一、源文件“引力模拟.py”

# -\*- coding: utf-8 -\*-

"""快捷键:

按Ctrl+“+”或Ctrl+“-”进行缩放。

按↑，↓，←，→键移动。

按“+”或“-”键增加或者降低速度。

单击屏幕开启或关闭轨道显示。

单击行星即可跟随该行星。

拖动鼠标即可发射飞船。

更多帮助，详见“操作说明文档”这一文件。

"""

from time import perf\_counter

from random import randrange

import math,turtle,pickle,os,sys,warnings

from turtle import \*

import tkinter as tk

import tkinter.ttk as ttk

import tkinter.messagebox as msgbox

import tkinter.filedialog as filediag

import pandas as pd

\_\_version\_\_="1.3.3"

PLANET\_SIZE=8 # 天体大小为1时的半径(像素)

# 各个行星的质量

SUN\_MASS=1000000

MERCURY\_MASS=125

VENUS\_MASS=8000

EARTH\_MASS=9000

MOON\_MASS=30

MARS\_MASS=700

PHOBOS\_MASS=2

AST\_MASS=2

JUPITER\_MASS=12000

SATURN\_MASS=6000

URANUS\_MASS=9000

NEPTUNE\_MASS=8000

SPACECRAFT\_MASS = 1

scr=None

def get\_floating\_point\_precision(num):

# 返回num该处浮点数的精度(以2的对数)

return math.floor(math.log2(abs(num))-52) # 如果是单精度浮点数，则52要改为23

def calc\_angle(center,cur,next):

# 计算天体扫过角度(返回单位为度)。参数center为中心，cur为当前位置，next为下一个位置

# 计算三角形三条边的长度

a = math.hypot(center[0]-cur[0],center[1]-cur[1]) # 中心到当前位置

b = math.hypot(center[0]-next[0],center[1]-next[1]) # 中心到下一位置

c = math.hypot(next[0]-cur[0],next[1]-cur[1]) # 当前位置到下一位置

if a==0 or b==0 or c==0:return 0 # 避免ZeroDivisionError

else:

try:return math.acos((a\*\*2+b\*\*2-c\*\*2)/(2\*a\*b)) \* 180 / math.pi # 余弦定理

except ValueError:return 0 # 天体坐标极大时，会发生 ValueError: math domain error

class GravSys:

# 引力系统

\_\_slots\_\_=['planets', 'removed\_planets', 't', 'dt', 'speed','show\_orbit',

'scale', 'scr\_x', 'scr\_y', 'G','enable\_accelerate','enable\_collision',

'key\_x', 'key\_y','startx','starty','calc\_time','elasticity','show\_label',

'show\_tip','\_\_last\_time','writer','pen','fps','following']

def \_\_init\_\_(self):

self.planets = []

self.removed\_planets=[]

self.G = 8

self.t = 0

self.dt = 0.00006 # 时间速度，每次计算经过的时间

#speed: 画面速度，程序在绘制一帧之前执行计算的次数

self.speed=340

self.scale=1 # 缩放比例

self.scr\_x=self.key\_x=0 # scr\_x,scr\_y:视野的偏移距离

self.scr\_y=self.key\_y=0

self.show\_tip=1;self.fps=20

self.calc\_time=0;self.elasticity=0.9

self.enable\_accelerate = True

self.enable\_collision = False

self.show\_orbit = True

self.show\_label = False

self.startx=self.starty=None

w=self.writer=Turtle()

w.ht();w.pu();w.color("white")

p=self.pen=Turtle()

p.ht();p.pu();p.color("green")

Star.\_init\_shape()

#following: 跟随某个行星

self.following=None

def init(self):

for p in self.planets: # 初始化各个行星

p.init()

self.\_\_last\_time=perf\_counter()

def start(self): # 主循环, 最关键的函数

for i in range(len(self.planets)):

p=self.planets[i]

if isinstance(p,Sun):

sun\_index=i;break

else:sun\_index=-1

while True:

calc\_starttime=perf\_counter()

# 计算行星的位置

if self.enable\_accelerate:

self.t += self.dt \* self.speed

# 计算行星的位置,使用numba库加速

lst=[]

for p in self.planets:

lst.extend([p.m,p.x,p.y,p.dx,p.dy,0.0,0.0])

lst = \_acc\_numba(self.speed,float(self.G),float(self.dt),lst,sun\_index)

for i in range(0,len(lst),7):

p=self.planets[i//7]

p.m,p.x,p.y,p.dx,p.dy,\*\_=lst[i:i+7]

else:

for \_ in range(self.speed):

self.t += self.dt

for p in self.planets: # 计算各行星加速度

p.acc()

for p in self.planets: # 计算速度、位移

p.step()

if self.enable\_collision:

for p in self.planets: # 计算碰撞

p.check\_collision()

for p in self.planets:

p.ax=p.ay=0

self.calc\_time=perf\_counter()-calc\_starttime

if self.following!=None:

self.scr\_x=-self.following.x+self.key\_x

self.scr\_y=-self.following.y+self.key\_y

else:

self.scr\_x=self.key\_x

self.scr\_y=self.key\_y

# 刷新行星

for p in self.planets:

p.update()

update()

self.fps=1/(perf\_counter()-self.\_\_last\_time) # 计算帧率

self.\_\_last\_time=perf\_counter()

# 显示文字

if self.show\_tip != 0:

if self.show\_tip==1: # 简明

tip = self.brief\_info()

left\_border=205

else:

tip = self.detailed\_info()

left\_border=410

top\_h=26;fontsize=12

h = (tip.count("\n")+1)\*(fontsize\*3-4)/2 + top\_h # 3和4为调试中推出的数值

self.writer.clear()

self.writer.goto(

scr.window\_width()//2-left\_border,scr.window\_height()//2-h

)

self.writer.write(

tip,

font = (None,fontsize)

)

else:self.writer.clear()

def brief\_info(self):

tip = "fps:%d\n放大倍数: %.4f\n时间速度: %.6f\n" % (

self.fps, self.scale, self.dt)

if self.following:

follow = self.following

tip += "\n正在跟随: %s\n质量: %d" % (follow.name,follow.m)

if getattr(follow,'parent',None):

tip+="\n到%s距离: %d" % (follow.parent.name,

follow.distance(follow.parent))

if isinstance(follow,SpaceCraft):

if getattr(follow,'parent',None):

# 计算飞船相对父天体速度

dx,dy = follow.dx-follow.parent.dx, \

follow.dy-follow.parent.dy

else:dx,dy = follow.dx, follow.dy

tip+="\n飞船速度: %.6f" % math.hypot(dx,dy) # hypot相当于math.sqrt(dx\*\*2+dy\*\*2)

tip+="\n按↑,↓,←,→键控制飞船"

tip += self.status\_info()

tip += "\n\n提示: 按F3键可查看更详\n 细的天体信息。"

else:tip+=self.status\_info()

return tip

def detailed\_info(self):

tip = "fps:{:.0f}\n放大倍数: {:.10g}\n".format(self.fps, self.scale)

tip += "时间: {:#.12g}\t时间速度: {:#.6f}\n".format(self.t, self.dt)

if self.following:

follow = self.following

tip += "\n正在跟随: {}\n质量: {:.10g}".format(follow.name,follow.m)

if getattr(follow,'parent',None):

tip+="\n到{}距离: {}".format(follow.parent.name,

follow.distance(follow.parent))

center = (follow.parent.x,follow.parent.y)

# 计算天体相对父天体速度

dx,dy = follow.dx-follow.parent.dx, \

follow.dy-follow.parent.dy

else:

center = (0,0)

dx,dy = follow.dx, follow.dy

x, y = follow.x, follow.y

tip+="\n坐标: ({:.16g}, {:.16g})".format(x,y)

tip+="\n(相对)速度: ({:.10g}, {:.10g})".format(dx,dy)

tip+="\n\t ({:.16g})\t角度: {:.4f}°".format(

math.hypot(dx,dy), math.atan2(dy,dx)/math.pi\*180)

tip+="\n所受引力合力: {:.10g}".format(follow.calc\_acc()\*follow.m)

if follow.parent:

ecc, semimajor\_axis, angle\_peri=follow.calc\_orbit()

tip+="\n轨道离心率: {:.5g}\t半长轴: {:.8g}".format(ecc,semimajor\_axis)

tip+="\n近日点与水平方向夹角: {:.4f}°".format(angle\_peri)

dx\_s, dy\_s = dx\*self.dt, dy\*self.dt

next\_x, next\_y = x+dx\_s, y+dy\_s

tip+="\n"

tip+="\n单次计算(相对)位移：({:.8g},{:.8g})".format(dx\_s,dy\_s)

# 天体相对父天体公转的角度

tip+="\n单次计算角度误差：{:.8g}°".format(calc\_angle(center,(x,y),(next\_x,next\_y)))

if not (x==0 or y==0):

prec\_x=get\_floating\_point\_precision(x)

prec\_y=get\_floating\_point\_precision(y)

# 倍数为单次计算位移与浮点数精度之比

tip+="\n浮点数精度误差：x:2^{}({:.4g}倍),\n\t\ty:2^{}({:.4g}倍)".format(

prec\_x, abs(dx\_s)/2\*\*prec\_x, prec\_y, abs(dy\_s)/2\*\*prec\_y)

# 程序运行中需要计算和渲染，这是单次循环中计算时间占总运行时间的比例

tip+="\n\n计算时间占运行时间比例：{:.4g}%".format(self.calc\_time/(1/self.fps)\*100)

tip+=self.status\_info(False)

tip+="\n\n提示:可按住下方滚动条暂停模拟，更好地查看数据。"

else:tip+=self.status\_info()

return tip

def status\_info(self,multiline=True):

tip="\n" if multiline else ""

tip+="\n硬件加速已启用" if self.enable\_accelerate else "\n硬件加速已禁用"

tip+="\n" if multiline else " "

tip+="碰撞已启用 弹性:%.4f"%self.elasticity if self.enable\_collision else "碰撞已禁用"

return tip

def follow(self,planet):

self.following=planet

self.key\_x=self.key\_y=0

self.update\_pen\_state()

scr.ontimer(self.clear\_scr, int(1000/self.fps))

def increase\_speed(self,event=None):

if self.enable\_accelerate:

self.dt+=0.000006

else:self.dt+=0.0004

def decrease\_speed(self,event=None):

if self.enable\_accelerate:

self.dt-=0.000006

else:self.dt-=0.0004

def \_update\_size(self):

for planet in self.planets:

scale=planet.\_size\*self.scale

if planet.keep\_on\_scr or self.following is planet:

planet.shapesize(max(0.08,scale))

else:

planet.shapesize(scale)

def zoom(self,scale): # 缩放

self.scale \*= scale

self.\_update\_size()

scr.ontimer(self.clear\_scr, max(int(1000/self.fps),17))

def clear\_scr(self):

for planet in self.planets:

planet.clear()

self.clear\_removed\_planets()

def up(self,event=None):

if isinstance(self.following,SpaceCraft):

self.following.accelerate() # 飞船加速

else:

self.key\_y -= 25 / self.scale

scr.ontimer(self.clear\_scr, max(int(1000/self.fps),17))

def down(self,event=None):

if isinstance(self.following,SpaceCraft):

self.following.slow\_down() # 飞船减速

else:

self.key\_y += 25 / self.scale

scr.ontimer(self.clear\_scr, max(int(1000/self.fps),17))

def left(self,event=None):

if isinstance(self.following,SpaceCraft):

self.following.turn\_left() # 飞船左转弯

else:

self.key\_x += 25 / self.scale

scr.ontimer(self.clear\_scr, max(int(1000/self.fps),17))

def right(self,event=None):

if isinstance(self.following,SpaceCraft):

self.following.turn\_right() # 飞船右转弯

else:

self.key\_x -= 25 / self.scale

scr.ontimer(self.clear\_scr, max(int(1000/self.fps),17))

def update\_pen\_state(self): # 更新行星轨道的隐藏和显示

follow=self.following

children=getattr(follow,"children",[])

if self.show\_orbit:

for planet in self.planets:

planet.clear()

# 如果跟随的天体不是太阳，则显示其子天体轨道

if planet.has\_orbit or \

(not isinstance(follow,Sun)\

and planet in children):

planet.pendown()

else:planet.penup()

else:

for planet in self.planets:

planet.clear()

planet.penup()

def switchpen(self):

self.show\_orbit=not self.show\_orbit

self.update\_pen\_state()

def onclick(self,x,y): # 用于处理鼠标单击

targets=[]

for planet in self.planets:

psize=max(planet.getsize()\*1.375, 2)

if abs(planet.xcor()-x) <= psize \

and abs(planet.ycor()-y) <= psize \

and planet is not self.following:

targets.append(planet)

if targets:self.follow(max(targets,key=lambda p:p.m))

else:

self.switchpen()

self.clear\_removed\_planets()

def \_onclick(self,event): # 鼠标按下事件

x, y = (scr.cv.canvasx(event.x)/scr.xscale,

-scr.cv.canvasy(event.y)/scr.yscale)

self.startx,self.starty=x,y

def \_ondrag(self,event): # 鼠标拖曳事件

scr.\_canvas.unbind("<B1-Motion>")

pen = self.pen

x, y = (scr.cv.canvasx(event.x)/scr.xscale, -scr.cv.canvasy(event.y)/scr.yscale)

pen.clear()

if math.hypot(x - self.startx, y - self.starty) >= 9: # 鼠标移动的距离足够大

pen.goto(self.startx,self.starty)

pen.dot(3) # 绘制圆点

pen.pendown()

pen.goto(x, y) # 绘制线条

v = math.hypot((x-self.startx)/self.scale, (y-self.starty)/self.scale)

pen.write("速度: {:.4g}".format(v),font = (None,11))

pen.penup()

scr.\_canvas.bind("<B1-Motion>",self.\_ondrag)

def \_onrelease(self,event): # 鼠标释放事件

if self.startx is None or self.starty is None:return

x, y = (scr.cv.canvasx(event.x)/scr.xscale,

-scr.cv.canvasy(event.y)/scr.yscale)

self.pen.clear()

if math.hypot(x - self.startx,

y - self.starty) < 9: # 鼠标移动的距离较小

self.onclick(x,y) # 处理鼠标单击

return

# “发射”飞船

pos = (self.startx/self.scale - self.scr\_x,

self.starty/self.scale - self.scr\_y)

if self.following: # 加入已跟踪天体的速度

dx=self.following.dx;dy=self.following.dy

else:dx=dy=0

v = ((x - self.startx)/self.scale + dx,

(y - self.starty)/self.scale + dy)

ship=SpaceCraft(self,SPACECRAFT\_MASS,pos,v,parent=self.following)

if self.show\_orbit:ship.pendown()

else:ship.penup()

self.startx=self.starty=None

def clear\_removed\_planets(self): # 清除已移除天体留下的轨道

for planet in self.removed\_planets:

planet.clear()

self.removed\_planets=[]

def remove(self,planet): # 移除天体

# 由于需要让天体的轨道保留一段时间，因此这里不调用clear\_removed\_planets

self.removed\_planets.append(planet)

self.planets.remove(planet)

planet.\_size = 0

planet.hideturtle()

parent=planet.parent

for child in planet.children:

child.parent=parent # 更新父天体

if parent:parent.children.append(child)

for i in range(len(self.planets)): # 更新天体索引

self.planets[i].\_index = i

def \_switch(self,dt):

# 切换到上/下一个行星

if not self.planets:return # 空列表

if self.following is None or self.following not in self.planets:

index=0

else:

index=self.planets.index(self.following)+dt

index = index % len(self.planets) # 控制index的范围

self.follow(self.planets[index])

def switch(self,event=None):

self.\_switch(1)

def reverse\_switch(self,event=None):

self.\_switch(-1)

def del\_planet(self,event=None): # 删除当前跟踪的行星

if self.following in self.planets:# if self.following is not None:

self.remove(self.following)

if self.following.parent:

self.follow(self.following.parent)

else:self.following=None

def clear\_spacecrafts(self,event=None):

for p in self.planets.copy(): # copy(): 避免更新self.planets导致循环不全

if type(p) is SpaceCraft:

self.remove(p)

if self.following not in self.planets and \

getattr(self.following,"parent",None):

self.follow(self.following.parent)

# 以下函数用于pickle保存状态功能

def \_\_new\_\_(cls): # 避免pickle中引发AttributeError

o=super().\_\_new\_\_(cls)

o.\_\_init\_\_()

return o

def \_\_getstate\_\_(self):

keys = ['t', 'dt', 'speed','scale', 'scr\_x', 'scr\_y',

'key\_x', 'key\_y','show\_tip','G','show\_orbit']

config = {}

for key in keys:

config[key]=getattr(self,key)

if self.following and self.following in self.planets:

config["following\_index"]=self.planets.index(self.following)

else:

config["following\_index"]=None

return self.planets, config

def \_\_setstate\_\_(self,state):

self.planets=state[0]

config=state[1]

index = config.pop("following\_index")

if index is not None:self.following=self.planets[index]

for key in config:

setattr(self,key,config[key])

# 将行星移动到新的大小和位置

self.\_update\_size()

for planet in self.planets:

planet.update()

# ----------------------应用numba模块提高速度--------------------------------

# lst的每7项分别是一个天体的质量，x、y坐标，x、y速度和x、y加速度

# 由于numba库不支持二维数组，也就是列表映射(List Reflection)，就使用一维数组代替

def \_acc\_numba(steps,G,dt,lst,sun\_index):

for \_ in range(steps):

# 计算加速度

index=sun\_index\*7

for i in range(0,len(lst),7):

for j in range(i+7,len(lst),7):

dx=lst[j+1]-lst[i+1]

dy=lst[j+2]-lst[i+2]

if not (dx==0 and dy==0): # 忽略除零的异常

b = G / math.hypot(dx,dy)\*\*3

if sun\_index==-1 or i!=index: # 太阳不移动

lst[i+5]+=b \* dx \* lst[j+0]

lst[i+6]+=b \* dy \* lst[j+0]

lst[j+5]-=b \* dx \* lst[i+0]

lst[j+6]-=b \* dy \* lst[i+0]

# 计算新的速度和位置

for i in range(0,len(lst),7):

lst[i+3] += dt\*lst[i+5]

lst[i+4] += dt\*lst[i+6]

lst[i+5]=lst[i+6]=0

lst[i+1] += dt\*lst[i+3]

lst[i+2] += dt\*lst[i+4]

return lst

try:

# 使用更快的C语言编译成的pyd文件提升速度

from solar\_system\_accelerate\_util import accelerate as \_acc\_numba

except ImportError:

warnings.warn("Failed to import module solar\_system\_accelerate\_util, using module numba instead.")

from numba import jit

\_acc\_numba = jit(nopython=True)(\_acc\_numba) # 相当于 @jit(nopython=True)

#--------------------------------------------------------------------------------

class Star(Turtle):

\_light=\_dark=\_circle=None

def \_\_init\_\_(self, gravSys, name, m, x, v,

shapesize=1,has\_orbit=True,

parent=None,keep\_on\_scr=False,rotation=None,sun=None,

shape=("#b3b3b3","#4d4d4d","gray30")):

Turtle.\_\_init\_\_(self)

self.name=name

self.gs = gravSys

self.\_shape=shape

self.\_size=shapesize

self.m = m

self.x,self.y=x

self.dx,self.dy=v

self.ax=self.ay=0

self.has\_orbit=has\_orbit

self.keep\_on\_scr = keep\_on\_scr

self.rotation=rotation

# self.sun用于行星、卫星的朝向，self.parent用于标识父天体

self.sun=sun or (self.gs.planets[0] if len(self.gs.planets) else None)

self.parent=parent or self.sun

gravSys.planets.append(self)

self.\_index=self.gs.planets.index(self)

self.orbit\_color=None

self.init\_shape()

self.penup()

self.setpos(self.x,self.y)

self.resizemode("user")

self.setundobuffer(None)

w=self.writer=Turtle()

w.ht();w.pu();w.color("white")

self.children=[]

if parent:parent.children.append(self)

elif self.sun:self.sun.children.append(self)

def init(self):

self.update() # 使行星的turtle移动到初始位置

self.clear() # 清除轨迹

self.m=float(self.m);self.x=float(self.x);self.y=float(self.y)

self.dx=float(self.dx);self.dy=float(self.dy)

if self.has\_orbit:

self.pendown()

def acc(self):

# \*\* 计算行星的引力、加速度 \*\*

G = self.gs.G

for i in range(self.\_index+1,len(self.gs.planets)):

planet=self.gs.planets[i]

dx=planet.x-self.x

dy=planet.y-self.y

try:

# 简化前的代码

#r = math.hypot(dx,dy)

#f = G \* self.m \* planet.m / r\*\*2

# 将力分解为水平、竖直方向的力

#ax+=f / self.m \* dx / r

#ay+=f / self.m \* dy / r

b = G / math.hypot(dx,dy)\*\*3

self.ax+=b \* dx \* planet.m

self.ay+=b \* dy \* planet.m

planet.ax-=b \* dx \* self.m

planet.ay-=b \* dy \* self.m

except ZeroDivisionError:pass

def step(self):

# 计算行星位置

dt = self.gs.dt

self.dx += dt\*self.ax

self.dy += dt\*self.ay

self.x+= dt\*self.dx

self.y+= dt\*self.dy

def update(self):

self.setpos((self.x+self.gs.scr\_x)\*self.gs.scale,

(self.y+self.gs.scr\_y)\*self.gs.scale)

if self.rotation is not None:

self.left(self.rotation\*self.gs.dt)

elif self.sun:

self.setheading(self.towards(self.sun))

if self.gs.show\_label:self.show\_label()

else:self.writer.clear()

#if abs(self.x)>14000 or abs(self.y)>14000:

# self.gs.remove(self) # 清除已飞出太阳系的天体

def check\_collision(self):

# 碰撞检测（备用的函数）

for planet in self.gs.planets:

if planet is self:continue

if self.hit(planet):

m1=self.m;m2=planet.m

adx=(self.dx+planet.dx)/2

ady=(self.dy+planet.dy)/2

dx1 = (m1-m2)/(m1+m2)\*self.dx + 2\*m2/(m2+m1)\*planet.dx

dy1 = (m1-m2)/(m1+m2)\*self.dy + 2\*m2/(m2+m1)\*planet.dy

dx2 = (m2-m1)/(m1+m2)\*planet.dx + 2\*m1/(m2+m1)\*self.dx

dy2 = (m2-m1)/(m1+m2)\*planet.dy + 2\*m1/(m2+m1)\*self.dy

rate = self.gs.elasticity # 碰撞的"弹性", 0为完全非弹性碰撞, 1为弹性碰撞

self.dx=dx1\*rate+adx\*(1-rate)

self.dy=dy1\*rate+ady\*(1-rate)

planet.dx=dx2\*rate+adx\*(1-rate)

planet.dy=dy2\*rate+ady\*(1-rate)

dx=planet.x-self.x;dy=planet.y-self.y

dis=math.hypot(dx,dy)

newdis=(self.\_size + planet.\_size) \* PLANET\_SIZE

self.x=planet.x-(dx\*newdis/dis+dx)/2

self.y=planet.y-(dy\*newdis/dis+dy)/2

planet.x=self.x+(dx\*newdis/dis+dx)/2

planet.y=self.y+(dy\*newdis/dis+dy)/2

def hit(self,other): # 新增

return self.distance(other) < \

self.\_size \* PLANET\_SIZE + other.\_size \* PLANET\_SIZE

def getsize(self): # 返回行星的显示大小(直径)

return self.\_stretchfactor[0]\*PLANET\_SIZE\*2

def distance(self,other):

return math.hypot(self.x-other.x,

self.y-other.y)

def grav(self,other,r=None):

# 计算两行星间的引力, F = G \*m1\*m2 / r\*\*2

if r is None:

dx=other.x-self.x; dy=other.y-self.y

r = math.hypot(dx,dy)

return self.gs.G \* self.m \* other.m / r\*\*2

def calc\_acc(self):

# 计算自身加速度大小并返回

G = self.gs.G; ax=ay=0

for planet in self.gs.planets:

dx=planet.x-self.x

dy=planet.y-self.y

try:

b = G / math.hypot(dx,dy)\*\*3

ax+=b \* dx \* planet.m

ay+=b \* dy \* planet.m

except ZeroDivisionError:pass

return math.hypot(ax,ay)

def calc\_orbit(self): # 计算轨道参数

if self.parent is None:

raise ValueError("Celestial Body %s has no parent"%self.name)

else:parent=self.parent

# 计算相对位置矢量和相对速度矢量

dx, dy = self.x - parent.x, self.y - parent.y

vx, vy = self.dx - parent.dx, self.dy - parent.dy

r = math.hypot(dx, dy)

v = math.hypot(vx, vy)

mu = self.gs.G \* (parent.m + self.m)

ecc\_vec = ((v \*\* 2 - mu / r) \* dx - (dx \* vx + dy \* vy) \* vx,

(v \*\* 2 - mu / r) \* dy - (dx \* vx + dy \* vy) \* vy)

ecc = math.hypot(ecc\_vec[0], ecc\_vec[1]) / mu

semimajor\_axis = 1 / (2 / r - v \*\* 2 / mu)

#pos\_peri = ((a \* (1 - ecc)) \* ecc\_vec[0] / ecc, (a \* (1 - ecc)) \* ecc\_vec[1] / ecc) # 近日点坐标

angle\_peri = math.atan2(ecc\_vec[1], ecc\_vec[0]) % (math.pi\*2) / math.pi\*180

return ecc, semimajor\_axis, angle\_peri

def getOrbitSpeed(self,r=None,other=None):

# 获取某一半径的圆轨道上天体的速率

# 引力=向心力=m \* v\*\*2 / r

other=other or self.sun

r=r or self.distance(other)

return math.sqrt(self.grav(other,r) \* r / self.m)

def getHillSphere(self,other=None):

# 获取行星希尔球半径（备用的函数）

# 希尔球是环绕在天体（像是行星）周围的空间区域，其中被它吸引的天体受到它的控制，而不是被它绕行的较大天体（像是恒星）所控制。

other=other or self.parent

return self.distance(other) \* (self.m/(other.m\*3)) \*\* (1/3)

@classmethod

def \_init\_shape(cls,QUALITY=32):

if cls.\_light and cls.\_dark and cls.\_circle:return # 已经初始化

s = Turtle()

s.reset()

s.ht()

s.pu()

s.fd(PLANET\_SIZE)

s.lt(90)

s.begin\_poly()

s.circle(PLANET\_SIZE, 180,steps=QUALITY//2)

s.end\_poly()

cls.\_light = s.get\_poly()

s.begin\_poly()

s.circle(PLANET\_SIZE, 180,steps=QUALITY//2)

s.end\_poly()

cls.\_dark = s.get\_poly()

s.begin\_poly()

s.circle(PLANET\_SIZE,steps=QUALITY)

s.end\_poly()

cls.\_circle = s.get\_poly()

s.hideturtle()

def init\_shape(self):

# 初始化turtle的形状

# shape表示方式:

# (亮色, 暗色, [轨道颜色]) (一半亮，一半暗)

# (颜色,) (一个圆)

# (形状名称, 颜色) (自定义形状)

# () (无形状)

if len(self.\_shape) == 0:return

shape = Shape("compound")

\_shape=self.\_shape;\_name=self.name

if \_shape[0] not in scr.\_shapes:

# \_shape[0]为颜色

if len(\_shape) >= 2: # (亮色, 暗色, [轨道颜色])

shape.addcomponent(self.\_light,\_shape[0])

shape.addcomponent(self.\_dark,\_shape[1])

self.orbit\_color = \_shape[2] if len(\_shape)>=3 else \_shape[0] # 无轨道颜色时默认以亮色代替

else: # (颜色,)

shape.addcomponent(self.\_circle,\_shape[0])

self.orbit\_color = \_shape[0]

self.color(\_shape[0])

scr.register\_shape(\_name, shape)

else:

# \_shape[0]为形状

\_name=\_shape[0]

self.orbit\_color = \_shape[1]

self.color(\_shape[1])

self.shape(\_name)

self.shapesize(self.\_size)

self.pencolor(self.orbit\_color)

def show\_label(self):

self.writer.clear()

self.writer.setpos(\*self.pos())

self.writer.write(self.name,font=(None,10))

def \_\_repr\_\_(self):

return object.\_\_repr\_\_(self)[:-1] + " Name: %s>"%self.name

# 用于pickle模块保存数据

def \_\_getstate\_\_(self):

return (self.gs,self.name,self.m,(self.x,self.y),(self.dx,self.dy),

self.\_size,self.has\_orbit,self.parent,self.keep\_on\_scr,self.rotation,

self.sun,self.\_shape)

def \_\_setstate\_\_(self,state):

self.\_\_init\_\_(\*state)

# 修复turtle模块绘制RoundStar的缺陷

def \_dot(self, pos, size, color):

dt=size/2

return self.cv.create\_oval(pos[0]-dt,-(pos[1]-dt),

pos[0]+dt,-(pos[1]+dt),

fill=color,outline=color)

turtle.TurtleScreenBase.\_dot = \_dot

class RoundStar(Star):

def \_\_init\_\_(self,gravSys, name, m, x, v,

shapesize=1,shape=("blank","gray"),\*args,\*\*kw):

Star.\_\_init\_\_(self,gravSys, name, m, x, v,

shapesize,\*args,shape=shape,\*\*kw)

def init(self):

Star.init(self)

self.setheading=lambda angle:None

self.\_id=None

def \_drawturtle(self):

# 删除之前绘制的点

if self.\_id is not None:

self.screen.\_delete(self.\_id)

if not self.\_shown:return # 若已经隐藏

size=self.getsize()

if size>0.04:

px=3 if size>0.2 else 2

# 绘制形状

self.\_id=self.dot(max(size,px))

def dot(self,size,\*color):

if not color:

if isinstance(size, (str, tuple)):

color = self.\_colorstr(size)

size = self.\_pensize + max(self.\_pensize, 4)

else:

color = self.\_pencolor

if not size:

size = self.\_pensize + max(self.\_pensize, 4)

else:

if size is None:

size = self.\_pensize + max(self.\_pensize, 4)

color = self.\_colorstr(color)

item = self.screen.\_dot(self.\_position, size, color)

return item

def \_\_getstate\_\_(self):

return (self.gs,self.name,self.m,(self.x,self.y),(self.dx,self.dy),

self.\_size,self.\_shape,self.has\_orbit,self.parent,

self.keep\_on\_scr,self.rotation,self.sun)

class Sun(Star):

# 太阳不移动, 固定在引力系统的中心

def \_\_init\_\_(self,\*args,\*\*kw):

Star.\_\_init\_\_(self,\*args,\*\*kw)

self.keep\_on\_scr=True

def acc(self):

G = self.gs.G

for i in range(self.\_index+1,len(self.gs.planets)):

planet=self.gs.planets[i]

dx=planet.x-self.x

dy=planet.y-self.y

try:

b = G \* self.m / math.hypot(dx,dy)\*\*3

planet.ax-=b \* dx

planet.ay-=b \* dy

except ZeroDivisionError:pass

def step(self):

pass

def update(self):

self.setpos((self.x+self.gs.scr\_x)\*self.gs.scale,

(self.y+self.gs.scr\_y)\*self.gs.scale)

if self.rotation is not None:

self.left(self.rotation\*self.gs.dt)

if self.gs.show\_label:self.show\_label()

else:self.writer.clear()

#Star.update(self)

class SpaceCraft(Star):

flag=False;id=0

def \_\_init\_\_(self, gravSys, m, x, v,

shapesize=1,has\_orbit=True,

parent=None,keep\_on\_scr=False,rotation=None,name=None):

SpaceCraft.id+=1

Star.\_\_init\_\_(self, gravSys, name or 'ship #%d' % SpaceCraft.id,

m, x, v,

shapesize,has\_orbit,

parent,keep\_on\_scr,rotation,shape=())

self.init()

@classmethod

def \_init\_shape(cls):

if SpaceCraft.flag:return

shape = Shape("compound")

shape.addcomponent(((0,0),(3.333,-6),(0,-4.667)),'#b3b3b3')

shape.addcomponent(((0,0),(-3.333,-6),(0,-4.667)),'#666666')

scr.register\_shape('craft', shape)

def init\_shape(self):

self.\_init\_shape()

self.tilt(-90)

self.shape('craft')

self.pencolor('#333333')

self.shapesize(self.gs.scale)

def getsize(self):

return self.\_stretchfactor[0] \* PLANET\_SIZE / 2

def update(self):

self.setpos((self.x+self.gs.scr\_x)\*self.gs.scale,

(self.y+self.gs.scr\_y)\*self.gs.scale)

if self.rotation is not None:

self.left(self.rotation\*self.gs.dt)

else:

if self.gs.following:

if self.gs.following is self:

planet=self.parent

else:planet=self.parent

if planet is not None:dx=planet.dx;dy=planet.dy

else:dx=dy=0

else:dx=dy=0

angle = math.atan2(self.dy - dy,self.dx - dx) \* 180 / math.pi + 90

self.setheading(angle)

if self.gs.show\_label:self.show\_label()

else:self.writer.clear()

#if abs(self.x)>14000 or abs(self.y)>14000: # 移除超出范围的飞船

# self.gs.removed\_planets.append(self)

# self.gs.planets.remove(self)

# self.hideturtle()

def adjust\_speed(self,rate):

# 改变飞船速度，rate为新速度与原速度的比值

if self.parent:p\_dx,p\_dy = self.parent.dx, self.parent.dy # 获取父天体速度

else:p\_dx = p\_dy = 0

dx,dy = self.dx-p\_dx,self.dy-p\_dy # 计算自身相对于父天体的速度

new\_dx,new\_dy = dx\*rate, dy\*rate # 计算新速度

self.dx,self.dy = new\_dx+p\_dx,new\_dy+p\_dy # 更新自身的速度

d\_dx=new\_dx-dx # 计算前后速度差值

d\_dy=new\_dy-dy

for child in self.children: # 使子天体获得相同的速度变化量

child.dx+=d\_dx;child.dy+=d\_dy

def accelerate(self): # 飞船加速

self.adjust\_speed(1.01)

def slow\_down(self): # 飞船减速

self.adjust\_speed(1/1.01)

def turn(self,angle):

# angle为逆时针的角度，单位为度

angle = angle \* math.pi / 180 # 转换为弧度

if self.parent:p\_dx,p\_dy = self.parent.dx, self.parent.dy

else:p\_dx = p\_dy = 0

dx,dy = self.dx-p\_dx,self.dy-p\_dy

new\_dx,new\_dy = dx\*math.cos(angle) - dy\*math.sin(angle), \

dy\*math.cos(angle) + dx\*math.sin(angle)

self.dx,self.dy = new\_dx+p\_dx,new\_dy+p\_dy

d\_dx = new\_dx-dx;d\_dy = new\_dy-dy

for child in self.children:

child.dx+=d\_dx;child.dy+=d\_dy

def turn\_left(self):

self.turn(1) # 左(逆时针)转1°

def turn\_right(self):

self.turn(-1) # 右(顺时针)转1°

def \_\_getstate\_\_(self):

return (self.gs,self.m,(self.x,self.y),(self.dx,self.dy),

self.\_size,self.has\_orbit,self.parent,self.keep\_on\_scr,

self.rotation,self.name)

# 避免eval()函数产生安全漏洞

globals\_ = globals()

def \_safe\_eval(value):

if value in globals\_:

return globals\_[value] # 从全局变量中直接取出

else:

dct = {"\_\_builtins\_\_":{}, # 不允许调用内置函数

"TRUE":True,"FALSE":False} # 允许使用Excel中"TRUE"和"FALSE"的数据

return \_old\_eval(value,dct)

\_old\_eval=eval

eval=\_safe\_eval

# tkinter 界面部分

def find\_planet(gs,name):

for planet in gs.planets:

if planet.name==name:

return planet

return None

def clear\_old\_planets():

for planet in gs.planets: # 清除旧行星

planet.clear()

planet.hideturtle()

gs.planets.clear()

def load\_file(filename):

table = str.maketrans("（），。“”【】",'(),.""[]') # 用于将用户输入的中文标点转换为英文标点

following\_planet=None

try:

if filename.lower().endswith(".xlsx"):

df=pd.read\_excel(filename,dtype=str,sheet\_name="Celestial Bodies")

conf=pd.read\_excel(filename,dtype=str,sheet\_name="Config")

for col in conf: # 读取配置

if col != "following":

setattr(gs,col,eval(conf[col][0]))

else:

following\_planet=conf[col][0] # 跟踪的天体

elif filename.lower().endswith(".pkl"):

with open(filename,'rb') as f:

new\_gs=pickle.load(f)

gs.writer.clear() # 清除旧的提示文字

clear\_old\_planets()

gs.planets.extend(new\_gs.planets)

for planet in gs.planets:

planet.gs=gs

for attr in new\_gs.\_\_slots\_\_:

# 更新新的gs对象的属性

try:setattr(gs,attr,getattr(new\_gs,attr))

except AttributeError:pass

gs.init() # 初始化

return

else:

df=pd.read\_csv(filename,dtype=str)

except Exception as err:

msgbox.showinfo("","读取文件%s时出错。(%s:%s)"%(filename,type(err).\_\_name\_\_,str(err)))

return

clear\_old\_planets()

gs.scr\_x=gs.key\_x=0

gs.scr\_y=gs.key\_y=0

for i in range(len(df)):

try:

line = df.loc[i] # 取出一行数据,line为Series类型

kwargs={} # 初始化天体用的参数

# 取出天体的各个参数

for key in line.index:

value = line[key] # values为Series类型

if str(value)=="nan": # 空值

continue

value = value.translate(table) # 转为英文标点

if key in ("name","parent","sun"):

kwargs[key] = value # 直接用字符串作为值

else:kwargs[key] = eval(value)

if kwargs == {}:

continue # 忽略表格中的空行

# 取出天体所属的类

star\_type = kwargs["type"]

del kwargs["type"]

# 取出天体的卫星及所属恒星

if "parent" in kwargs:

parent=find\_planet(gs,kwargs["parent"])

if parent is None:

msgbox.showinfo("","没有找到天体 %s 的父天体，请修改您的天体列表。"%kwargs["name"])

del kwargs["parent"]

else:

kwargs["parent"]=parent

if "sun" in kwargs:

sun=find\_planet(gs,kwargs["sun"])

if sun is None:

msgbox.showinfo("","没有找到天体 %s 对应的恒星，请修改您的天体列表。"%kwargs["name"])

del kwargs["sun"]

else:

kwargs["sun"]=sun

star\_type(gs,\*\*kwargs) # 初始化天体实例

except Exception as err:

# 显示错误信息

msgbox.showinfo("","读取表格第 %d 行 %s 列出错，请修改您的天体列表。\n%s: %s" % (

i+1+1, key,type(err).\_\_name\_\_,str(err)))

gs.following=find\_planet(gs,following\_planet)

gs.zoom(1)

gs.init()

def save(filename):

df=pd.DataFrame(columns=["name","m","x","v","shapesize","shape",

"type","has\_orbit","parent","keep\_on\_scr",

"rotation","sun"])

for planet in gs.planets:

parent\_name=getattr(planet.parent,"name",None)

sun\_name=getattr(planet.sun,"name",None)

if isinstance(planet,SpaceCraft):sun\_name=None # 飞船不支持shape和sun参数

shape=planet.\_shape if not isinstance(planet,SpaceCraft) else None

data = [planet.name,planet.m,(planet.x,planet.y),(planet.dx,planet.dy),

planet.\_size,shape,type(planet).\_\_name\_\_,planet.has\_orbit,

parent\_name,planet.keep\_on\_scr,planet.rotation,sun\_name]

data = [str(d) if d is not None else "" for d in data]

df.loc[len(df)] = data

try:

if filename.lower().endswith(".xlsx"):

# 保存gs对象的配置

attr\_list=["t","dt","speed","scale","G","show\_tip","show\_orbit","show\_label"]

conf=pd.DataFrame(columns=attr\_list)

for attr in attr\_list:

conf[attr] = [str(getattr(gs,attr))]

conf["following"] = getattr(gs.following,"name",None) or ''

with pd.ExcelWriter(filename) as w:

df.to\_excel(w,sheet\_name="Celestial Bodies",index=False) # 不保存索引

conf.to\_excel(w,sheet\_name="Config",index=False)

elif filename.lower().endswith(".pkl"):

with open(filename,'wb') as f: # 保存pickle数据

pickle.dump(gs,f)

else:

df.to\_csv(filename,index=False)

except Exception as err:

msgbox.showinfo("错误",

"保存文件%s时出错，可能是您未关闭这个文件。(%s:%s)"%(filename,type(err).\_\_name\_\_,str(err)))

FILETYPES=[("Excel xlsx文件","\*.xlsx"),("CSV文件","\*.csv"),

("pickle文件(\*.pkl)","\*.pkl"),("所有文件","\*.\*")]

def open\_file(event=None):

if event is not None:master = scr.\_canvas.master # 如果用户通过按下快捷键保存

else:master = win

filename=filediag.askopenfilename(master=master,title='打开',filetypes=FILETYPES)

if not filename:return

load\_file(filename)

def save\_file(event=None):

if event is not None:master = scr.\_canvas.master # 如果用户通过按下快捷键保存

else:master = win

filename=filediag.asksaveasfilename(master=master,title='保存',

filetypes=FILETYPES,defaultextension='.xlsx')

if not filename:return

save(filename)

def switch\_tip\_mode(event=None):

# 切换显示提示信息的模式

gs.show\_tip = (gs.show\_tip+1)%3

def switch\_acceleration(event=None):

if gs.enable\_accelerate:

gs.speed=6

gs.dt \*= 0.004/0.00006

else:

gs.speed=340

gs.dt \*= 0.00006/0.004

gs.enable\_collision=False

gs.enable\_accelerate = not gs.enable\_accelerate

def switch\_collision(event=None):

if not gs.enable\_accelerate:

gs.enable\_collision = not gs.enable\_collision

else:gs.enable\_collision = False

def switch\_label(event=None):

gs.show\_label = not gs.show\_label

def show\_help():

msgbox.showinfo("帮助",\_\_doc\_\_,master=win)

def exit():

win.destroy();scr.bye() # 关闭窗口

def main():

global scr,gs,win

scr=Screen();scr.title("Python 天体引力模拟的探索")

scr.screensize(6000,6000)

try:

scr.\_canvas.master.state("zoomed")

except TclError:pass

scr.bgcolor("black")

scr.tracer(0,0)

win=tk.Tk()

win.title("控制")

win.geometry("210x85")

win.protocol("WM\_DELETE\_WINDOW",lambda:win.iconify()) # 关闭控制窗口时自动最小化

btns=tk.Frame(win)

btns.pack(side=tk.TOP)

file = "默认天体列表.xlsx"

if len(sys.argv)>1:

file=sys.argv[1]

gs = GravSys()

if os.path.isfile(file):

load\_file(file)

else:

sun = Sun(gs,"太阳",SUN\_MASS, (0,0), (0,0),

2.3,has\_orbit=False,shape=('yellow',))

mercury = Star(gs,"水星",MERCURY\_MASS, (60,0), (0,330),

0.5, shape=("#b3b3b3","#7f7f7f","#4d4d4d"))

venus = Star(gs,"金星",VENUS\_MASS, (-130,0), (0,-250),

0.7, shape=("gold","brown","gold4"))

earth = Star(gs,"地球",EARTH\_MASS, (260,0), (0,173),

0.8, shape=("blue","#00008b","blue"))

moon = Star(gs,"月球",MOON\_MASS, (269,0), (0,262),

0.5,shape=("#b3b3b3","#4d4d4d","gray30"),

has\_orbit=False, parent=earth)

mars = Star(gs,"火星",MARS\_MASS, (0,430), (-140, 0),

0.6, shape=("red","#8b0000","red"))

phobos = Star(gs,"火卫一",PHOBOS\_MASS, (0,438), (-167, 0),

0.1,shape=('circle',"orange"),

has\_orbit=False,parent=mars)

phobos.fillcolor("orange")

# 创建小行星

for i in range(10):

ast=RoundStar(gs,"小行星%d"%i, AST\_MASS,(0,0),(0,0),

0.05,has\_orbit=False)

ast.setheading(randrange(360))

ast.forward(randrange(700,800))

ast.x,ast.y=ast.pos()

v = ast.pos().rotate(90)

ast.dx,ast.dy=v[0]/7,v[1]/7

ast.pu()

ast.color("gray")

# 木星及卫星

jupiter = Star(gs, "木星", JUPITER\_MASS, (1100,0), (0, 86),

1.2,shape=("#ffd39b","#8b7355","#8b6508"))

mw1 = Star(gs,"木卫一", MOON\_MASS, (1125,0), (0,145),

0.05, shape=("circle","yellow"),

has\_orbit=False,parent=jupiter)

mw2 = Star(gs,"木卫二", MOON\_MASS, (1142,0), (0,134),

0.07,shape=("circle","#cd950c"),

has\_orbit=False,parent=jupiter)

# 土星

saturn = Star(gs,"土星",SATURN\_MASS, (2200,0), (0, 60),

1.0, shape=("#fff68f","#8b864e","#8b864e"))

# 天王星

uranus = Star(gs, "天王星", URANUS\_MASS, (0, 4300), (-43, 0),

0.8, shape=("#add8e6","blue","blue"))

# 海王星

neptune = Star(gs, "海王星", NEPTUNE\_MASS, (7500,0), (0, 34),

0.8, shape=("blue","#483d8b","#191970"))

hw2 = Star(gs, "海卫二", MOON\_MASS, (7600,0), (0, 48),

0.16, shape=("square","gray30"),

has\_orbit=False,parent=neptune)

# 绑定事件

cv=scr.getcanvas()

cv.bind\_all("<Key-Up>",gs.up)

cv.bind\_all("<Key-Down>",gs.down)

cv.bind\_all("<Key-Left>",gs.left)

cv.bind\_all("<Key-Right>",gs.right)

cv.bind\_all("<Key-equal>",gs.increase\_speed)

cv.bind\_all("<Key-minus>",gs.decrease\_speed)

cv.master.bind("<Key-Tab>",gs.switch) # 修复python 3.10+下无法绑定的bug

cv.bind\_all("<Key-Delete>",gs.del\_planet)

cv.master.bind("<Shift-Key-Tab>",gs.reverse\_switch)

cv.bind\_all("<Control-Key-equal>",lambda event:gs.zoom(4/3.0)) #Ctrl+"+"

cv.bind\_all("<Control-Key-minus>",lambda event:gs.zoom(3/4.0)) #Ctrl+"-"

cv.bind\_all("<Control-Key-d>",gs.clear\_spacecrafts)

cv.bind\_all("<Control-Key-o>",open\_file) # 打开和保存快捷键

cv.bind\_all("<Control-Key-s>",save\_file)

cv.bind\_all("<F3>",switch\_tip\_mode)

cv.bind\_all("<F4>",switch\_label)

cv.bind\_all("<F5>",switch\_acceleration)

cv.bind\_all("<F6>",switch\_collision)

cv.bind("<Button-1>",gs.\_onclick)

cv.bind("<B1-Motion>",gs.\_ondrag)

cv.bind("<B1-ButtonRelease>",gs.\_onrelease)

ttk.Button(btns,text="加速",command=gs.increase\_speed,width=5).grid(row=0,column=0)

ttk.Button(btns,text="减速",command=gs.decrease\_speed,width=5).grid(row=0,column=1)

ttk.Button(btns,text="隐藏/显示轨道",command=gs.switchpen,width=15).grid(row=0,column=2,columnspan=2)

ttk.Button(btns,text="放大",command=lambda:gs.zoom(4/3),width=5).grid(row=1,column=0)

ttk.Button(btns,text="缩小",command=lambda:gs.zoom(3/4),width=5).grid(row=1,column=1)

ttk.Button(btns,text="切换天体",command=gs.switch,width=13).grid(row=1,column=2,columnspan=2)

ttk.Button(btns,text="打开",command=open\_file,width=5).grid(row=2,column=0)

ttk.Button(btns,text="保存",command=save\_file,width=5).grid(row=2,column=1)

ttk.Button(btns,text="帮助",command=show\_help,width=5).grid(row=2,column=2)

ttk.Button(btns,text="退出",command=exit,width=5).grid(row=2,column=3)

cv.focus\_force()

globals().update(locals()) # 便于程序退出后, 在交互模式(>>> )中调试程序

gs.init()

try:gs.start()

except (Terminator,tk.TclError):

try:win.destroy()

except tk.TclError:pass

if \_\_name\_\_ == '\_\_main\_\_':main()

二、源文件“引力模拟 - 日地月.py”

import math

import tkinter as tk

import tkinter.ttk as ttk

from time import perf\_counter

from turtle import \*

G = 50 # 引力常量

d\_t = 0.000008 # 一轮计算经过的"时间", 经测试说明越小越精确

speed = 200 # 刷新一次屏幕之前执行计算的次数, 越大越快

lst=[]

class Star(Turtle):

def \_\_init\_\_(self, mass, position, velocity):

super().\_\_init\_\_()

self.shape("circle")

self.m=mass # 行星质量

self.x,self.y=position # 行星初始位置

self.dx,self.dy=velocity # 行星初始速度

self.ax=self.ay=0 # 行星加速度

lst.append(self)

self.penup()

self.setpos((self.x,self.y))

self.pendown()

def gravity(self):

# 计算行星自身受到的加速度，以及列表中位于自己之后的行星受到自己引力的加速度

for i in range(lst.index(self)+1, len(lst)):

p=lst[i] # 另一个行星

dx=p.x-self.x

dy=p.y-self.y

d = math.hypot(dx,dy) # 相当于 (dx\*\*2 + dy\*\*2)再开根号

f = G \* self.m \* p.m / d\*\*2

# 将力正交分解为水平、竖直方向并计算加速度

self.ax+=f / self.m \* dx / d

self.ay+=f / self.m \* dy / d

p.ax-=f / p.m \* dx / d

p.ay-=f / p.m \* dy / d

def step(self):

# 计算行星速度、位移

self.dx += d\_t\*self.ax

self.dy += d\_t\*self.ay

self.x+= d\_t\*self.dx

self.y+= d\_t\*self.dy

def update(self):

self.setpos((self.x,self.y))

class Sun(Star): # 太阳固定在中心, 继承自Star类

def gravity(self):

for i in range(lst.index(self)+1, len(lst)):

p=lst[i] # 另一个行星

dx=p.x-self.x

dy=p.y-self.y

d = math.hypot(dx,dy)

f = G \* self.m \* p.m / d\*\*2

# 将力正交分解为水平、竖直方向并计算加速度

p.ax-=f / p.m \* dx / d

p.ay-=f / p.m \* dy / d

def step(self):

pass

def clear\_screen(\*\_): # 清除行星轨迹

for p in lst:

p.clear()

def play():

global d\_t

d\_t=0.000008

sp["text"]="速度: %.2f" % (d\_t\*10\*\*6)

def stop():

global d\_t

d\_t=0

sp["text"]="速度: %.2f" % (d\_t\*10\*\*6)

def increase\_speed():

global d\_t

d\_t\*=1.2

sp["text"]="速度: %.2f" % (d\_t\*10\*\*6)

def decrease\_speed():

global d\_t

d\_t/=1.2

sp["text"]="速度: %.2f" % (d\_t\*10\*\*6)

def acc\_planet():

global t,S

t=1e-10; S=0 # 重置时间和扫过面积

earth.dx\*=1.1

earth.dy\*=1.1

moon.dx += earth.dx\*0.1

moon.dy += earth.dy\*0.1

def slow\_planet():

global t,S

t=1e-10; S=0 # 重置时间和扫过面积

earth.dx/=1.1

earth.dy/=1.1

moon.dx -= earth.dx\* 1/9

moon.dy -= earth.dy\* 1/9

def zoom\_earth\_orbit(scale): # 扩大/缩小地球轨道,scale为比例

x,y,dx,dy=earth.x,earth.y,earth.dx,earth.dy # 存储地球旧的位置、速度

earth.x\*=scale # 改变地球位置

earth.y\*=scale

# 改变地球速度

earth.dx/=math.sqrt(scale) # 根据GM/R² = mv²/R, 得速度v与√R成反比

earth.dy/=math.sqrt(scale)

moon.x += earth.x-x # 使月球和地球改变相同的位置、速度

moon.y += earth.y-y

moon.dx += earth.dx-dx

moon.dy += earth.dy-dy

def inc\_rd(): # 日地距离+

zoom\_earth\_orbit(1.1)

def dec\_rd(): # 日地距离-

zoom\_earth\_orbit(1/1.1)

def zoom\_moon\_orbit(scale): # 等比例改变月轨道

x,y,dx,dy=moon.x-earth.x,moon.y-earth.y,\

moon.dx-earth.dx,moon.dy-earth.dy # 存储月球相对地球位置、速度

moon.x = earth.x + x \* scale # 改变月球位置

moon.y = earth.y + y \* scale

moon.dx = earth.dx + dx/math.sqrt(scale) # 改变月球速度

moon.dy = earth.dy + dy/math.sqrt(scale)

def inc\_yd(): # 月地距离+

zoom\_moon\_orbit(1.1)

def dec\_yd(): # 月地距离-

zoom\_moon\_orbit(1/1.1)

def exit():

win.destroy();scr.bye() # 关闭窗口

# 创建tkinter 界面

win=tk.Tk()

win.title("控制")

win.geometry("320x150")

btns=tk.Frame(win)

btns.pack(side=tk.LEFT)

ttk.Button(btns,text="播放",command=play,width=5).grid(row=0,column=1)

ttk.Button(btns,text="暂停",command=stop,width=5).grid(row=0,column=2)

ttk.Button(btns,text="加快",command=increase\_speed,width=5).grid(row=0,column=3)

ttk.Button(btns,text="减慢",command=decrease\_speed,width=5).grid(row=0,column=4)

ttk.Button(btns,text="清屏",command=clear\_screen,width=5).grid(row=0,column=5)

ttk.Button(btns,text="行星加速",command=acc\_planet).grid(row=1,column=1,columnspan=2)

ttk.Button(btns,text="行星减速",command=slow\_planet).grid(row=1,column=3,columnspan=2)

ttk.Button(btns,text="日地距离+",command=inc\_rd).grid(row=2,column=1,columnspan=2)

ttk.Button(btns,text="日地距离-",command=dec\_rd).grid(row=2,column=3,columnspan=2)

ttk.Button(btns,text="月地距离+",command=inc\_yd).grid(row=3,column=1,columnspan=2)

ttk.Button(btns,text="月地距离-",command=dec\_yd).grid(row=3,column=3,columnspan=2)

ttk.Button(btns,text="退出",command=exit,width=5).grid(row=1,column=5)

tk.Label(btns,text="提示:月地距离过大或日地距离过小，\n会导致月地轨道不稳定。",

font=("微软雅黑",8)).grid(row=4,column=1,columnspan=4)

labels=tk.Frame(win)

labels.pack(side=tk.RIGHT,expand=True)

fps=tk.Label(labels,text="FPS: 0")

fps.pack(side=tk.TOP)

sp=tk.Label(labels,text="速度: %.2f" % (d\_t\*10\*\*6))

sp.pack(side=tk.TOP)

scr=Screen()

scr.title("Python 天体引力模拟的探索")

scr.bgcolor("black")

scr.tracer(0,0)

scr.onclick(clear\_screen) #点击屏幕清屏

w=Turtle() # w 用于输出文字

w.penup();w.hideturtle()

w.color("white") # 设置文字颜色为白色

sun = Sun(1e6, (0,0), (0,0))

sun.penup()

sun.color("yellow")

sun.shapesize(2)

earth = Star(1e4, (260,0), (0,400))

earth.color("blue")

earth.shapesize(0.7)

moon = Star(1,(269,0), (0,600))

moon.color("gray")

moon.shapesize(0.5)

t = 0 # 程序运行的总"时间"

time=perf\_counter() # 用于计算FPS

while True:

# 计算行星的位置

for i in range(speed):

t += d\_t

# 分别计算每个行星受到的加速度

for p in lst:

p.gravity()

# 根据计算的加速度, 求出速度和位移

for p in lst:

p.step()

for p in lst:

p.ax=p.ay=0 # 重置加速度

try:

# 刷新行星

for p in lst:

p.update()

update()

fps["text"]='FPS: %d' % int(1 / (perf\_counter()-time))

time=perf\_counter()

w.clear()

w.goto(scr.window\_width()//2-260,

scr.window\_height()//2-60)

w.write("日地距离: %.3f " % math.hypot(sun.x-earth.x,sun.y-earth.y) + "\n" + \

"月地距离: %.3f" % math.hypot(moon.x-earth.x,moon.y-earth.y)

, font=(None,12))

except (Terminator,tk.TclError):break # 如果窗口已关闭，忽略错误

三、源文件“引力模拟 - 开普勒第一定律.py”

import math

import tkinter as tk

import tkinter.ttk as ttk

from time import perf\_counter

from turtle import \*

G = 50 # 引力常量

d\_t = 0.000005 # 一轮计算经过的"时间", 经测试说明越小越精确

speed = 600 # 刷新一次屏幕之前执行计算的次数, 越大越快

lst=[]

class Star(Turtle):

def \_\_init\_\_(self, mass, position, velocity):

super().\_\_init\_\_()

self.shape("circle")

self.m=mass # 行星质量

self.x,self.y=position # 行星初始位置

self.dx,self.dy=velocity # 行星初始速度

self.ax=self.ay=0 # 行星加速度

lst.append(self)

self.penup()

self.setpos((self.x,self.y))

self.pendown()

def gravity(self):

# 计算行星自身受到的加速度，以及列表中位于自己之后的行星受到自己引力的加速度

for i in range(lst.index(self)+1, len(lst)):

p=lst[i] # 另一个行星

dx=p.x-self.x

dy=p.y-self.y

d = math.hypot(dx,dy) # 相当于 (dx\*\*2 + dy\*\*2)再开根号

f = G \* self.m \* p.m / d\*\*2

# 将力正交分解为水平、竖直方向并计算加速度

self.ax+=f / self.m \* dx / d

self.ay+=f / self.m \* dy / d

p.ax-=f / p.m \* dx / d

p.ay-=f / p.m \* dy / d

def step(self):

# 计算行星速度、位移

self.dx += d\_t\*self.ax

self.dy += d\_t\*self.ay

self.x+= d\_t\*self.dx

self.y+= d\_t\*self.dy

def update(self):

self.setpos((self.x,self.y))

class Sun(Star): # 太阳固定在中心, 继承自Star类

def gravity(self):

for i in range(lst.index(self)+1, len(lst)):

p=lst[i] # 另一个行星

dx=p.x-self.x

dy=p.y-self.y

d = math.hypot(dx,dy)

f = G \* self.m \* p.m / d\*\*2

# 将力正交分解为水平、竖直方向并计算加速度

p.ax-=f / p.m \* dx / d

p.ay-=f / p.m \* dy / d

def step(self):

pass

def get\_orbit\_shape(): # 获取椭圆轨道的长轴两端点坐标, 及焦点F2坐标

max\_x=max(x\_lst)

min\_x=min(x\_lst)

middle = (max\_x + min\_x)/2 # 椭圆中心X坐标, 焦点F1是太阳, X是F1F2的中点

return min\_x,max\_x,2 \* middle - 0

def clear\_screen(\*\_): # 清除行星轨迹

for p in lst:

p.clear()

def play():

global d\_t

d\_t=0.000005

sp["text"]="速度: %.2f" % (d\_t\*10\*\*6)

def stop():

global d\_t

d\_t=0

sp["text"]="速度: %.2f" % (d\_t\*10\*\*6)

def increase\_speed():

global d\_t

d\_t\*=1.2

sp["text"]="速度: %.2f" % (d\_t\*10\*\*6)

def decrease\_speed():

global d\_t

d\_t/=1.2

sp["text"]="速度: %.2f" % (d\_t\*10\*\*6)

def exit():

win.destroy();scr.bye() # 关闭窗口

# 创建tkinter 界面

win=tk.Tk()

win.title("控制")

win.geometry("250x80")

btns=tk.Frame(win)

btns.pack(side=tk.LEFT)

ttk.Button(btns,text="播放",command=play,width=5).grid(row=0,column=1)

ttk.Button(btns,text="暂停",command=stop,width=5).grid(row=0,column=2)

ttk.Button(btns,text="加快",command=increase\_speed,width=5).grid(row=0,column=3)

ttk.Button(btns,text="减慢",command=decrease\_speed,width=5).grid(row=0,column=4)

ttk.Button(btns,text="清除轨迹",command=clear\_screen).grid(row=1,column=2,columnspan=2)

ttk.Button(btns,text="退出",command=exit,width=5).grid(row=1,column=4)

labels=tk.Frame(win)

labels.pack(side=tk.RIGHT,expand=True)

fps=tk.Label(labels,text="FPS: 0")

fps.pack(side=tk.TOP)

sp=tk.Label(labels,text="速度: %.2f" % (d\_t\*10\*\*6))

sp.pack(side=tk.TOP)

scr=Screen()

scr.title("Python 天体引力模拟 - 开普勒第一定律演示")

scr.bgcolor("black")

scr.tracer(0,0)

scr.onclick(clear\_screen) #点击屏幕清屏

w=Turtle() # w 用于输出文字

w.penup();w.hideturtle()

w.color("white") # 设置文字颜色为白色

sun = Sun(1e6, (0,0), (0,0)) # 恒星

sun.penup()

sun.color("yellow")

sun.shapesize(2)

p = Star(1e4, (260,0), (0,300)) # 行星

p.color("blue")

p.shapesize(0.7)

t = 0 # 程序运行的总"时间"

x\_lst=[] # 行星与太阳距离的列表

time=perf\_counter() # 用于计算FPS

while True:

# 计算行星的位置

for i in range(speed):

t += d\_t

# 分别计算每个行星受到的加速度

for p in lst:

p.gravity()

# 根据计算的加速度, 求出速度和位移

for p in lst:

p.step()

for p in lst:

p.ax=p.ay=0 # 重置加速度

try:

# 刷新行星

for p in lst:

p.update()

fps["text"]='FPS: %d' % int(1 / (perf\_counter()-time))

update()

time=perf\_counter()

# 验证椭圆轨道

if t < 1:

x\_lst.append(p.x)

w.clear()

w.goto(scr.window\_width()//2-310,

scr.window\_height()//2-30)

w.write("生成结果中 ...", font=(None,12))

else:

x1,x2,x\_f2 = get\_orbit\_shape()

\_2a = x2-x1

d = math.hypot(p.x,p.y) # 行星到恒星距离

d2 = math.hypot(x\_f2-p.x,p.y) # 行星到焦点F2距离

# 输出文字

w.clear()

w.goto(scr.window\_width()//2-310,

scr.window\_height()//2-30)

w.write("PF1 + PF2= %f 2a= %f"%(d+d2, \_2a), font=(None,12)) # 如果PF1+PF2近似等于2a, 则验证通过

# 画上标签

w.goto(sun.x, sun.y)

w.write(" F1", font=(None,12))

w.pendown()

w.goto(p.x, p.y)

w.write(" P", font=(None,12))

w.goto(x\_f2, 0)

w.dot(5)

w.write("F2", font=(None,12))

w.penup()

# 长轴2a

w.goto(x1,0)

w.pendown()

w.goto(x2,0)

w.penup()

w.goto((1.2\*x1 + 0.8\*x2)/2,0)

w.write("2a", font=(None,12))

except (Terminator,tk.TclError):break # 如果窗口已关闭，忽略错误

四、源文件“引力模拟 - 开普勒第二定律.py”

import math

import tkinter as tk

import tkinter.ttk as ttk

import time

from turtle import \*

G = 50 # 引力常量

d\_t = 0.00001 # 一轮计算经过的"时间", 经测试说明越小越精确

speed = 900 # 刷新一次屏幕之前执行计算的次数, 越大越快

FPS=60 # 锁定FPS

lst=[]

class Star(Turtle):

def \_\_init\_\_(self, mass, position, velocity):

super().\_\_init\_\_()

self.shape("circle")

self.m=mass # 行星质量

self.x,self.y=position # 行星初始位置

self.dx,self.dy=velocity # 行星初始速度

self.ax=self.ay=0 # 行星加速度

lst.append(self)

self.penup()

self.setpos((self.x,self.y))

self.pendown()

def gravity(self):

# 计算行星自身受到的加速度，以及列表中位于自己之后的行星受到自己引力的加速度

for i in range(lst.index(self)+1, len(lst)):

p=lst[i] # 另一个行星

dx=p.x-self.x

dy=p.y-self.y

d = math.hypot(dx,dy) # 相当于 (dx\*\*2 + dy\*\*2)再开根号

f = G \* self.m \* p.m / d\*\*2

# 将力正交分解为水平、竖直方向并计算加速度

self.ax+=f / self.m \* dx / d

self.ay+=f / self.m \* dy / d

p.ax-=f / p.m \* dx / d

p.ay-=f / p.m \* dy / d

def step(self):

# 计算行星速度、位移

self.dx += d\_t\*self.ax

self.dy += d\_t\*self.ay

self.x+= d\_t\*self.dx

self.y+= d\_t\*self.dy

def update(self):

self.setpos((self.x,self.y))

def getOrbitSpeed(self,r):

# 获取某一半径的圆轨道上天体的速率

# 引力=向心力=m \* v\*\*2 / r

g = G \* self.m / r\*\*2

return math.sqrt(g \* r)

class Sun(Star): # 太阳固定在中心, 继承自Star类

def gravity(self):

for i in range(lst.index(self)+1, len(lst)):

p=lst[i] # 另一个行星

dx=p.x-self.x

dy=p.y-self.y

d = math.hypot(dx,dy)

f = G \* self.m \* p.m / d\*\*2

# 将力正交分解为水平、竖直方向并计算加速度

p.ax-=f / p.m \* dx / d

p.ay-=f / p.m \* dy / d

def step(self):

pass

def calc\_square(a,b,c): # 根据边长计算三角形面积

p = (a+b+c)/2

return math.sqrt(p\*(p-a)\*(p-b)\*(p-c))

def sleep(seconds):

# 重写time.sleep方法，避免其用于过短的时间（如小于0.017s）延迟过长

start=time.perf\_counter()

while time.perf\_counter()-start<seconds:

pass

def clear\_screen(\*\_): # 清除行星轨迹

for p in lst:

p.clear()

d.clear()

def play():

global d\_t

d\_t=0.00001

sp["text"]="速度: %.2f" % (d\_t\*10\*\*6)

def stop():

global d\_t

d\_t=0

sp["text"]="速度: %.2f" % (d\_t\*10\*\*6)

def increase\_speed():

global d\_t

d\_t\*=1.2

sp["text"]="速度: %.2f" % (d\_t\*10\*\*6)

def decrease\_speed():

global d\_t

d\_t/=1.2

sp["text"]="速度: %.2f" % (d\_t\*10\*\*6)

def acc\_planet():

global t,S

t=1e-10; S=0 # 重置时间和扫过面积

p.dx\*=1.1

p.dy\*=1.1

def slow\_planet():

global t,S

t=1e-10; S=0 # 重置时间和扫过面积

p.dx/=1.1

p.dy/=1.1

def exit():

win.destroy();scr.bye() # 关闭窗口

# 创建tkinter 界面

win=tk.Tk()

win.title("控制")

win.geometry("295x80")

btns=tk.Frame(win)

btns.pack(side=tk.LEFT)

ttk.Button(btns,text="播放",command=play,width=5).grid(row=0,column=1)

ttk.Button(btns,text="暂停",command=stop,width=5).grid(row=0,column=2)

ttk.Button(btns,text="加快",command=increase\_speed,width=5).grid(row=0,column=3)

ttk.Button(btns,text="减慢",command=decrease\_speed,width=5).grid(row=0,column=4)

ttk.Button(btns,text="清屏",command=clear\_screen,width=5).grid(row=0,column=5)

ttk.Button(btns,text="行星加速",command=acc\_planet).grid(row=1,column=1,columnspan=2)

ttk.Button(btns,text="行星减速",command=slow\_planet).grid(row=1,column=3,columnspan=2)

ttk.Button(btns,text="退出",command=exit,width=5).grid(row=1,column=5)

labels=tk.Frame(win)

labels.pack(side=tk.RIGHT,expand=True)

fps=tk.Label(labels,text="FPS: 0")

fps.pack(side=tk.TOP)

sp=tk.Label(labels,text="速度: %.2f" % (d\_t\*10\*\*6))

sp.pack(side=tk.TOP)

scr=Screen()

scr.title("Python 天体引力模拟的探索 - 开普勒第二定律演示")

scr.bgcolor("black")

scr.tracer(0,0)

scr.onclick(clear\_screen) #点击屏幕清屏

w=Turtle() # w 用于输出文字

w.penup();w.hideturtle()

w.color("white") # 设置文字颜色为白色

d=Turtle() # d 用于画出行星扫过的部分

d.penup();d.hideturtle()

d.color("gray")

d.begin\_fill()

sun = Sun(1e6, (0,0), (0,0))

sun.penup()

sun.color("yellow")

sun.shapesize(2)

p = Star(1e4, (260,0), (0,300))

p.color("blue")

p.shapesize(0.7)

count = 0 # 循环的次数

t0 = 0 # 程序运行的总"时间"

t=time.perf\_counter() # 用于计算FPS

S = 0 # 累计扫过面积

prev\_x, prev\_y = 260,0 # 行星的前一个坐标

while True:

# 计算行星的位置

for i in range(speed):

t0 += d\_t

# 分别计算每个行星受到的加速度

for p in lst:

p.gravity()

# 根据计算的加速度, 求出速度和位移

for p in lst:

p.step()

for p in lst:

p.ax=p.ay=0 # 重置加速度

try:

# 刷新行星

for p in lst:

p.update()

update()

# 锁定FPS

sleep(max(1 / FPS - (time.perf\_counter()-t),0))

fps["text"]='FPS: %d' % int(1 / (time.perf\_counter()-t))

t=time.perf\_counter()

# 画出行星与太阳连线扫过的部分

count += 1

if count%2==0:

d.goto(p.x,p.y)

d.goto(sun.x,sun.y)

d.end\_fill()

d.begin\_fill()

# 每隔一定时间清除画出图形，并输出文字

if count%20==0 and d\_t!=0: # d\_t!=0：未暂停

clear\_screen()

# 输出文字

w.clear()

w.goto(scr.window\_width()//2-360,

scr.window\_height()//2-80)

w.write("时间间隔:%.4f"%t0+"\n行星扫过面积:%.4f"%S + \

"\n行星扫过面积÷时间 = %.4f"%(S/t0) + \

"\n提示：加速/减速行星，可计算不同轨道下的结果。", font=(None,12)) # 如果行星扫过面积÷时间 是一个定值, 则验证通过

t0=0

S=0

d.goto(p.x,p.y)

# 算出3条边

a = math.hypot(p.x-prev\_x, p.y-prev\_y)

b = math.hypot(p.x, p.y)

c = math.hypot(prev\_x, prev\_y)

S += calc\_square(a,b,c) # 累加轨道扫过的图形面积

prev\_x, prev\_y = p.x,p.y

except (Terminator,tk.TclError):break # 如果窗口已关闭，忽略错误

五、程序“引力模拟 – 开普勒第三定律.py”

import math

import tkinter as tk

import tkinter.ttk as ttk

from time import perf\_counter

from turtle import \*

G = 50 # 引力常量

d\_t = 0.00005 # 一轮计算经过的"时间", 经测试说明越小越精确

speed = 500 # 刷新一次屏幕之前执行计算的次数, 越大越快

lst=[]

class Star(Turtle):

def \_\_init\_\_(self, mass, position, velocity):

super().\_\_init\_\_()

self.shape("circle")

self.m=mass # 行星质量

self.x,self.y=position # 行星初始位置

self.dx,self.dy=velocity # 行星初始速度

self.ax=self.ay=0 # 行星加速度

lst.append(self)

self.penup()

self.setpos((self.x,self.y))

self.pendown()

def gravity(self):

# 计算行星自身受到的加速度，以及列表中位于自己之后的行星受到自己引力的加速度

for i in range(lst.index(self)+1, len(lst)):

p=lst[i] # 另一个行星

dx=p.x-self.x

dy=p.y-self.y

d = math.hypot(dx,dy) # 相当于 (dx\*\*2 + dy\*\*2)再开根号

f = G \* self.m \* p.m / d\*\*2

# 将力正交分解为水平、竖直方向并计算加速度

self.ax+=f / self.m \* dx / d

self.ay+=f / self.m \* dy / d

p.ax-=f / p.m \* dx / d

p.ay-=f / p.m \* dy / d

def step(self):

# 计算行星速度、位移

self.dx += d\_t\*self.ax

self.dy += d\_t\*self.ay

self.x+= d\_t\*self.dx

self.y+= d\_t\*self.dy

def update(self):

self.setpos((self.x,self.y))

def calc\_orbit(self,parent): # 计算轨道参数

# 计算相对位置矢量和相对速度矢量

dx, dy = self.x - parent.x, self.y - parent.y

vx, vy = self.dx - parent.dx, self.dy - parent.dy

r = math.hypot(dx, dy)

v = math.hypot(vx, vy)

mu = G \* (parent.m + self.m)

ecc\_vec = ((v \*\* 2 - mu / r) \* dx - (dx \* vx + dy \* vy) \* vx,

(v \*\* 2 - mu / r) \* dy - (dx \* vx + dy \* vy) \* vy)

ecc = math.hypot(ecc\_vec[0], ecc\_vec[1]) / mu

semimajor\_axis = 1 / (2 / r - v \*\* 2 / mu)

return ecc, semimajor\_axis

class Sun(Star): # 太阳固定在中心, 继承自Star类

def gravity(self):

for i in range(lst.index(self)+1, len(lst)):

p=lst[i] # 另一个行星

dx=p.x-self.x

dy=p.y-self.y

d = math.hypot(dx,dy)

f = G \* self.m \* p.m / d\*\*2

# 将力正交分解为水平、竖直方向并计算加速度

p.ax-=f / p.m \* dx / d

p.ay-=f / p.m \* dy / d

def step(self):

pass

def get\_orbit\_shape():

max\_d=max(d\_lst)

min\_d=min(d\_lst)

return (max\_d + min\_d)/2

def clear\_screen(\*\_): # 清除行星轨迹

for p in lst:

p.clear()

def play():

global d\_t

d\_t=0.00001

sp["text"]="速度: %.2f" % (d\_t\*10\*\*6)

def stop():

global d\_t

d\_t=0

sp["text"]="速度: %.2f" % (d\_t\*10\*\*6)

def increase\_speed():

global d\_t

d\_t\*=1.2

sp["text"]="速度: %.2f" % (d\_t\*10\*\*6)

def decrease\_speed():

global d\_t

d\_t/=1.2

sp["text"]="速度: %.2f" % (d\_t\*10\*\*6)

def acc\_planet():

global t,S

t=1e-10; S=0 # 重置时间和扫过面积

earth.dx\*=1.1

earth.dy\*=1.1

def slow\_planet():

global t,S

t=1e-10; S=0 # 重置时间和扫过面积

earth.dx/=1.1

earth.dy/=1.1

def exit():

win.destroy();scr.bye() # 关闭窗口

# 创建tkinter 界面

win=tk.Tk()

win.title("控制")

win.geometry("295x80")

btns=tk.Frame(win)

btns.pack(side=tk.LEFT)

ttk.Button(btns,text="播放",command=play,width=5).grid(row=0,column=1)

ttk.Button(btns,text="暂停",command=stop,width=5).grid(row=0,column=2)

ttk.Button(btns,text="加快",command=increase\_speed,width=5).grid(row=0,column=3)

ttk.Button(btns,text="减慢",command=decrease\_speed,width=5).grid(row=0,column=4)

ttk.Button(btns,text="清屏",command=clear\_screen,width=5).grid(row=0,column=5)

ttk.Button(btns,text="行星加速",command=acc\_planet).grid(row=1,column=1,columnspan=2)

ttk.Button(btns,text="行星减速",command=slow\_planet).grid(row=1,column=3,columnspan=2)

ttk.Button(btns,text="退出",command=exit,width=5).grid(row=1,column=5)

labels=tk.Frame(win)

labels.pack(side=tk.RIGHT,expand=True)

fps=tk.Label(labels,text="FPS: 0")

fps.pack(side=tk.TOP)

sp=tk.Label(labels,text="速度: %.2f" % (d\_t\*10\*\*6))

sp.pack(side=tk.TOP)

scr=Screen()

scr.title("Python 天体引力模拟的探索 - 开普勒第三定律演示")

scr.bgcolor("black")

scr.tracer(0,0)

scr.onclick(clear\_screen) #点击屏幕清屏

w=Turtle() # w 用于输出文字

w.penup();w.hideturtle()

w.color("white") # 设置文字颜色为白色

w.goto(scr.window\_width()//2-260,

scr.window\_height()//2-60)

w.write("生成结果中 ...", font=(None,12))

sun = Sun(1e6, (0,0), (0,0))

sun.penup()

sun.color("yellow")

sun.shapesize(2)

earth = Star(1e4, (260,0), (0,400))

earth.color("blue")

earth.shapesize(0.7)

time=perf\_counter() # 用于计算FPS

t = 0 # 程序运行的总"时间"

t\_start=0 # 连线的角度为0°时的"时间"

d\_lst=[] # 存储行星和太阳距离

a\_lst=[0,0] # 存储行星和太阳连线的角度

a=0; T=1

while True:

# 计算行星的位置

for i in range(speed):

t += d\_t

# 分别计算每个行星受到的加速度

for p in lst:

p.gravity()

# 根据计算的加速度, 求出速度和位移

for p in lst:

p.step()

for p in lst:

p.ax=p.ay=0 # 重置加速度

try:

# 刷新行星

for p in lst:

p.update()

update()

fps["text"]='FPS: %d' % int(1 / (perf\_counter()-time))

time=perf\_counter()

d\_lst.append(math.hypot(sun.x-earth.x,sun.y-earth.y))

a\_lst.append(math.atan2(sun.y-earth.y,earth.x-sun.x)) # atan2(y, x)

if a\_lst[-2] > 0 and a\_lst[-1]<=0: # 行星、太阳连线与水平方向夹角恰好经过0°

T = t - t\_start # 轨道周期

t\_start = t

a = get\_orbit\_shape()

# 清除列表

d\_lst.clear()

a\_lst=[0,0]

# 输出文字

ecc, semimajor\_axis=earth.calc\_orbit(sun)

tip=("a^3=%.4f" % a\*\*3 + "\nT^2=%.4f" % T\*\*2 + \

"\nk = %.4f" % (a\*\*3/T\*\*2))

tip+="\n轨道离心率: {:.4g}\t半长轴: {:.5g}".format(ecc,semimajor\_axis)

tip+="\n提示：加速/减速行星，可计算不同情况下的\n 轨道参数。"

w.clear()

w.goto(scr.window\_width()//2-340,

scr.window\_height()//2-110)

w.write(tip, font=(None,12))

except (Terminator,tk.TclError):break # 如果窗口已关闭，忽略错误

六、源文件“引力模拟 - 第一、二宇宙速度.py”

import math

import tkinter as tk

import tkinter.ttk as ttk

from time import perf\_counter

from turtle import \*

G = 50 # 引力常量

d\_t = 0.000002 # 一轮计算经过的"时间", 经测试说明越小越精确

speed = 500 # 刷新一次屏幕之前执行计算的次数, 越大越快

lst=[]

class Star(Turtle):

def \_\_init\_\_(self, mass, position, velocity):

super().\_\_init\_\_()

self.shape("circle")

self.m=mass # 行星质量

self.x,self.y=position # 行星初始位置

self.dx,self.dy=velocity # 行星初始速度

self.ax=self.ay=0 # 行星加速度

lst.append(self)

self.penup()

self.setpos((self.x,self.y))

self.pendown()

def gravity(self):

# 计算行星自身受到的加速度，以及列表中位于自己之后的行星受到自己引力的加速度

for i in range(lst.index(self)+1, len(lst)):

p=lst[i] # 另一个行星

dx=p.x-self.x

dy=p.y-self.y

d = math.hypot(dx,dy) # 相当于 (dx\*\*2 + dy\*\*2)再开根号

f = G \* self.m \* p.m / d\*\*2

# 将力正交分解为水平、竖直方向并计算加速度

self.ax+=f / self.m \* dx / d

self.ay+=f / self.m \* dy / d

p.ax-=f / p.m \* dx / d

p.ay-=f / p.m \* dy / d

def step(self):

# 计算行星速度、位移

self.dx += d\_t\*self.ax

self.dy += d\_t\*self.ay

self.x+= d\_t\*self.dx

self.y+= d\_t\*self.dy

def update(self):

self.setpos((self.x,self.y))

def getOrbitSpeed(self,r):

# 获取某一半径的圆轨道上天体的速率

# 引力=向心力=m \* v\*\*2 / r

g = G \* self.m / r\*\*2

return math.sqrt(g \* r)

class Earth(Star): # 地球固定在中心, 继承自Star类

def gravity(self):

for i in range(lst.index(self)+1, len(lst)):

p=lst[i] # 另一个行星

dx=p.x-self.x

dy=p.y-self.y

d = math.hypot(dx,dy)

f = G \* self.m \* p.m / d\*\*2

# 将力正交分解为水平、竖直方向并计算加速度

p.ax-=f / p.m \* dx / d

p.ay-=f / p.m \* dy / d

def step(self):

pass

def clear\_screen(\*\_): # 清除行星轨迹

for p in lst:

p.clear()

def play():

global d\_t

d\_t=0.000002

sp["text"]="速度: %.2f" % (d\_t\*10\*\*6)

def stop():

global d\_t

d\_t=0

sp["text"]="速度: %.2f" % (d\_t\*10\*\*6)

def increase\_speed():

global d\_t

d\_t\*=1.2

sp["text"]="速度: %.2f" % (d\_t\*10\*\*6)

def decrease\_speed():

global d\_t

d\_t/=1.2

sp["text"]="速度: %.2f" % (d\_t\*10\*\*6)

def acc\_planet():

global t,S

t=1e-10; S=0 # 重置时间和扫过面积

p.dx\*=1.1

p.dy\*=1.1

def slow\_planet():

global t,S

t=1e-10; S=0 # 重置时间和扫过面积

p.dx/=1.1

p.dy/=1.1

def exit():

win.destroy();scr.bye() # 关闭窗口

def reset(): # 重置各个天体

test.x,test.y,test.dx,test.dy = 20, 0, 0, earth.getOrbitSpeed(r)

test2.x,test2.y,test2.dx,test2.dy = 0, 20, -earth.getOrbitSpeed(r)\*1.3, 0

test3.x,test3.y,test3.dx,test3.dy = 0, -20, earth.getOrbitSpeed(r)\*math.sqrt(2), 0

scr.ontimer(clear\_screen, 50)

# 创建tkinter 界面

win=tk.Tk()

win.title("控制")

win.geometry("340x110")

btns=tk.Frame(win)

btns.pack(side=tk.LEFT)

ttk.Button(btns,text="播放",command=play,width=5).grid(row=0,column=1)

ttk.Button(btns,text="暂停",command=stop,width=5).grid(row=0,column=2)

ttk.Button(btns,text="加快",command=increase\_speed,width=5).grid(row=0,column=3)

ttk.Button(btns,text="减慢",command=decrease\_speed,width=5).grid(row=0,column=4)

ttk.Button(btns,text="清屏",command=clear\_screen,width=5).grid(row=0,column=5)

ttk.Button(btns,text="重来",command=reset,width=5).grid(row=0,column=6)

ttk.Button(btns,text="行星加速",command=acc\_planet).grid(row=1,column=1,columnspan=2)

ttk.Button(btns,text="行星减速",command=slow\_planet).grid(row=1,column=3,columnspan=2)

ttk.Button(btns,text="退出",command=exit,width=5).grid(row=1,column=5)

labels=tk.Frame(win)

labels.pack(side=tk.RIGHT,expand=True)

fps=tk.Label(labels,text="FPS: 0")

fps.pack(side=tk.TOP)

sp=tk.Label(labels,text="速度: %.2f" % (d\_t\*10\*\*6))

sp.pack(side=tk.TOP)

scr=Screen()

scr.title("Python 天体引力模拟的探索 - 第一、二宇宙速度演示")

scr.bgcolor("black")

scr.tracer(0,0)

scr.onclick(clear\_screen) #点击屏幕清屏

w=Turtle() # w 用于输出文字

w.penup();w.hideturtle()

w.color("white") # 设置文字颜色为白色

earth = Earth(1e6, (0,0), (0,0))

earth.penup()

earth.color("blue")

earth.shapesize(2)

# 测试第一、第二宇宙速度

r = 20

print('"地球"半径:', r)

print('"地球"的第一宇宙速度:', earth.getOrbitSpeed(r))

test = Star(1,(20,0), (0, earth.getOrbitSpeed(r))) # 检验第一宇宙速度

test.color("red")

test.shapesize(0.4)

print('"地球"的第二宇宙速度:', earth.getOrbitSpeed(r)\*math.sqrt(2)) # 第一宇宙速度的√2倍

test2 = Star(1, (0,20), (-earth.getOrbitSpeed(r)\*1.3, 0)) # 速度介于第一、第二宇宙速度之间

test2.color("gray")

test2.shapesize(0.5)

test3 = Star(1,(0, -20), (earth.getOrbitSpeed(r)\*math.sqrt(2), 0)) # 检验第二宇宙速度, 观察到test2的轨迹是抛物线

test3.color("purple")

test3.shapesize(0.4)

t = 0 # 程序运行的总"时间"

time=perf\_counter() # 用于计算FPS

while True:

# 计算行星的位置

for i in range(speed):

t += d\_t

# 分别计算每个行星受到的加速度

for p in lst:

p.gravity()

# 根据计算的加速度, 求出速度和位移

for p in lst:

p.step()

for p in lst:

p.ax=p.ay=0 # 重置加速度

try:

# 刷新行星

for p in lst:

p.update()

update()

fps["text"]='FPS: %d' % int(1 / (perf\_counter()-time))

time=perf\_counter()

# 绘制标签

w.clear()

w.goto(test.x,test.y)

w.write("v=v1", font=(None,12))

w.goto(test2.x,test2.y)

w.write("v1<v<v2", font=(None,12))

w.goto(test3.x,test3.y)

w.write("v=v2", font=(None,12))

except (Terminator,tk.TclError):break # 如果窗口已关闭，忽略错误

七、源文件“引力模拟 - 卫星变轨.py”

import math

import tkinter as tk

import tkinter.ttk as ttk

from time import perf\_counter

from turtle import \*

from random import randint

G = 50 # 引力常量

d\_t = 0.000005 # 一轮计算经过的"时间", 经测试说明越小越精确

speed = 200 # 刷新一次屏幕之前执行计算的次数, 越大越快

lst=[]

class Star(Turtle):

def \_\_init\_\_(self, mass, position, velocity):

super().\_\_init\_\_()

self.shape("circle")

self.m=mass # 行星质量

self.x,self.y=position # 行星初始位置

self.dx,self.dy=velocity # 行星初始速度

self.ax=self.ay=0 # 行星加速度

lst.append(self)

self.penup()

self.setpos((self.x,self.y))

self.pendown()

def gravity(self):

# 计算天体自身受到的加速度

p=earth # 本程序中，假设卫星只受到地球的引力

dx=p.x-self.x

dy=p.y-self.y

d = math.hypot(dx,dy) # 相当于 (dx\*\*2 + dy\*\*2)再开根号

f = G \* self.m \* p.m / d\*\*2

# 将力正交分解为水平、竖直方向并计算加速度

self.ax+=f / self.m \* dx / d

self.ay+=f / self.m \* dy / d

p.ax-=f / p.m \* dx / d

p.ay-=f / p.m \* dy / d

def step(self):

# 计算行星速度、位移

self.dx += d\_t\*self.ax

self.dy += d\_t\*self.ay

self.x+= d\_t\*self.dx

self.y+= d\_t\*self.dy

def update(self):

self.setpos((self.x,self.y))

def calc\_orbit(self,parent): # 计算轨道参数

# 计算相对位置矢量和相对速度矢量

dx, dy = self.x - parent.x, self.y - parent.y

vx, vy = self.dx - parent.dx, self.dy - parent.dy

r = math.hypot(dx, dy)

v = math.hypot(vx, vy)

mu = G \* (parent.m + self.m)

ecc\_vec = ((v \*\* 2 - mu / r) \* dx - (dx \* vx + dy \* vy) \* vx,

(v \*\* 2 - mu / r) \* dy - (dx \* vx + dy \* vy) \* vy)

ecc = math.hypot(ecc\_vec[0], ecc\_vec[1]) / mu

semimajor\_axis = 1 / (2 / r - v \*\* 2 / mu)

return ecc, semimajor\_axis

class Earth(Star): # 地球固定在中心, 继承自Star类

def gravity(self):

pass

def step(self):

pass

def clear\_screen(\*\_): # 清除行星轨迹,\*\_ 用于忽略传入的参数

for p in lst:

p.clear()

del lst[2:] # 清除所有备份天体

def get\_rand\_color(): # 返回随机的颜色

return (randint(64,192),

randint(64,192),

randint(64,192))

def play():

global d\_t

d\_t=0.000005

sp["text"]="速度: %.2f" % (d\_t\*10\*\*6)

def stop():

global d\_t

d\_t=0

sp["text"]="速度: %.2f" % (d\_t\*10\*\*6)

def increase\_speed():

global d\_t

d\_t\*=1.2

sp["text"]="速度: %.2f" % (d\_t\*10\*\*6)

def decrease\_speed():

global d\_t

d\_t/=1.2

sp["text"]="速度: %.2f" % (d\_t\*10\*\*6)

def acc\_satellite():

backup=Star(sat.m,(sat.x,sat.y),(sat.dx,sat.dy)) # 创建一个备份天体，用于显示卫星变轨前的运动

backup.color(tuple(int(v) for v in sat.pencolor()))

backup.hideturtle()

sat.dx\*=1.1

sat.dy\*=1.1

sat.pencolor(get\_rand\_color())

def slow\_satellite():

backup=Star(sat.m,(sat.x,sat.y),(sat.dx,sat.dy)) # 创建一个备份天体，用于显示卫星变轨前的运动

backup.color(tuple(int(v) for v in sat.pencolor()))

backup.hideturtle()

sat.dx/=1.1

sat.dy/=1.1

sat.pencolor(get\_rand\_color())

def exit():

win.destroy();scr.bye() # 关闭窗口

def reset(): # 重置各个天体

sat.x,sat.y,sat.dx,sat.dy = 100, 0, 0, math.sqrt(2)\*500

sat.color((128,128,128))

scr.ontimer(clear\_screen, 50) # 定时清除旧的轨迹

# 创建tkinter 界面

win=tk.Tk()

win.title("控制")

win.geometry("340x110")

btns=tk.Frame(win)

btns.pack(side=tk.LEFT)

ttk.Button(btns,text="播放",command=play,width=5).grid(row=0,column=1)

ttk.Button(btns,text="暂停",command=stop,width=5).grid(row=0,column=2)

ttk.Button(btns,text="加快",command=increase\_speed,width=5).grid(row=0,column=3)

ttk.Button(btns,text="减慢",command=decrease\_speed,width=5).grid(row=0,column=4)

ttk.Button(btns,text="清屏",command=clear\_screen,width=5).grid(row=0,column=5)

ttk.Button(btns,text="重来",command=reset,width=5).grid(row=0,column=6)

ttk.Button(btns,text="卫星加速",command=acc\_satellite).grid(row=1,column=1,columnspan=2)

ttk.Button(btns,text="卫星减速",command=slow\_satellite).grid(row=1,column=3,columnspan=2)

ttk.Button(btns,text="退出",command=exit,width=5).grid(row=1,column=5)

labels=tk.Frame(win)

labels.pack(side=tk.RIGHT,expand=True)

fps=tk.Label(labels,text="FPS: 0")

fps.pack(side=tk.TOP)

sp=tk.Label(labels,text="速度: %.2f" % (d\_t\*10\*\*6))

sp.pack(side=tk.TOP)

scr=Screen()

scr.title("Python 天体引力模拟的探索")

scr.bgcolor("black")

scr.tracer(0,0)

scr.colormode(255)

scr.onclick(clear\_screen) #点击屏幕清屏

w=Turtle() # w 用于输出文字

w.penup();w.hideturtle()

w.color("white") # 设置文字颜色为白色

earth = Earth(1e6, (0,0), (0,0))

earth.penup()

earth.color("blue")

earth.shapesize(2)

sat = Star(1, (100,0), (0,math.sqrt(2)\*500))

sat.color((128,128,128))

sat.shapesize(0.7)

t = 0 # 程序运行的总"时间"

time=perf\_counter() # 用于计算FPS

while True:

# 计算行星的位置

for i in range(speed):

t += d\_t

# 分别计算每个行星受到的加速度

for p in lst:

p.gravity()

# 根据计算的加速度, 求出速度和位移

for p in lst:

p.step()

for p in lst:

p.ax=p.ay=0 # 重置加速度

try:

# 刷新行星

for p in lst:

p.update()

update()

fps["text"]='FPS: %d' % int(1 / (perf\_counter()-time))

time=perf\_counter()

ecc, semimajor\_axis=sat.calc\_orbit(earth)

# math.hypot(x,y)相当于math.sqrt(x\*\*2+y\*\*2)

tip="地球、卫星距离: %.3f\n卫星速度: %.3f" % (

math.hypot(sat.x-earth.x,sat.y-earth.y), math.hypot(sat.dx,sat.dy))

tip+="\n轨道离心率: {:.4g}\t半长轴: {:.5g}".format(ecc,semimajor\_axis)

w.clear()

w.goto(scr.window\_width()//2-340,

scr.window\_height()//2-60)

w.write(tip, font=(None,12))

except (Terminator,tk.TclError):break # 如果窗口已关闭，忽略错误

八、源文件“solar\_system\_accelerate\_util.c”

#include <Python.h>

#include <math.h>

// 文档字符串

PyDoc\_STRVAR(module\_doc, u8"solar\_system\_accelerate\_util模块，使用了C语言代码提升程序的性能，用于实现精细度更高的模拟。");

PyDoc\_STRVAR(accelerate\_doc, u8"accelerate(steps,G,dt,lst,sun\_index)\n用法与solar\_system - accelerate.py中的\_acc\_numba()函数相同。");

// accelerate函数

static PyObject\* accelerate(PyObject\* self, PyObject\* args, PyObject\* kwargs) {

static char\* keywords[] = {"steps", "G", "dt", "lst", "sun\_index", NULL};

int steps, sun\_index;

double G, dt, dx, dy, b;

PyObject\* lst\_obj;

if (!PyArg\_ParseTupleAndKeywords(args, kwargs, "iddOi", keywords, &steps, &G, &dt, &lst\_obj, &sun\_index)) {

return NULL;

}

PyObject\* lst = PySequence\_Fast(lst\_obj, "expected a sequence");

if (lst == NULL) {

return NULL;

}

Py\_ssize\_t len = PySequence\_Fast\_GET\_SIZE(lst);

double\* lst\_data = (double\*)malloc(len \* sizeof(double));

if (lst\_data == NULL) {

Py\_DECREF(lst);

return PyErr\_NoMemory();

}

for (Py\_ssize\_t i = 0; i < len; ++i) {

PyObject\* item = PySequence\_Fast\_GET\_ITEM(lst, i);

lst\_data[i] = PyFloat\_AsDouble(item);

if (PyErr\_Occurred()) {

free(lst\_data);

Py\_DECREF(lst);

return NULL;

}

}

// 由Python的代码改编而来

for (int \_ = 0; \_ < steps; ++\_) {

int index = sun\_index \* 7;

int i, j;

for (i = 0; i < len; i += 7) {

for (j = i + 7; j < len; j += 7) {

dx = lst\_data[j + 1] - lst\_data[i + 1];

dy = lst\_data[j + 2] - lst\_data[i + 2];

if ( !(dx == 0 && dy == 0) ) {

b = G / pow(sqrt(dx \* dx + dy \* dy), 3);

if (sun\_index == -1 || i != index) {

lst\_data[i + 5] += b \* dx \* lst\_data[j + 0];

lst\_data[i + 6] += b \* dy \* lst\_data[j + 0];

}

lst\_data[j + 5] -= b \* dx \* lst\_data[i + 0];

lst\_data[j + 6] -= b \* dy \* lst\_data[i + 0];

}

}

}

for (i = 0; i < len; i += 7) {

lst\_data[i + 3] += dt \* lst\_data[i + 5];

lst\_data[i + 4] += dt \* lst\_data[i + 6];

lst\_data[i + 5] = lst\_data[i + 6] = 0;

lst\_data[i + 1] += dt \* lst\_data[i + 3];

lst\_data[i + 2] += dt \* lst\_data[i + 4];

}

}

PyObject\* result = PyList\_New(len);

for (Py\_ssize\_t i = 0; i < len; ++i) {

PyObject\* value = PyFloat\_FromDouble(lst\_data[i]);

PyList\_SET\_ITEM(result, i, value);

}

free(lst\_data);

Py\_DECREF(lst);

return result;

}

static PyMethodDef module\_functions[] = {

{"accelerate", (PyCFunction)accelerate, METH\_VARARGS | METH\_KEYWORDS, accelerate\_doc},

{NULL, NULL, 0, NULL} /\* marks end of array \*/

};

int exec\_mod(PyObject \*module) {

PyModule\_AddFunctions(module, module\_functions);

// 加入版本信息

PyModule\_AddStringConstant(module, "\_\_version\_\_", "1.3.3");

return 0; /\* success \*/

}

static PyModuleDef\_Slot module\_slots[] = {

{ Py\_mod\_exec, exec\_mod },

{ 0, NULL }

};

static struct PyModuleDef module\_def = {

PyModuleDef\_HEAD\_INIT,

"solar\_system\_accelerate\_util",

module\_doc,

0, /\* m\_size \*/

NULL, /\* m\_methods \*/

module\_slots,

NULL, /\* m\_traverse \*/

NULL, /\* m\_clear \*/

NULL, /\* m\_free \*/

};

PyMODINIT\_FUNC PyInit\_solar\_system\_accelerate\_util() {

return PyModuleDef\_Init(&module\_def);

}}

九、源文件“setup.py”，用于编译“solar\_system\_accelerate\_util.c”，并生成Python扩展模块

import sys,os

from setuptools import setup,Extension

try:os.chdir(os.path.split(\_\_file\_\_)[0])

except:pass

# 操作说明：

# 在命令提示符中转到solar\_system\_accelerate\_util.c所在目录，

# 运行命令python setup.py bdist，即可编译该Python扩展模块。

# 再将生成的build\lib.winxx-cpythonxx目录中的.pyd文件

# 移动至"引力模拟.py"所在目录，即可使用。

setup(

name='solar-system-accelerate-util',

version='1.3.3',

ext\_modules=[Extension(

"solar\_system\_accelerate\_util",["solar\_system\_accelerate\_util.c"]

)],

)