import simpy  
import random  
import math  
  
seed = 2023  
S = 453  
if S > 1000:  
 N = S  
elif S > 10:  
 N = S + 1000  
else:  
 N = S \* 300  
  
  
K = math.ceil(N/24)  
lamd = 1/300.  
mu1 = 1/6.  
mu2 = 0.1  
mu3 = mu1 \* random.uniform(1, 2)  
p = 0.2  
  
bed\_list = []  
service\_times = []  
finish\_time = 1000  
for i in range(K+1):  
 bed\_list.append([])  
  
file\_name = "f.txt"  
  
  
class Person(object):  
 written = 0  
 num\_of\_sick = 0  
 num\_of\_used\_bed = 0  
 last\_time = 0  
  
 def \_\_init\_\_(*self*, *name*, *env*, *decision*):  
 *self*.env = *env  
 self*.name = *name  
 self*.arrival\_t = *self*.env.now  
 *self*.decision = *decision  
 self*.action = *env*.process(*self*.sick())  
  
 def sick(*self*):  
 Person.num\_of\_sick += 1  
 is\_another\_check = False  
 another\_another\_check = False  
 if *self*.decision == 1:  
 if Person.num\_of\_used\_bed >= K:  
 is\_another\_check = True  
 else:  
 random\_time = random.expovariate(mu1)  
 current\_time = *self*.env.now  
 time\_interval = (Person.last\_time, current\_time)  
 bed\_list[Person.num\_of\_used\_bed].append(time\_interval)  
 Person.last\_time = current\_time  
 Person.num\_of\_used\_bed += 1  
 elif *self*.decision == 2:  
 random\_time = random.expovariate(mu2)  
 else:  
 if Person.num\_of\_used\_bed < K:  
 another\_another\_check = True  
 else:  
 random\_time = random.expovariate(mu3)  
 if is\_another\_check:  
 random\_time = random.expovariate(mu3)  
 *self*.decision = 3  
 if another\_another\_check:  
 random\_time = random.expovariate(mu1)  
 current\_time = *self*.env.now  
 time\_interval = (Person.last\_time, current\_time)  
 bed\_list[Person.num\_of\_used\_bed].append(time\_interval)  
 Person.last\_time = current\_time  
 Person.num\_of\_used\_bed += 1  
 *self*.decision = 1  
  
 if Person.written < 81:  
 Person.written += 1  
  
 # print('%s initiated a call at %g, with duration %s, event no = %d, sick people=%d, beds full=%d, decision=%d' % (self.name, self.env.now, random\_time, self.written, self.num\_of\_sick, Person.num\_of\_used\_bed, self.decision))  
  
 ## print('%d\t%s\t%g\t%d\t%d\t%d\tA' % (self.written, self.name, self.env.now, self.num\_of\_sick, self.num\_of\_used\_bed, self.decision))  
  
  
 yield *self*.env.timeout(random\_time)  
 service\_times.append(env.now-*self*.arrival\_t)  
  
  
 Person.num\_of\_sick -= 1  
 if *self*.decision == 1:  
 current\_time = *self*.env.now  
 time\_interval = (Person.last\_time, current\_time)  
 bed\_list[Person.num\_of\_used\_bed].append(time\_interval)  
 Person.last\_time = current\_time  
 Person.num\_of\_used\_bed -= 1  
  
 if Person.written < 10000:  
 Person.written += 1  
  
 # print('%s has healed at %g, event no=%d, sick people=%d, beds full=%d, decision=%d' % (self.name, self.env.now, self.written, self.num\_of\_sick, self.num\_of\_used\_bed, self.decision))  
  
 ## print('%d\t%s\t%g\t%d\t%d\t%d\tD' % (self.written, self.name, self.env.now, self.num\_of\_sick, self.num\_of\_used\_bed, self.decision))  
  
  
  
def people\_generator(*env*, *already\_number*):  
 lambd = 1/300.  
 start\_time = *env*.now  
 count = 0  
 if *already\_number* > K:  
 already\_number = K  
 while *env*.now <= finish\_time+start\_time:  
 if *already\_number* > 0:  
 *already\_number* -= 1  
 count += 1  
 human = Person("P%s" %(count), *env*, 1)  
 continue  
 decision = random.uniform(0, 1)  
 if decision < 0.2 and Person.num\_of\_used\_bed < K:  
 decision = 1  
 elif decision < 0.2:  
 decision = 3  
 else:  
 decision = 2  
  
 yield *env*.timeout(random.expovariate(N\*lambd))  
 count += 1  
 human = Person('P%s' %(count), *env*, decision)  
  
  
env = simpy.Environment()  
operator = simpy.Resource(env, capacity=1)  
env.process(people\_generator(env, 0))  
env.run()  
  
  
def prob\_of\_being\_empty():  
 neu = []  
 for k in bed\_list[0]:  
 neu.append(k[1]-k[0])  
 result\_zeros = sum(neu)  
 return result\_zeros/finish\_time  
  
  
def avg\_sampvar\_occupied\_beds():  
 neu = []  
 for k in range(len(bed\_list)):  
 time\_length = 0  
 for i in bed\_list[k]: # bed\_list[x] represents the time intervals of x beds being used  
 time\_length += i[1] - i[0]  
 neu.append(time\_length\*k)  
 avg\_occupied = sum(neu) / finish\_time  
 temp = 0  
  
 x = 0  
 for i, bed in enumerate(bed\_list):  
 sum\_time = 0  
 for c in bed:  
 sum\_time += c[1] - c[0]  
 avgTime = sum\_time / finish\_time  
 x += (avg\_occupied - i)\*\*2 \* avgTime  
  
  
 for l in range(len(bed\_list)):  
 temp += pow(neu[l] - avg\_occupied, 2)  
 sample\_var\_occupied = temp / (len(bed\_list)-1)  
 return avg\_occupied, x  
  
def avg\_prop\_of\_sick\_on\_population():  
 result = sum(service\_times)  
 return (result/finish\_time)/N # result/finish\_time = avg people in the system(avg sick ppl)  
 #if we divide it with N we get the proportion of sick ppl  
  
  
def total\_avg\_sickness\_time():  
 avg\_sickness = sum(service\_times) / len(service\_times)  
 temp = 0  
 for i in range(len(service\_times)):  
 temp += pow(service\_times[i] - avg\_sickness, 2)  
 sample\_var\_sickness = temp / (len(service\_times)-1)  
 return avg\_sickness, sample\_var\_sickness  
 # we find sample mean and sample variance of duration of sickness times because we do not keep  
 # individual sickness records  
  
  
a = 3  
  
print(prob\_of\_being\_empty())  
x = avg\_sampvar\_occupied\_beds()  
print(x[0])  
print(x[1])  
print(avg\_prop\_of\_sick\_on\_population())  
print(total\_avg\_sickness\_time())