```python
import random
from math import pi, sin, cos, e
from matplotlib.ticker import PercentFormatter
import numpy as np
import matplotlib.pyplot as plt

class Simulation(object):
    def __init__(self, N, L, h, t, v=1, m=10, T=2, Berendsen=1, d_intermolecular=1.5, d_wall=1):
        self.N = N
        self.L = L
        self.h = h
        self.t = t
        self.v = v
        self.m = m
        self.T = T
        self.Berendsen=Berendsen
        self.d_intermolecular = d_intermolecular
        self.d_wall = d_wall
        self.pop = []

        self.K0 = (2 * self.N /2) * self.T
        self.spawn()
        self.runEvent()

    def spawn(self):
        for i in range(self.N):
            new_molecule = Molecule(self.L,self.v)
            while not self.distanceCheck(new_molecule):
                new_molecule = Molecule(self.L, self.v)
            self.pop.append(new_molecule)
            new_molecule.l_position.append((new_molecule.x, new_molecule.y))
            new_molecule.l_velocity.append((new_molecule.v_x, new_molecule.v_y))

    def distanceCheck(self,a):
        if len(self.pop)<1:
            return True
        else:
            for i in self.pop:
                d_m = ((a.x-i.x)**2) + ((a.y-i.y)**2)
                if d_m <= self.d_intermolecular**2:
                    return False
            if (a.x < self.d_wall) or (a.y < self.d_wall) or (a.x > self.L-self.d_wall) or (a.y > self.L-self.d_wall):
                return False
        return True

```

```
    def plotTrajectory(self):

        plt.style.use('ggplot')
        fig = plt.figure()
        ax1=fig.add_subplot(111)

        num=0
        for i in self.pop:
            num+=1
            x = []
            y = []

            x1=i.l_position[0][0]
            y1=i.l_position[0][1]
            ax1.scatter(x1, y1, color='r')
            for j in i.l_position:
                x.append(j[0])
                y.append(j[1])
            ax1.scatter(j[0], j[1], color='b')
            ax1.plot(x,y,'--')

        ax1.set_xlabel("x")
        ax1.set_ylabel("y")
        plt.title('Leonard Jones Simulation, t = ' + str(self.t-0.1))
        plt.show()

    def plotInitialState(self,withVelocity=False):
        x = []
        y = []
        plt.style.use('ggplot')
        fig = plt.figure()
        ax1 = fig.add_subplot(111)

        for i in self.pop:
            x.append(i.l_position[0][0])
            y.append(i.l_position[0][1])
            ax1.annotate("("+str(i.l_position[0][0])[:4]+","+str(i.l_position[0][1])[:4]+")",(i.l_position[0][0],i.l_position[0][1]))
        ax1.scatter(x,y,color='r')

        if withVelocity:
            for i in self.pop:
                ax1.arrow(i.l_position[0][0],i.l_position[0][1],i.l_velocity[0][0],i.l_velocity[0][1],color='k', head_width=0.15)
        ax1.set_xbound(0,self.L)
        ax1.set_ybound(0,self.L)
        ax1.set_xlabel("x")
```

```
        ax1.set_ylabel("y")
        plt.title('Initial State of Atoms')
        plt.legend()
        plt.show()

    def runEvent(self):
        for i in self.pop:
            i.l_position.append((i.x+self.h*i.v_x,i.y+self.h*i.v_y))
            i.x=i.l_position[-1][0]
            i.y=i.l_position[-1][1]

        time = self.h
        while time <= self.t:
            time += self.h
            F_total = []

            K_system=0
            for i in self.pop:
                K_system += 0.5 * self.m * ((i.v_x**2)+(i.v_y**2))

            for i in self.pop:
                F_wall_x = ((self.m)/(i.x**(self.m+1)))+((self.m)/((i.x-self.L)**(self.m+1)))
                F_wall_y = ((self.m)/(i.y**(self.m+1)))+((self.m)/((i.y-self.L)**(self.m+1)))

                F_lj_x = 0
                F_lj_y = 0
                for j in self.pop:
                    if i!=j:
                        r2 = ((i.x-j.x)**2) + ((i.y-j.y)**2)
                        r2_factor = 24 * (2*(1/(r2**7)) - (1/(r2**4)))
                        F_lj_x += (i.x-j.x)*r2_factor
                        F_lj_y += (i.y-j.y)*r2_factor

                F_thermostat_x = self.m * self.Berendsen*((self.K0/K_system)-1)*i.v_x
                F_thermostat_y = self.m * self.Berendsen*((self.K0/K_system)-1)*i.v_y

                F_total_x = F_wall_x + F_lj_x + F_thermostat_x
                F_total_y = F_wall_y + F_lj_y + F_thermostat_y
                F_total_vec = (F_total_x,F_total_y)
                F_total.append(F_total_vec)
```

```

            for i in range(len(self.pop)):
                new_x = 2*self.pop[i].l_position[-1][0] - self.pop[i].l_position[-2][0] + F_total[i][0]*(self.h**2)
                new_y = 2*self.pop[i].l_position[-1][1] - self.pop[i].l_position[-2][1] + F_total[i][1]*(self.h**2)
                self.pop[i].l_position.append((new_x,new_y))

                new_vx = (new_x - self.pop[i].x)/self.h
                new_vy = (new_y - self.pop[i].y)/self.h
                self.pop[i].l_velocity.append((new_vx, new_vy))

                self.pop[i].x = new_x
                self.pop[i].y = new_y
                self.pop[i].v_x = new_vx
                self.pop[i].v_y = new_vy

    def MaxwellSpeedDistribution(self):

        v_3=[]
        v_4=[]
        v_5=[]
        v_6=[]

        for i in self.pop:
            v_3.append(i.l_velocity[100:(10**3):10])
            v_4.append(i.l_velocity[100:(10**4):10])
            v_5.append(i.l_velocity[100:(10**5):10])
            v_6.append(i.l_velocity[100::100])

        for i in [v_3,v_4,v_5,v_6]:
            for j in i:
                for k in range(len(j)):
                    j[k]=(j[k][0]**2)+(j[k][1]**2)
        v_3=np.array(v_3)
        v_4=np.array(v_4)
        v_5=np.array(v_5)
        v_6=np.array(v_6)

        v_3_avg=[]
        v_3_rms=[]

        for i in range(int(np.size(v_3)/self.N)):
            v_3_avg.append(np.mean(v_3[:,i]))
            v_3_rms.append(np.sqrt(np.mean(v_3[:,i]**2)))

        # v_3_avgS=np.std(v_3_avg)
        # v_3_rmsS=np.std(v_3_rms)
```

```
        v_3_avg=np.mean(v_3_avg)
        v_3_rms_m=np.mean(v_3_rms)

        v_4_avg=[]
        v_4_rms=[]

        for i in range(int(np.size(v_4)/self.N)):
            v_4_avg.append(np.mean(v_4[:,i]))
            v_4_rms.append(np.sqrt(np.mean(v_4[:,i]**2)))

        # v_4_avgS=np.std(v_4_avg)
        # v_4_rmsS=np.std(v_4_rms)
        v_4_avg=np.mean(v_4_avg)
        v_4_rms_m=np.mean(v_4_rms)

        v_5_avg=[]
        v_5_rms=[]

        for i in range(int(np.size(v_5)/self.N)):
            v_5_avg.append(np.mean(v_5[:,i]))
            v_5_rms.append(np.sqrt(np.mean(v_5[:,i]**2)))

        # v_5_avgS=np.std(v_5_avg)
        # v_5_rmsS=np.std(v_5_rms)
        v_5_avg=np.mean(v_5_avg)
        v_5_rms_m=np.mean(v_5_rms)

        v_6_avg=[]
        v_6_rms=[]

        for i in range(int(np.size(v_6)/self.N)):
            v_6_avg.append(np.mean(v_6[:,i]))
            v_6_rms.append(np.sqrt(np.mean(v_6[:,i]**2)))

        # v_6_avgS=np.std(v_6_avg)
        # v_6_rmsS=np.std(v_6_rms)
        v_6_avg=np.mean(v_6_avg)
        v_6_rms_m=np.mean(v_6_rms)

        v=np.linspace(0,5,100)
        g=(((1/(2*pi*2))**1.5)*4*pi*(v**2))*(e**((-1)*(v**
    2)/(2*2)))

        plt.style.use('ggplot')
        fig = plt.figure()
        ax1 = fig.add_subplot(221)
        ax1.hist(v_3_rms,weights=np.ones(len(v_3_rms))/len
    (v_3_rms),label=r"M = $10^3$ : " + r"$\bar v$ = " + str(
    v_3_avg)[:6] + r", $v_{rms}$ = " + str(v_3_rms_m)[:6])
        ax1.plot(v,g,label="Maxwell")
```

```

        ax2 = fig.add_subplot(222)
        ax2.hist(v_4_rms,weights=np.ones(len(v_4_rms))/len(v_4_rms),label=r"M = $10^4$ : " + r"$\bar v$ = " + str(v_4_avg)[:6] + r", $v_{rms}$ = " + str(v_4_rms_m)[:6])
        ax2.plot(v,g,label="Maxwell")

        ax3 = fig.add_subplot(223)
        ax3.hist(v_5_rms,weights=np.ones(len(v_5_rms))/len(v_5_rms),label=r"M = $10^5$ : " + r"$\bar v$ = " + str(v_5_avg)[:6] + r", $v_{rms}$ = " + str(v_5_rms_m)[:6])
        ax3.plot(v,g,label="Maxwell")

        ax4 = fig.add_subplot(224)
        ax4.hist(v_6_rms, weights=np.ones(len(v_6_rms)) / len(v_6_rms),label=r"M = $10^6$ : " + r"$\bar v$ = " + str(v_6_avg)[:6] + r", $v_{rms}$ = " + str(v_6_rms_m)[:6])
        ax4.plot(v,g,label="Maxwell")

        ax1.yaxis.set_major_formatter(PercentFormatter(1))
        ax2.yaxis.set_major_formatter(PercentFormatter(1))
        ax3.yaxis.set_major_formatter(PercentFormatter(1))
        ax4.yaxis.set_major_formatter(PercentFormatter(1))
        ax1.set_xlabel("Speed")
        ax1.set_ylabel("Density")
        ax2.set_xlabel("Speed")
        ax2.set_ylabel("Density")
        ax3.set_xlabel("Speed")
        ax3.set_ylabel("Density")
        ax4.set_xlabel("Speed")
        ax4.set_ylabel("Density")
        fig.suptitle('Maxwell Speed Distribution Simulation')
        ax1.legend()
        ax2.legend()
        ax3.legend()
        ax4.legend()
        plt.show()

class Molecule(object):
    def __init__(self, L, v):
        self.L = L
        self.v = v
        self.x = random.uniform(0,1)*self.L
        self.y = random.uniform(0,1)*self.L
        theta=random.uniform(0,2*pi)
        self.v_x = sin(theta)*self.v
        self.v_y = cos(theta)*self.v
        self.l_position = []
        self.l_velocity = []
```

```python


# test=Simulation(N=10,L=20,h=0.001,t=1000.1)

# Problem 3.3
# test.plotInitialState() #a
# test.plotInitialState(True) #b

# Problem 3.4
# m = 10
# h = 0.001
# v0 = 1
# L = 20
# x = [L/2]
# v = [v0]
# x.append(10+0.001*v0)
#
# for i in range(10**5):
#     F_wall_x = (m / (x[-1] ** (m + 1))) + (m / ((x[-1] - L) ** (m + 1)))
#     x_new=2*x[-1]-x[-2]+F_wall_x*(h**2)
#     v.append((x_new-x[-1])/h)
#     x.append(x_new)
# t=[i*h for i in range(0,10**5+2)]
#
# plt.style.use('ggplot')
# fig = plt.figure()
# ax1 = fig.add_subplot(111)
# ax1.plot(t,x,label="Trajectory x(t)")
# t.pop()
# ax1.plot(t,0.5*m*(np.array(v)**2),label="Energy E(t)")
# ax1.set_xbound(-1, h*(10**5))
# ax1.set_ybound(-1, L)
# ax1.legend()
# plt.show()

# Problem 3.7

# test=Simulation(N=10,L=20,h=0.001,t=2.1)
# test=Simulation(N=10,L=20,h=0.001,t=0.2)
# test.plotTrajectory()

# Problem 3.10
# test.MaxwellSpeedDistribution()
```