```
import tkinter
import math
from time import sleep
import random

class Window:
    def __init__(self, xdim, ydim, scale):
        self.root = tkinter.Tk()
        self.canvas = tkinter.Canvas(self.root, bg="white", height=ydim*scale,
width=xdim*scale)
        self.scale = scale
        self.canvas.pack()

    def drawLine(self, x, y, length, angleToVertical):
        self.canvas.create_line(x*self.scale, y*self.scale,
(x+length*math.sin(angleToVertical))*self.scale,
(y+length*math.cos(angleToVertical))*self.scale, width=3)

    def drawLineColour(self, x1, y1, x2, y2, colour):
        self.canvas.create_line(x1*self.scale, y1*self.scale, x2*self.scale,
y2*self.scale, width=1, fill=colour)

class Pendulum:
    def __init__(self, x, y, length, mass, theta, order):
        self.x = x
        self.y = y
        self.v = 0
        self.length = length
        self.mass = mass
        self.theta = theta
        self.order = order

    def draw(self, win):
        win.drawLine(self.x, self.y, self.length, self.theta)

    def getEnd(self):
        return self.x+self.length*math.sin(self.theta),
self.y+self.length*math.cos(self.theta)

    def simulate(self, timestep, val):
        self.v += timestep*val
        self.theta += timestep*self.v
        self.theta = self.theta%(2*math.pi)

global sim,follow
sim = 1
follow = []

def update(win, pendulums, points, showOther=True):
    global sim
    win.canvas.delete("all")
    for a in range(sim):
        if len(points[a]) > 1:
            for i in range(len(points[a])-1):
                if a in follow:
                    win.drawLineColour(points[a][i][0],points[a][i][1],points[a]
[i+1][0],points[a][i+1][1],"blue")
                elif showOther:
```

```
                    win.drawLineColour(points[a][i][0],points[a][i][1],points[a][i+1][0],points[a][i+1][1],"red")
        for p in pendulums[a]:
            p.draw(win)

 win.canvas.create_text(5*math.ceil(math.log(sim,10)),10,fill="black",text=str(sim))
    win.root.update()

timestep = 0.05
initalAngle = math.pi/2 + 0.6*(random.random()-0.5)

win = Window(400,400,2)
allPairs = []
for i in range(sim):

 allPairs.append([Pendulum(200,200,90,10,initalAngle,1),Pendulum(200+90*math.sin(initalAngle),200+90*math.cos(initalAngle),90,10,2*initalAngle,1)])
ps = []
points = [[] for n in range(sim)]
update(win, allPairs, points)

while True:
    for n in range(sim):
        try:
            ps = allPairs[n]
        except IndexError:
            break
        try:
            calc1 = (-ps[1].mass*math.cos(ps[0].theta-ps[1].theta)*ps[0].length*math.pow(ps[0].v,2)*math.sin(ps[0].theta-ps[1].theta)+
                     ps[1].mass*math.cos(ps[0].theta-ps[1].theta)*9.81*math.sin(ps[1].theta)-

ps[1].mass*ps[1].length*math.pow(ps[1].v,2)*math.sin(ps[0].theta-ps[1].theta)-

(ps[0].mass+ps[1].mass)*9.81*math.sin(ps[0].theta))/(ps[0].length*(ps[0].mass+ps[1].mass)-ps[1].mass*math.pow(math.cos(ps[0].theta-ps[1].theta),2))
            ps[0].simulate(timestep, calc1)
            ps[1].x, ps[1].y = ps[0].getEnd()
            calc2 = (ps[0].mass+ps[1].mass)*(ps[0].length*math.pow(ps[0].v,2)*math.sin(ps[0].theta-ps[1].theta)+

((math.pow(ps[1].v,2)*math.sin(ps[0].theta-ps[1].theta)*math.cos(ps[0].theta-ps[1].theta)*ps[1].mass*ps[1].length)/(ps[0].mass+ps[1].mass))+
                                                math.cos(ps[0].theta-ps[1].theta)*9.81*math.sin(ps[0].theta)-

9.81*math.sin(ps[1].theta))/(ps[0].length*(ps[0].mass+ps[1].mass*math.pow(math.sin(ps[0].theta-ps[1].theta),2)))
            ps[1].simulate(timestep, calc2)
            points[n].append(list(ps[1].getEnd()))
            if len(points[n]) > 500:
                points[n].pop(0)
        except OverflowError:
            sim -= 1
            allPairs.pop(n)
            points.pop(n)
    update(win, allPairs, points)

```

```
import tkinter
import math
from time import sleep
import random

class Window:
    def __init__(self, xdim, ydim, scale):
        self.root = tkinter.Tk()
        self.canvas = tkinter.Canvas(self.root, bg="white", height=ydim*scale, width=xdim*scale)
        self.scale = scale
        self.canvas.pack()

    def drawLine(self, x, y, length, angleToVertical):
        self.canvas.create_line(x*self.scale, y*self.scale, (x+length*math.sin(angleToVertical))*self.scale, (y+length*math.cos(angleToVertical))*self.scale, width=3)

    def drawLineColour(self, x1, y1, x2, y2, colour):
        self.canvas.create_line(x1*self.scale, y1*self.scale, x2*self.scale, y2*self.scale, width=1, fill=colour)

class Pendulum:
    def __init__(self, x, y, length, mass, theta, order):
        self.x = x
        self.y = y
        self.v = 0
        self.length = length
        self.mass = mass
        self.theta = theta
        self.order = order

    def draw(self, win):
        win.drawLine(self.x, self.y, self.length, self.theta)

    def getEnd(self):
        return self.x+self.length*math.sin(self.theta), self.y+self.length*math.cos(self.theta)

    def simulate(self, timestep, val):
        self.v += timestep*val
        self.theta += timestep*self.v
        self.theta = self.theta%(2*math.pi)

global sim,follow
sim = int(math.pow(2,11))
#follow = [0,1,3,15,31,63,127,255]
follow = []

def update(win, pendulums, points, showBeams=False, showOther=True):
    global sim
    win.canvas.delete("all")
    for a in range(sim):
        if len(points[a]) > 1:
            for i in range(len(points[a])-1):
                if a in follow:
                    win.drawLineColour(points[a][i][0],points[a][i][1],points[a][i+1][0],points[a][i+1][1],"blue")
                elif showOther:
```

```
                    win.drawLineColour(points[a][i][0],points[a][i][1],points[a][i+1][0],points[a][i+1][1],"red")
        if showBeams:
            for p in pendulums[a]:
                p.draw(win)

 win.canvas.create_text(5*math.ceil(math.log(sim,10)),10,fill="black",text=str(sim))
    win.root.update()

timestep = 0.1
initalAngle = math.pi*0.25
angleStep = 1.5*math.pi/sim

win = Window(400,400,2)
allPairs = []
for i in range(sim):

 allPairs.append([Pendulum(200,200,90,10,initalAngle+angleStep*i,1),Pendulum(200+90*math.sin(initalAngle),200+90*math.cos(initalAngle),90,10,2*(initalAngle+angleStep*i),1)])
ps = []
points = [[] for n in range(sim)]
update(win, allPairs, points)

while True:
    for n in range(sim):
        try:
            ps = allPairs[n]
        except IndexError:
            break
        try:
            calc1 = (-ps[1].mass*math.cos(ps[0].theta-ps[1].theta)*ps[0].length*math.pow(ps[0].v,2)*math.sin(ps[0].theta-ps[1].theta)+
                     ps[1].mass*math.cos(ps[0].theta-ps[1].theta)*9.81*math.sin(ps[1].theta)-
ps[1].mass*ps[1].length*math.pow(ps[1].v,2)*math.sin(ps[0].theta-ps[1].theta)-
(ps[0].mass+ps[1].mass)*9.81*math.sin(ps[0].theta))/(ps[0].length*(ps[0].mass+ps[1].mass)-ps[1].mass*math.pow(math.cos(ps[0].theta-ps[1].theta),2))
            ps[0].simulate(timestep, calc1)
            ps[1].x, ps[1].y = ps[0].getEnd()
            calc2 = (ps[0].mass+ps[1].mass)*(ps[0].length*math.pow(ps[0].v,2)*math.sin(ps[0].theta-ps[1].theta)+
((math.pow(ps[1].v,2)*math.sin(ps[0].theta-ps[1].theta)*math.cos(ps[0].theta-ps[1].theta)*ps[1].mass*ps[1].length)/(ps[0].mass+ps[1].mass))+
                                                math.cos(ps[0].theta-ps[1].theta)*9.81*math.sin(ps[0].theta)-
9.81*math.sin(ps[1].theta))/(ps[0].length*(ps[0].mass+ps[1].mass*math.pow(math.sin(ps[0].theta-ps[1].theta),2)))
            ps[1].simulate(timestep, calc2)
            points[n].append(list(ps[1].getEnd()))
            if len(points[n]) > 2:
                points[n].pop(0)
        except OverflowError:
            sim -= 1
            allPairs.pop(n)
```

```
            points.pop(n)
    update(win, allPairs, points)
```