'''

Self-Organized Criticality in a Pile of Sand

'''

from random import\*

from visual import\*

import numpy as np

from visual.graph import \*

# Initial Conditions

sandrad = 0.5

t = 1

#Grid and Scene

scene=display(x=0,y=0,width=650,height=700,title="Avalanche of Sand",center=(14,14,14),autoscale=0,forward=(0,1,-1),range=(20,20,20))

graph1 = gdisplay(x = 650, y = 0, width = 650, height = 700, title = "Changing Angles as a Function of Time", xtitle = "Time (s)", ytitle ="Angle (degrees)", ymin = 30, ymax = 95)

func1 = gdots(gdisplay = graph1, color = color.orange)

#Scene Fullscreen = 1

for i in arange(0.5,30):

curve(pos=[(i,0.5,0.5),(i,29.5,0.5)],color=color.blue)

curve(pos=[(0.5,i,0.5),(29.5,i,0.5)],color=color.blue)

#Drawing Coordinate Axes

xaxis = curve(pos=[(-5,35,0),(0,35,0)], color = color.white)

yaxis = curve(pos=[(-5,35,0),(-5,30,0)], color = color.white)

zaxis = curve(pos=[(-5,35,0),(-5,35,5)], color = color.white)

xaxislabel = label(pos = (0,35,0), text = "X", xoffset = 5, height = 20)

yaxislabel = label(pos = (-5,30,0), text = "Y", xoffset = -5, height = 20)

zaxislabel = label(pos = (-5,35,5), text = "Z", yoffset = 5, height = 20)

grid=[]

sand=[]

for i in arange(31):

grid.append([])

for j in arange(31):

grid[i].append([])

grid[i][j].append(0) #Number of sand grains at location

grid[i][j].append([]) #Index of each grain at location

grid[i][j].append(0) # 0 if stack is idle, >0 if landed upon

# Function to add a grain of sand to the center of the grid

def drop():

global grid, sand

grid[15][15][0]+=1

sand.append(sphere(pos=(15,15,grid[15][15][0]), color=(random.random(),random.random(),random.random()), radius=sandrad))

grid[15][15][1].append(len(sand)-1)

# Function to describe a grain falling from one pile to an adjacent pile

def fall(x,y,D):

global grid, sand

grid[x+D%3-1][y+1-D/3][0]+=1 #adds one to receiving square

s=grid[x][y][1].pop() #index of sand being moved

grid[x+D%3-1][y+1-D/3][1].append(s) #adds index to receivign square

grid[x][y][0]-=1 #removes one from falling square

sand[s].pos=(x+D%3-1,y+1-D/3,grid[x+D%3-1][y+1-D/3][0]) #Relocates Sand Particles

#while t<300000:

while True:

rate(1000)

drop()

for a in arange (1,31):

for b in arange(1,31):

if len(grid[(a)][(b)][1])>=1: #Only check square with Sand

adjacent=[0,1,2,3,5,6,7,8] #Not to compare to square off the grid

if a==0:

adjacent.remove(0)

adjacent.remove(3)

adjacent.remove(6)

elif a==30:

adjacent.remove(2)

adjacent.remove(5)

adjacent.remove(8)

if b==0:

if a!=0:

adjacent.remove(6)

adjacent.remove(7)

if a!=30:

adjacent.remove(8)

elif b==30:

if a!=0:

adjacent.remove(0)

adjacent.remove(1)

if a!=30:

adjacent.remove(2)

# If adjacent squares have not been compared to, random.random chooses one

while len(adjacent)>0:

d=int(choice(adjacent))

adjacent.remove(d)

# If square has more sand than adjacent square

s=float(grid[a][b][0]-grid[a+d%3-1][b+1-d/3][0]) # Difference in pile sizes

if (s>0) and (random.random()<((s/10)\*\*4\*(1+grid[a][b][2]))): #Larger differences and recently fallen on stacks more likely to fall

grid[a][b][2]=0

numfall=round(s\*(random.random()/2)) #Drop up to half the difference

for i in arange(numfall):

fall(a,b,d)

grid[a+d%3-1][b+1-d/3][2]+=0.2

if grid[a][b][0] == 1:

xmax = a

if grid[a][b][0] == 1:

ymax = b

#Calculate lines of a triangle

#Use lines to solve pythagorean theorem

#To find angle between the rate of angle change

top = vector(15,15,grid[15][15][0]\*sandrad)

bottom = vector(xmax,ymax)

origin = vector(15,15,0)

A = origin - bottom

B = top - bottom

j = diff\_angle(A,B)

angle = j\*(180/np.pi)

t += 1

func1.plot(pos=(t, angle))