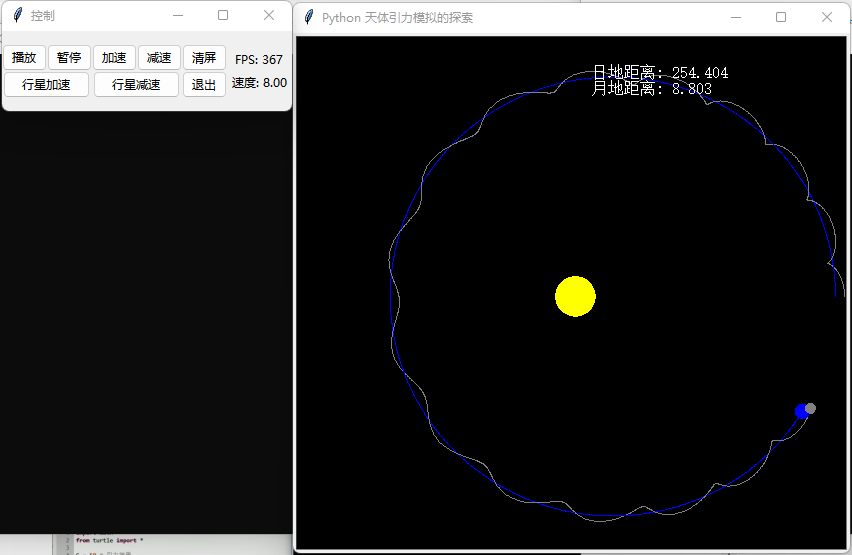
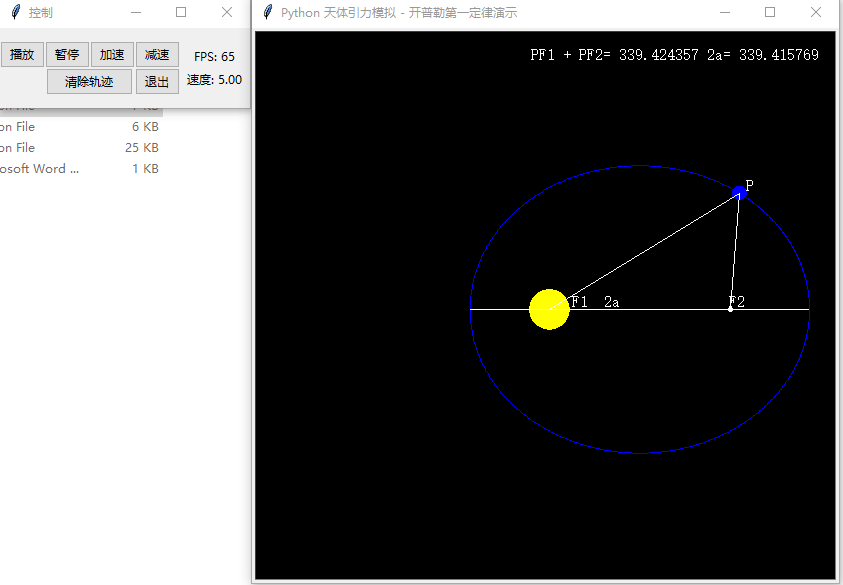
#### 作品效果图

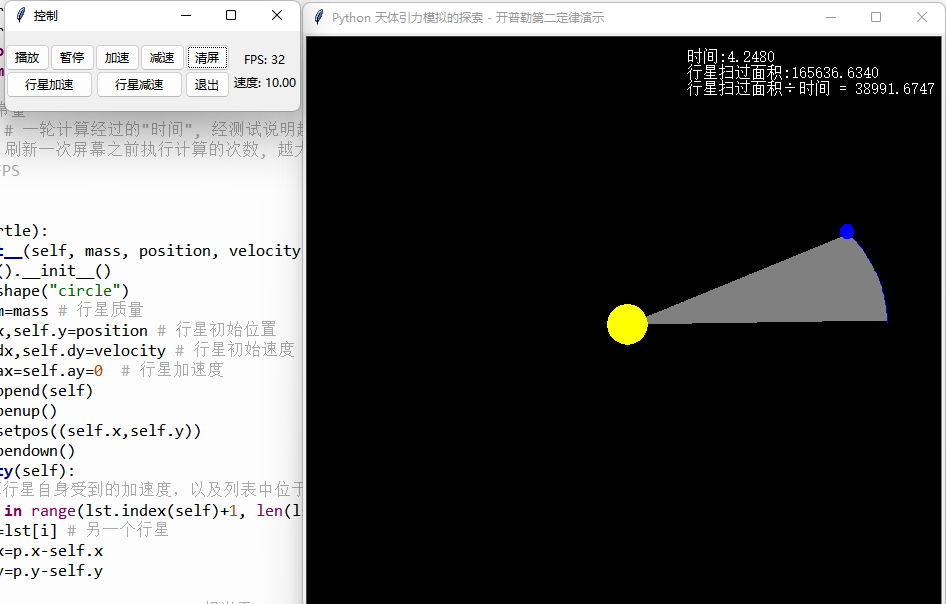
主程序运行截图：



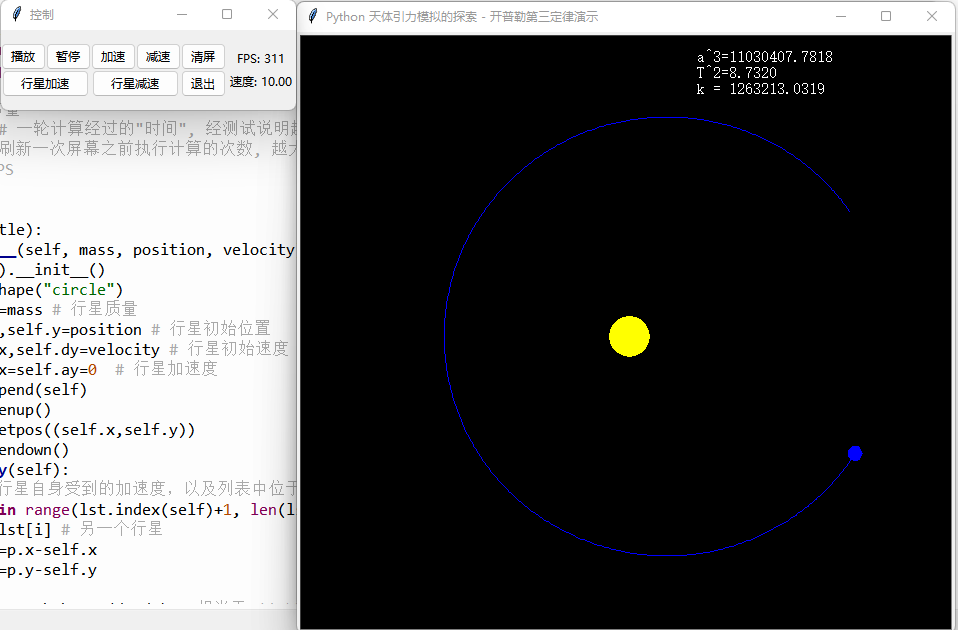
验证开普勒第一定律截图：



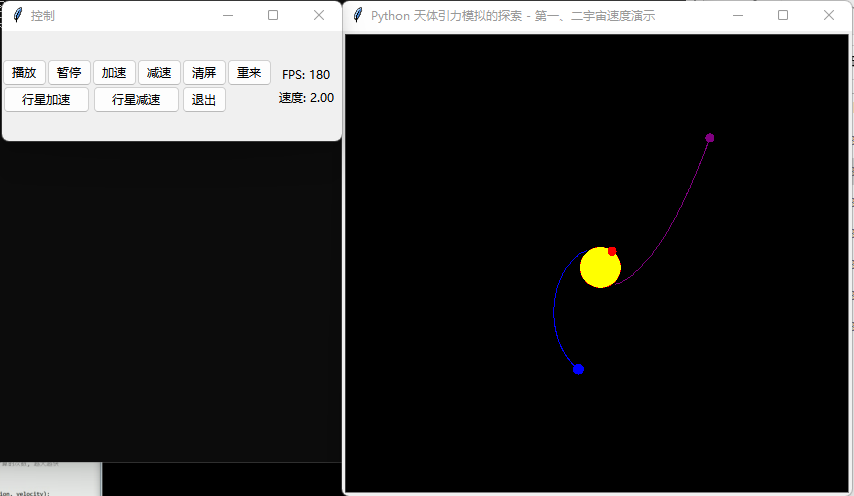
验证开普勒第二定律截图：



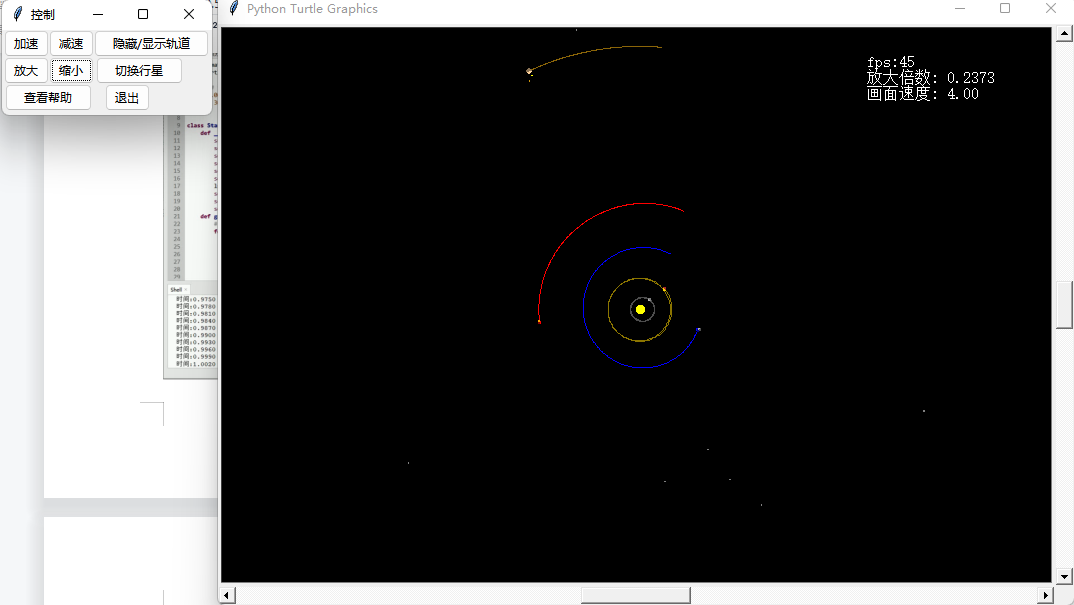
验证开普勒第三定律截图：



验证第一、二宇宙速度截图：



模拟太阳系截图



#### 二、作品主题:

作者学习了高中物理《必修二》后，自己虽然明白了万有引力定律与第一宇宙速度、天体圆轨道之间的关系。但万有引力定律与第二宇宙速度，以及与开普勒第一、第二定律的关系在物理书中没有直接的推理过程。为此，编写了本程序, 研究万有引力定律与天体轨道行为之间的关系，以及验证开普勒第一、第二定律。

作品Python天体引力模拟的探索，在物理课本的公式基础上加以创新而成，可以模拟出天体的椭圆轨道、以及太阳系中的太阳、行星及卫星。

作品中，中心的黄色天体为中心的恒星，蓝色天体为行星，灰色天体为行星的卫星。天体具有运行轨道。

代码中Star类的gravity方法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身引力的加速度。

通过修改程序中的G、d\_t、speed等常数，可以自定义引力模拟的效果。

1. **编程技巧说明:**

**1. 抽象与建模**

在真实的宇宙中，天体与其他各个天体之间都存在引力。

为简化研究，程序使用“降维”的思想，将真实的三维宇宙转换为二维的宇宙。

宇宙的本身属性引力常量G使用一个常量表示。每一个行星可以表示为它的质量、速度、x坐标、y坐标的属性。

**天体加速度的计算：**

假设有两个天体A，B，

则引力为，天体之间的距离为 ，天体A在x方向上的加速度为 ，在y方向上的加速度为。

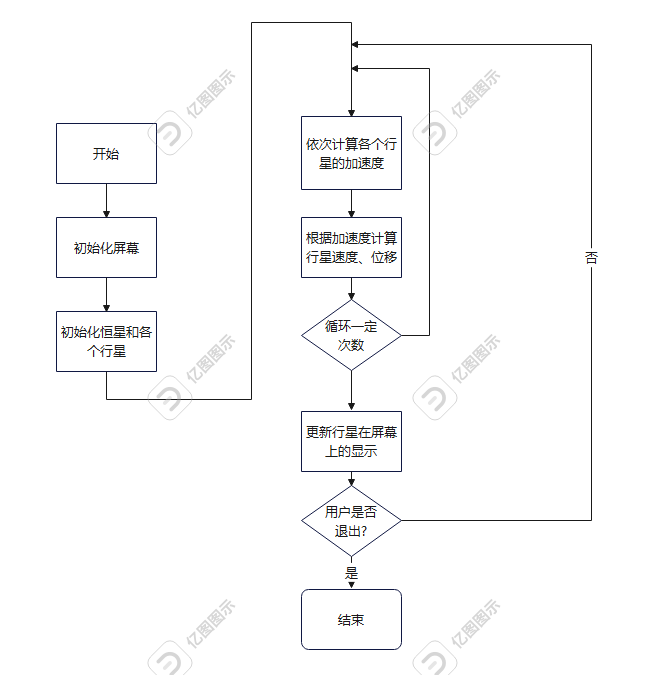
设程序单步经历的时间为，则新一轮天体A速度，位移。

程序这样重复这一个计算，不断地循环，就能逼真地模拟天体的运动过程。越小，模拟越精确。

**2.主程序部分**

程序使用Python内置turtle模块绘制图形；使用了面向对象编程的方法，类Star对应一个天体，并继承自内置turtle模块中的turtle类。

程序流程图如下：



程序首先初始化屏幕、恒星和各个行星，然后先计算行星受到重力的加速度，再计算速度和位移。代码如下：

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

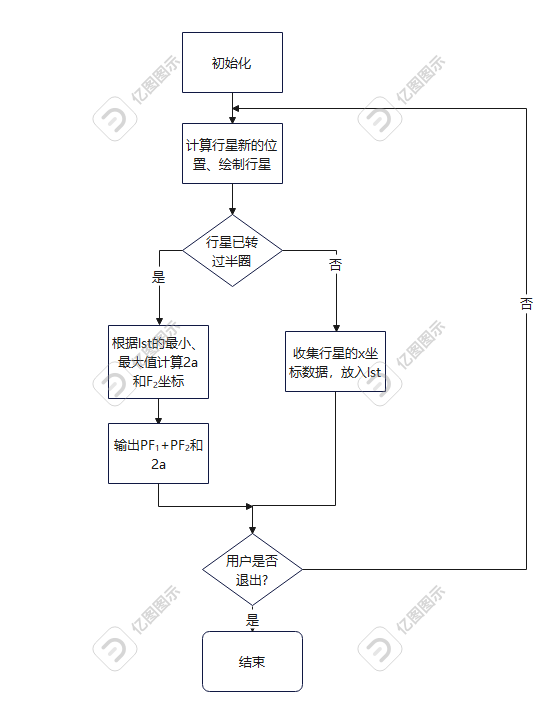
self.x+= d\_t\*self.dx

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

**3.验证开普勒第一定律部分**

开普勒行星运动第一定律，指的是每一行星沿各自的椭圆轨道环绕太阳，而太阳则处在椭圆的一个焦点上。

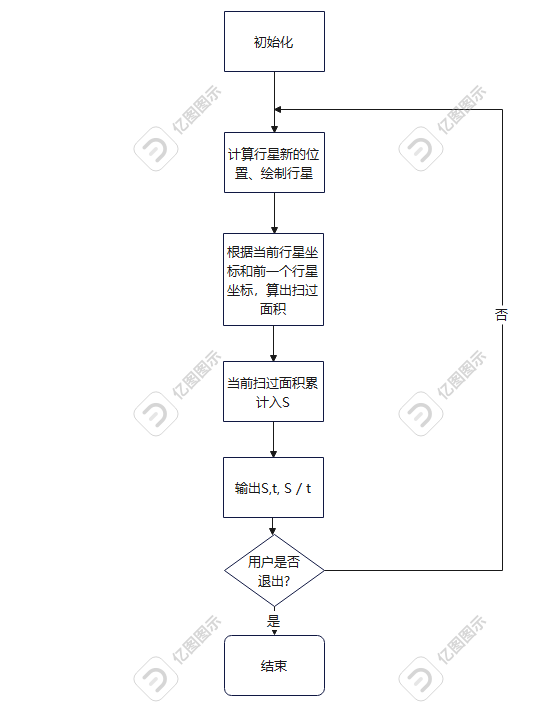
流程图如下：



椭圆上的任意一点到两个焦点的距离之和为一个定值，等于它的长轴。根据这个特性，设P为行星，F1、F2为焦点，太阳位于F1上，如果PF1+PF2近似等于长轴2a，则验证通过。

**4.验证开普勒第二定律部分**

流程图如下：

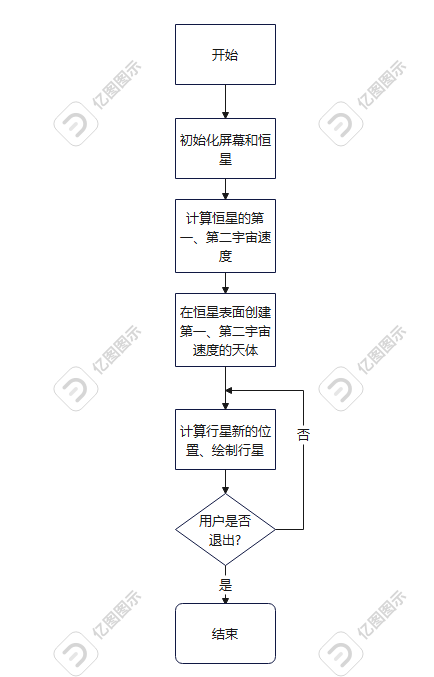


开普勒行星运动第二定律，指的是太阳系中太阳和运动中的行星的连线（矢径）在相等的时间内扫过相等的面积。

程序将行星轨道扫过的部分分割成一个个三角形, 分别计算每个三角形面积，再累加得到行星轨道扫过的面积S, 并除以经过的时间t。如果S/t是一个定值, 则验证通过。

**5.验证第一、第二宇宙速度部分**

流程图如下：



程序首先计算恒星的第一、第二宇宙速度，然后在恒星表面创建第一、第二宇宙速度的天体。观察现象。可以看到，第一宇宙速度的天体沿着圆形轨道环绕恒星运行，而第二宇宙速度的则沿着抛物线飞出。

1. **图形界面部分**

图形界面部分的程序，使用Python 3自带的模块tkinter设计，用于实现加速、减速、清屏/控制行星等功能。

图形界面的代码如下：

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 exit():

win.destroy();scr.bye() # 关闭窗口

# 创建tkinter 界面

win=tk.Tk()

win.title("控制")

win.geometry("290x80")

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="加速",command=increase\_speed,width=5).grid(row=0,column=3)

ttk.Button(btns,text="减速",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)

1. **程序“引力模拟太阳系”的设计**

程序“引力模拟太阳系”的设计，与前面的程序相比，增加了一个GraySys类，用于记录单个引力系统的信息，并增加了完善的图形界面功能，及读取Excel中天体数据的功能。

导入模块、定义行星质量等常量的代码：

from time import perf\_counter # 用于计算FPS

from random import randrange

import math,turtle,pickle,os,sys

from turtle import \*

import tkinter as tk

import tkinter.ttk as ttk

import tkinter.messagebox as msgbox

import pandas as pd

G = 8

PLANET\_SIZE=8 # 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

scr=None

创建GraySys类的代码，其中的start()方法实现前面所说的事件循环：

class GravSys:

# 引力系统

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

self.planets = []

self.removed\_planets=[]

self.t = 0

self.dt = 0.004 # 速度

#speed: 程序在绘制一帧之前执行计算的次数

self.speed=6

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=True;self.fps=20

self.writer=Turtle()

self.writer.hideturtle()

self.writer.penup()

self.writer.color("white")

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

while True:

# 计算行星的位置

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()

for p in self.planets:

p.ax=p.ay=0

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:

tip="""fps:%d

放大倍数: %.4f

画面速度: %.2f

""" % (self.fps, self.scale, self.dt\*10\*\*3)

if self.following:

tip+="""

正在跟随: %s

质量: %d""" % (self.following.name,self.following.m)

if getattr(self.following,'parent',None):

tip+="""

到%s距离: %d""" % (self.following.parent.name,

self.following.distance(self.following.parent))

else:

tip+='\n\n'

self.writer.clear()

self.writer.goto(

scr.window\_width()//2-200,scr.window\_height()//2-130

)

self.writer.write(

tip,

font = (None,12)

)

实现跟踪行星图形界面功能的代码：

def follow(self,planet):

if self.following:

self.following.onfollow(False)

self.following=planet

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

planet.onfollow(True)

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

def onclick(self,event=None): # 用于处理鼠标单击

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

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

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()

实现加速、减速功能：

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

self.dt+=0.0004

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

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)

实现缩放、移动视野功能：

def zoom(self,scale): # 缩放

self.scale \*= scale

self.\_update\_size()

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

self.clear\_removed\_planets()

def up(self,event=None):

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

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

self.clear\_removed\_planets()

def down(self,event=None):

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

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

self.clear\_removed\_planets()

def left(self,event=None):

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

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

self.clear\_removed\_planets()

def right(self,event=None):

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

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

self.clear\_removed\_planets()

实现隐藏/显示行星轨道：

def switchpen(self): # 隐藏/显示行星轨道

for planet in self.planets:

if not planet.has\_orbit:

continue

if planet.isdown():

planet.penup()

else:planet.pendown()

planet.clear()

按del键移除天体：

def remove(self,planet): # 移除天体

self.removed\_planets.append(planet)

self.planets.remove(planet)

planet.\_size = 1e-323 # 接近0

planet.hideturtle()

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)

切换行星：

def \_switch(self,dt):

# 切换到上/下一个行星

if not self.planets:return # 空列表

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

index=0

else:

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

if index < 0 or index>=len(self.planets):

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)

其他的类方法：

def clear\_scr(self):

for planet in self.planets:

planet.clear()

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=[]

这是实现Star等天体的类的代码。代码如下：

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.init\_shape()

self.penup()

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

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.resizemode("user")

self.setundobuffer(None)

self.children=[]

if parent:

parent.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

if self.has\_orbit:

self.pendown()

这里计算天体的引力、加速度、速度和位移，模拟天体的运动：

def acc(self):

# \*\* 计算行星的引力、加速度 \*\*

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

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

planet=self.gs.planets[i]

dx=planet.x-self.x

dy=planet.y-self.y

# 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

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

这是图形部分的代码，更新行星在屏幕上的显示：

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

# self.gs.remove(self) 清除已飞出太阳系的天体

Star类的其他方法：

def getsize(self): # 返回行星的显示大小

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

def distance(self,other): #返回两天体间距离

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

self.y-other.y)

def onfollow(self,arg): # arg:True或False

for p in self.children:

p.has\_orbit=arg

if arg and self.isdown():

p.pendown()

else:p.penup()

#self.keep\_on\_scr=arg

这一部分在前面的Star类的基础上进行了扩充，加入了行星一半亮一半暗的形状，使用\_init\_shape()和init\_shape()方法实现。

这一部分借鉴了Python 3中turtledemo.planet\_and\_moon模块的部分代码 (详细信息见参考与引用说明)：

@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()

update()

s.hideturtle()

def init\_shape(self):

# 初始化turtle的行星形状

# 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, [轨道颜色]) (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]

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]

if len(\_shape) >=2:

self.orbit\_color = \_shape[1]

self.color(\_shape[1])

self.shape(\_name)

self.shapesize(self.\_size)

self.pencolor(self.orbit\_color)

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):

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

planet=self.gs.planets[i]

dx=planet.x-self.x

dy=planet.y-self.y

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

planet.ax-=b \* dx

planet.ay-=b \* dy

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)

主程序的部分：

scr=Screen()

scr.screensize(6000,6000)

try:

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

except TclError:pass

scr.bgcolor("black")

scr.tracer(0,0)

# 创建tkinter 界面

def show\_help():

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

def exit():

win.destroy();scr.bye() # 关闭窗口

win=tk.Tk()

win.title("控制")

win.geometry("210x85")

btns=tk.Frame(win)

btns.pack(side=tk.TOP)

gs = GravSys()

调用pandas模块，读取excel表格中天体数据到程序中：

if os.path.isfile("天体列表.xlsx"):

lst=pd.read\_excel("天体列表.xlsx",dtype=str)

for i in range(len(lst)):

line = lst[i:i+1] # 取出一行数据

# 取出初始化类的各个参数

kwargs={}

for key in line:

values = line[key]

if len(values) == 0:

continue

if str(values[i])=="nan": # 空值

continue

kwargs[key]=eval(values[i])

del kwargs["type"]

type\_ = eval(line["type"][i]) # 取出天体所属的类

type\_(gs,\*\*kwargs) # 创建天体实例

创建各个天体：

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

0.1,shape=("circle","gray"),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

# 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)

创建控制按钮，绑定各个事件到类GraySys的方法：

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.bind\_all("<Key-Tab>",gs.switch)

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

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

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=3)

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="切换行星",command=gs.switch,width=11).grid(row=1,column=2,columnspan=2)

ttk.Button(btns,text="查看帮助",command=show\_help,width=11).grid(row=2,column=0,columnspan=2)

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

启动主事件循环：

gs.init()

try:gs.start()

except (Terminator,tk.TclError):

try:win.destroy()

except tk.TclError:pass

**四、参考与引用说明：**

程序中先计算再绘制的循环，以及G、d\_t常量的设计等，有一部分借鉴了标准版Python 3.7中的turtledemo.planet\_and\_moon模块的源码。

另外, 程序Star类的设计，部分借鉴了其中的面向对象编程思想。

**自己创新点：**

1. 计算引力、加速度、速度和位移的部分由本人原创，作者用自己的知识重写了其中的代码。
2. turtledemo.planet\_and\_moon模块中计算一次位移，就重新绘制一次行星，降低了性能。程序对这一缺陷加以改进，计算多次 (次数是speed) 位移，再重新绘制一次行星，提高性能。
3. 加入了原创的验证开普勒第一、第二定律，验证第一和第二宇宙速度部分。

如何看turtledemo.planet\_and\_moon模块的代码？

方法1.打开标准版Python 3.7的IDLE，“Help”菜单 –》 “turtle demo”，“Examples” –》“planet\_and\_moon”，即可。

方法2.打开标准Python 3.7的安装目录，找到文件“Lib\turtledemo\planet\_and\_moon.py”。