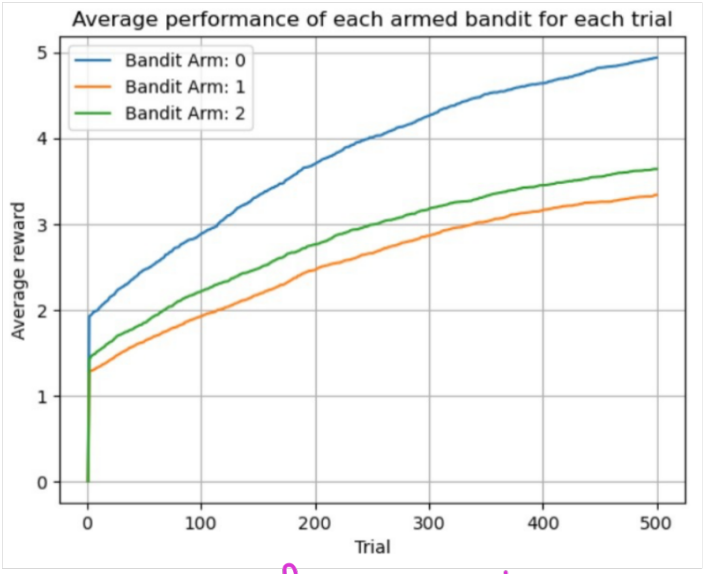
running continuous probability reward example with epsilon = 0.1

In [219]:
```
allMachineQValueMatrix = epsilonGreedyMethod(e = 0.1, numberOfBanditArms = numberOfBanditArms
                                             ,qValuesOfBanditArms = qValuesOfBanditArms,runs=runs,trials = trials)
#print(allMachineQValueMatrix)
column_wise_mean = []
for banditArmIndex in range(numberOfBanditArms):
    column_wise_mean.append(np.mean(allMachineQValueMatrix[banditArmIndex], axis=0))

timeStepArray = np.arange(1,trials + 1)



for i in range(numberOfBanditArms):
    plt.plot(timeStepArray, column_wise_mean[i],label = "Bandit Arm: " + str(i))

plt.legend(loc = "best")
plt.title("Average performance of each armed bandit for each trial")
plt.xlabel("Trial")
plt.ylabel("Average reward")
plt.grid(True)
plt.show()
```

true expected values = [5.6, 3.9, 4.2]



for $\epsilon = 0.1$



for $\epsilon = 0.01$

Let us now make things more interesting

In [220]:
```
#a cute-little adventurous kung fu panda wearing cyberpunk hoodie standing on a slack line on one leg, across two buildings,
```

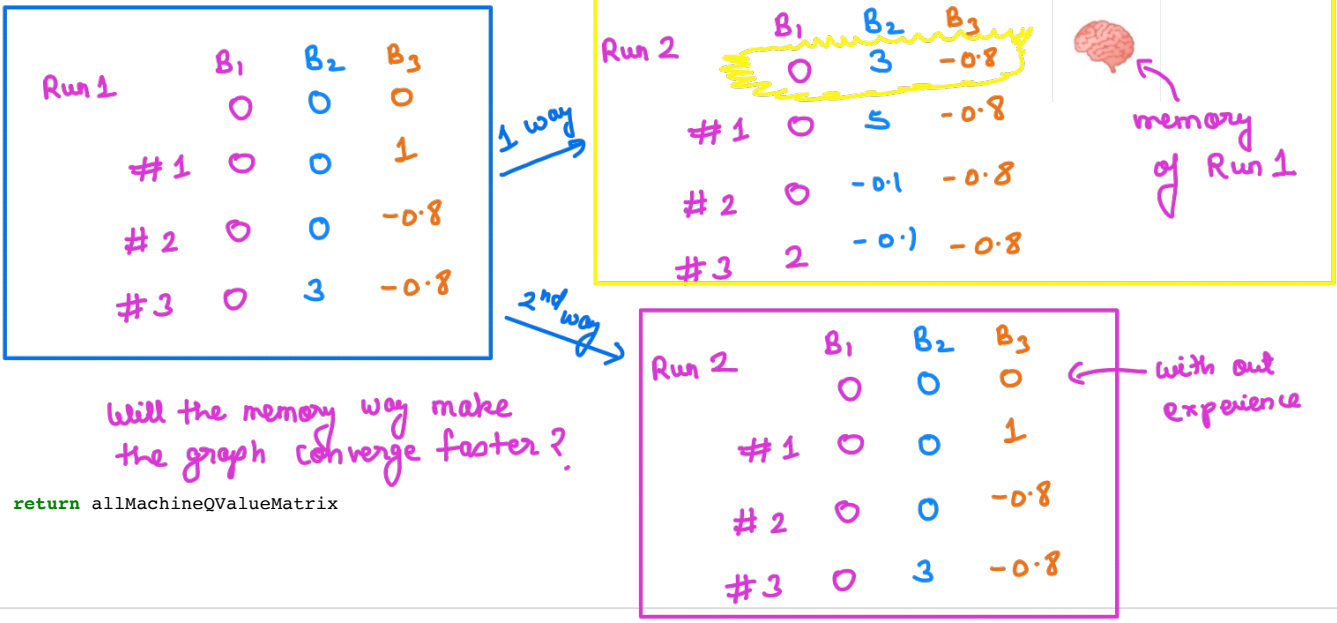
Code

In [221]:

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



def epsilonGreedyMethodMem(e, numberOfBanditArms, qValuesOfBanditArms, runs, trials):

    allMachineQValueMatrix = np.zeros((numberOfBanditArms,runs,trials))

    for run in range(runs):

        #starting a fresh game
        actionSelectedCounter = np.zeros(numberOfBanditArms)

        for trial in range(trials):


            rand_num = np.random.uniform(0, 1)

            '''
            Step 1: Action selection:
                    e = epsilon
                    probability of exploration = e
                    probability of exploitation = 1 - e

                    R is the reward
                    A is the action
            '''


            if rand_num <= 1 - e:
                '''
                exploit

                '''

                #storing the expected Q value for each arm for the current trial
                QvalueOfBanditArms = np.zeros(numberOfBanditArms)

                for arm in range(numberOfBanditArms):
                    QvalueOfBanditArms[arm] = allMachineQValueMatrix[arm][run][trial]

                #finding the max expected value of an action (Qvalue) in the QvalueArrayForAllBanditArms
                maxExpectedValue = max(QvalueOfBanditArms)

                #finding which actions are giving me this maxExpectedValue
                shortListedActions = [action for action, estimateValue in enumerate(QvalueOfBanditArms)
                                      if estimateValue == maxExpectedValue]

                #breaking the tie randomly if there are more than one selected actions
                actionLength = len(shortListedActions)
                if actionLength > 1:

                    #choosing an action randomly from the shortListedActions
                    finalAction = random.choice(shortListedActions)

                else:
                    finalAction = shortListedActions[0]

                #print("the agent pulled the arm of bandit machine number ", finalAction)
                actionSelectedCounter[finalAction] += 1

                A = finalAction
                '''
                reward added for action
                '''

            else:
                '''
                explore
                '''

                #selecting randomly from among all the actions with equal probability.
                randActionIndex = np.random.randint(0,numberOfBanditArms)
                #rewardOfRandAction = rewardListForEachBanditArmsAction[randActionIndex]
                actionSelectedCounter[randActionIndex] += 1
                #print("actionSelectedCounter: ", actionSelectedCounter)

                A = randActionIndex


            '''
            Step 2: Reward Allocation
            When learning method applied to that k bandit problem selected action A at time step t,
            the actual reward R, is being selected from a normal distribution with q(A at time t) and variance 1
            '''

            R = np.random.normal(qValuesOfBanditArms[A], 1, 1)


            '''
            Step 3:
            Action Value Updation:
            NewEstimate <- OldEstimate + stepSize[Target - Old Estimate]
            '''
            '''
            Matrix Updation For Performance Measurement
```

```
            A is actual arm lever that you pulled dowm
            '''
            #if condition to prevent index out of bound error.
            if trial+1 != trials:
                for arm in range(numberOfBanditArms):
                    if arm == A :
                        #updation of the expected value for Q for the arm that we pulled down
                        update = (R - allMachineQValueMatrix[arm][run][trial])/actionSelectedCounter[A]
                        allMachineQValueMatrix[arm][run][trial+1] = allMachineQValueMatrix[arm][run][trial]+update
                    else:
                        #updation of the expected Q value for the other arms that we left in this trial
                        #basically we are updating the Q value of the arm of the previous trial if this arm was not pulled
                        allMachineQValueMatrix[arm][run][trial + 1] = allMachineQValueMatrix[arm][run][trial]
            else:
                if run + 1 != runs:
                    for arm in range(numberOfBanditArms):
                        allMachineQValueMatrix[arm][run + 1][0] = allMachineQValueMatrix[arm][run][trial]

    return allMachineQValueMatrix
```

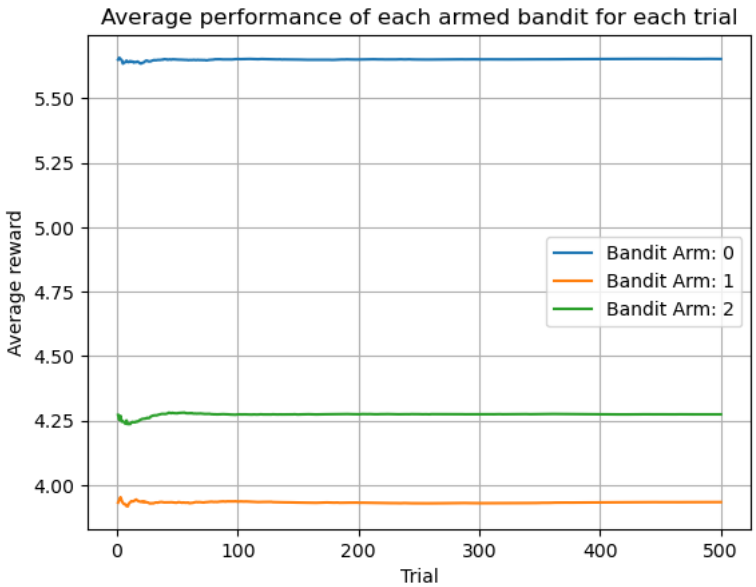
Magic Lines



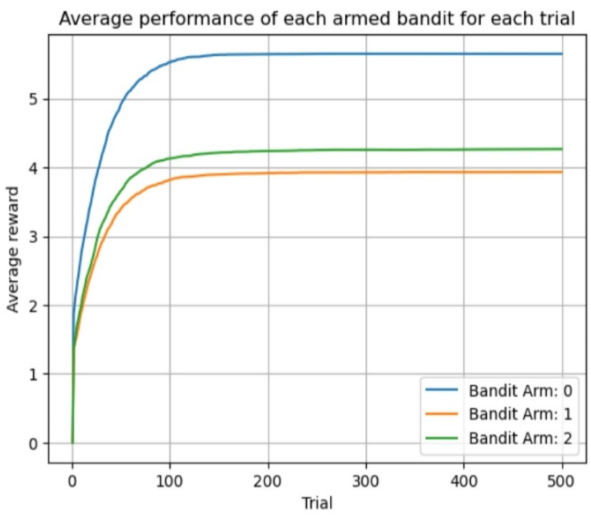
In [222]:

```
runs = 2000
trials = 500
allMachineQValueMatrix = epsilonGreedyMethodMem(e = 0.9, numberOfBanditArms = numberOfBanditArms
                                                ,qValuesOfBanditArms=qValuesOfBanditArms,runs=runs,trials=trials)
#print(allMachineQValueMatrix)
column_wise_mean = []
for banditArmIndex in range(numberOfBanditArms):
    column_wise_mean.append(np.mean(allMachineQValueMatrix[banditArmIndex], axis=0))

timeStepArray = np.arange(1,trials + 1)



for i in range(numberOfBanditArms):
    plt.plot(timeStepArray, column_wise_mean[i],label = "Bandit Arm: " + str(i))

plt.legend(loc = "best")
plt.title("Average performance of each armed bandit for each trial")
plt.xlabel("Trial")
plt.ylabel("Average reward")
plt.grid(True)
plt.show()
```

with



without



In [ ]:

```
1
```

image credits: DALLE.2