Assignment 12

TODO 1:

```
# TODO:1 : write a function that give the probability of choosing arm randomly
def randomize(self,state):
    p_actions = np.zeros(len(state))
    sample_size = 1
    start_x = 0
    cur_prob = 1
    for i in range(len(state)-1):
        prob_i = sp.stats.uniform.rvs(loc = start_x, scale = cur_prob, size = sample_size)
        p_actions[i] = prob_i
        cur_prob = cur_prob - prob_i
    p_actions[-1] = cur_prob
    return p_actions
```

ทำการสุ่มความน่าจะเป็นของ bandit ต่าง ๆ (0 ถึง 1) โดยให้ความน่าจะเป็นของทุก bandit รวมเป็น 1

TODO 2:

```
# TODO:2 : write a function that give the probability of choosing arm based on epsilon greedy policy
def eps_greedy(self, state, t, start_eps=0.3, end_eps=0.01, gamma=0.99):
    if t <= 0:
        return self.equal_weights(state)

    p_actions = np.zeros(len(state))
    miu = np.array([bandit[1] / bandit[0] if bandit[0] != 0.0 else 0 for bandit in state])
    argmax = np.argmax(miu)
    cur_eps = max(start_eps*(gamma**t) , end_eps)
    if np.random.rand() > cur_eps:
        p_actions[argmax] = 1 # exploit
    else:
        rand_bandit = np.random.randint(len(state))
        p_actions[rand_bandit] = 1 # explore
    return p_actions
```

ถ้าหากจำนวนครั้งที่เล่น <= 0 ก็จะให้ความน่าจะเป็นของทุกเครื่องเท่ากัน

แต่ถ้าไม่ ก็จะทำการคำนวณความน่าจะเป็นที่จะได้ reward ของเครื่องต่าง ๆ ใส่ใน miu แล้วหา max (เครื่องที่โอกาสได้ reward สูงสุด) จากนั้นสุ่มเลขมาเลขหนึ่งหากเลยนั้นมีค่ามากกว่า epsilon ปัจจุบันก็จะทำการ exploit เครื่องที่มีโอกาสได้ reward สูงสุด แต่หากน้อยหว่าหรือเท่ากับก็จะทำการ explore เครื่องอื่น ๆ

TODO 3:

```
# TODO:3 : write a function that give the probability of choosing arm based on softmax greedy policy
def softmax(self, state, t, start_tau=1e-1, end_tau=1e-4, gamma=0.9):
    if t <= 0:
        return self.equal_weights(state)

    p_actions = np.zeros(len(state))
    cur_tau = max(start_tau*pow(gamma,t) , end_tau)
    miu = np.array([bandit[1]/bandit[0] if bandit[0] != 0.0 else 0 for bandit in state])
    miu_T = miu/cur_tau
    alter_miu_T = miu_T - np.max(miu_T)
    p_actions = np.exp(alter_miu_T) / np.sum(np.exp(alter_miu_T))
    return p_actions
```

ถ้าหากจำนวนครั้งที่เล่น <= 0 ก็จะให้ความน่าจะเป็นของทุกเครื่องเท่ากัน

แต่ถ้าไม่ ก็จะทำการคำนวณความน่าจะเป็นที่จะได้ reward ของเครื่องต่าง ๆ ใส่ใน miu แล้วนำไปหารด้วย Tau ปัจจุบันและเก็บใน miu_T แต่เนื่องจากมีบางค่า miu_T มีค่าสูงจนเกินไปจนมีค่าเป็น NaN จึงต้องทำการปรับให้ค่าลดลงโดยการนำทุกตัวลบด้วย max ของ miu_T แล้วค่อยนำไปใส่ function softmax

TODO 4:

```
# TODO:4 : write a function that give the probability of choosing arm based on UCB policy
def ucb(self, state, t):
    if t <= 0:
        return self.equal_weights(state)

    p_actions = np.zeros(len(state))
    miu = np.array([bandit[1] / bandit[0] if bandit[0] != 0.0 else 0 for bandit in state])
    n = np.array([bandit[0] if bandit[0] != 0.0 else np.inf for bandit in state])
    confidence_interval = np.sqrt(2*np.exp(1/t)/n)
    argmax = np.argmax(miu + confidence_interval)
    p_actions[argmax] = 1
    return p_actions
```

ถ้าหากจำนวนครั้งที่เล่น <= 0 ก็จะให้ความน่าจะเป็นของทุกเครื่องเท่ากัน

แต่ถ้าไม่ ก็จะทำการคำนวณความน่าจะเป็นที่จะได้ reward ของเครื่องต่าง ๆ ใส่ใน miu และนำจำนวนครั้งในการเล่นเครื่องต่าง ๆ ใส่ใน n จากนั้นคำนวณหา upper bound ของ confidence interval ของเครื่องต่าง ๆ และให้โอกาสการเลือกเครื่องที่ upper bound ของ confidence interval สูงสุดเป็น 1

TODO 5:

จากกราฟสรุปผลข้างต้นจะเห็นว่า ทั้งกราฟ opt_impressions_rate และ total_rate นั้นการใช้วิธีแบบ softmax ให้ค่าสูงที่สุด ในขณะที่กราฟ regret และ regret_rate นั้นวิธีแบบ softmax ก็ให้ค่าที่ต่ำที่สุดเช่นกัน จึงสรุปได้ว่าวิธีแบบ softmax ให้ performance ดีที่สุด