

中国科学技术大学计算机学院

程序设计 II 实验报告



实验题目: N 体问题
学生姓名: 艾语晨
学生学号: PB1800227
版本日期: Sep. 7th, 2020

计算机实验教学中心制

2019 年 09 月

N 体问题实验报告

艾语晨 PB18000227

2020 年 9 月 7 日

目录

1 背景介绍	2
2 实验目的	3
3 实验环境	4
3.1 开发环境	4
3.2 运行环境	4
3.3 工具	4
3.4 库	4
4 实验内容	6
4.1 原始代码	6
4.2 改编部分	6
4.2.1 设计概述	6
4.2.2 所用知识	6
4.3 代码解析	7
4.3.1 固定大小的窗口及自定义字体	7
4.3.2 作者页和登陆页	7
4.3.3 N 体问题程序	9
4.3.4 更改球体颜色	12
5 实验结果	13
6 总结与收获	15
7 参考资料及文献	16

第 1 章 背景介绍

在物理学中， n 体问题是预测一组由万有引力相互吸引的天体的运动轨迹的问题。一直以来，了解日月星辰的运动成为解决此问题的动机。在 20 世纪，了解球状星团的动力学成了最为重要的 n 体问题。广义相对论中的 n 体问题很难解决。

经典的物理问题可以非正式的表达为：

给定一组天体的准稳态轨道性质（瞬时位置，速度和时间），预测其相互作用力；因此，可以预测未来所有时间的真实轨道运动。¹

将各个星体抽象为质点，并根据经典力学的相关理论，若在三维空间中给定位置向量与各自的初速度向量，根据万有引力定律与 Newton 运动定律，可以计算出物体下一步的运动情况。为进一步简化问题，可以将时间分割为单位时间单元，以离散的时间变量代替连续的变量。

¹以上译自 Wikipedia

第 2 章 实验目的

成功安装 Python 3.8、pygame 并运行现有代码，并对其进行一定的改编，使之具有更多的功能，一个交互式界面

第 3 章 实验环境

3.1 开发环境

本实验所用程序设计语言为 python，版本为 python-3.8.5。所用包管理工具为 pip，版本为 pip-20.2.1。

3.2 运行环境

本实验运行在 Windows 10 专业版操作系统（Mac 虚拟机）上

3.3 工具

程序设计平台使用 Visual Studio Code，运行代码使用 vscode 的插件 code runner

3.4 库

所用到的库函数如下：

表 3.1: 实验所用到的库文件

MODULE	DESCRIPTION
pygame	GUI correlated
stdio.py	functions to read/write numbers and text from/to stdin and stdout
stddraw.py	functions to draw geometric shapes
stdaudio.py	functions to create, play, and manipulate sound
stdrandom.py	functions related to random numbers
stdarray.py	functions to create, read, and write 1D and 2D arrays
stdstats.py	functions to compute and plot statistics
color.py	data type for colors
picture.py	data type to process digital images
instream.py	data type to read numbers and text from files and URLs
outstream.py	data type to write numbers and text to files

第 4 章 实验内容

4.1 原始代码

原有代码的设计是典型的封装和实例化的过程，其物理思想就是利用了牛顿万有引力定律，在微元近似的条件下，认为 Δt 时间内速度不变，通过 $\vec{r}' = \vec{r} + \vec{v}\Delta t$ 计算出星体质点单位时间之后的坐标向量。人为建立一个坐标系并以向量来表示质点的参量，会简化计算。

4.2 改编部分

4.2.1 设计概述

把 4 个 N 体问题文件组织在一个统一的文件里面，提供一个类似游戏的小程序。新的程序包括一个作者页、登陆界面（需要密码）和 N 体问题程序唤起页。

4.2.2 所用知识

利用 Tkinter 库的相关应用，如下：

类/方法	作用	使用位置
<code>tkinter.messagebox</code>	弹出消息框	登陆输入密码位置
<code>tkinter.font</code>	自定义字体	N 体问题程序选择按钮
<code>tkinter.Label</code>	标签，显示文本或图像	密码输入提示文字、登陆页背景图片
<code>tkinter.Button</code>	按钮	结合事件响应弹出登陆密码输入框 和带命令行参数运行 N 体问题程序
<code>tkinter.Entry</code>	文本框	用于输入密码并显示为 *
<code>tkinter.Canvas</code>	画布	用于放置 Author Page

4.3 代码解析

4.3.1 固定大小的窗口及自定义字体

固定大小是为了和登陆界面的背景图片保持大小一致，而不是一开始展示的作者页；自定义字体改变了字体和字号，使得按钮上的文字更大更清楚。

字符串中的空格在 `LTEXListing` 中将显示为

`' '`

，故我用下划线代替

```
root = tkinter.Tk(className='N_Body_Problem')
root.geometry('499x737')

# define my font
myFontHelvetica = font.Font(family='Helvetica', size=18, weight='bold')
```

4.3.2 作者页和登陆页

作者页采用画布 `Canvas` 绘制，并调用 `bind()` 函数，在鼠标左键抬起的时候响应，切换到登陆界面。登陆页采用 `Label` 定义其背景图片，有一个按钮，其绑定响应为登陆 `Messangbox` 弹窗，只有当输入密码为 `'lapland'` 时，登陆通过，调用 `place_forget()` 隐藏登陆界面并创建 4 个按钮

作者页 `Author Page` 在 `Canvas` 上制作，定义 `self.canvas` 实例，调用参数函数定义大小与背景颜色；调用 `tkinter` 的图片引入函数，将背景图片作为实例变量并导入画布；利用继承的 `canvas` 方法创建文字。此部分代码如下所示：

```
def ShowAuthor(self):
    self.canvas = tkinter.Canvas(self)
    self.canvas["width"] = 500
    self.canvas["height"] = 480
    self.canvas["bg"] = 'Azure'

    # import logo
    self.logo = tkinter.PhotoImage(file='phoenix6.png')
    self.canvas.create_image(250, 300, image=self.logo)

    # caption
    self.canvas.create_text(
```

```

        250, 80, font=myFontHelvetica, text="N_Body_Problem")
self.canvas.create_text(
    250, 120, font=myFontHelvetica, text='by_OrigamiAyc')
self.canvas.pack()

# respond to Left-Click, changing to Author Page from Welcome Page
self.canvas.bind('<ButtonRelease-1>', self.create_background)

```

登陆页 Login Page 采取了与作者页不同的处理方式，即将 Label 组件作为背景嵌入到窗口的最底层，并在上层添加其他组件（这里为 Button）。作者页与登陆页之间的跳转采用事件响应的策略，通过响应鼠标左键的抬起来跳转（上面的 `self.canvas.bind()`）。按钮背景通过 HTML5 的十六进制 RGB 方式来赋值，按钮与背景标签均采用更灵活的 `place()` 函数来相对于窗口定位。登陆操作通过弹窗输入密码来完成，调用使用 `messagebox` 的类 `MyDialog` 实例化一个 password 弹窗实例。通过 `MyDialog` 的成员方法 `top()` 来等待弹窗结果，并调用方法 `get()` 来接收返回值。在一个简单的 `if` 语句判断之后，确定是否通过验证，并调用 `showinfo()` 方法来报告给用户。此部分代码如下：

```

def create_background(self, events):
    self.canvas.pack_forget()
    self.backgroundname = tkinter.PhotoImage(file="ThreeBodyWelcome.png")
    self.background_label = Label(self.master, image=self.backgroundname)
    self.background_label.place(relx=0, rely=0)

    # Create a Button to Login
    # self.ThreeBodyLogin = tkinter.PhotoImage('ThreeBodyLogin.jpg')
    self.login = tkinter.Button(
        self.master, text='Login', command=self.Create_Message_Box)
    self.login['bg'] = '#87CEFA' # DeepSkyBlue
    self.login.place(relx=0.46, rely=0.6)

def Create_Message_Box(self):
    password = MyDialog(self.master)
    self.login.wait_window(password.top)
    if password.get() == 'lapland':
        tkinter.messagebox.showinfo(
            'NBody', 'Welcome_to_the_world_of_NBody')
        self.create_buttons()
    else:
        tkinter.messagebox.showinfo('NBody', 'Incorrect_Password')

```

对话框 对话框调用的类如下，定义了一个弹出式对话框并返回接收到的字符串的功能。

```
class MyDialog:
    def __init__(self, root):
        self.top = tkinter.Toplevel(root)
        label = tkinter.Label(self.top, text='Your_Password')
        label.pack()
        self.entry = tkinter.Entry(self.top, show='*')
        self.entry.pack()
        self.entry.focus()
        button = tkinter.Button(self.top, text='Ok',
                                command=self.Ok)
        button.pack()

    def Ok(self):
        self.input = self.entry.get()
        self.top.destroy()

    def get(self):
        return self.input
```

4.3.3 N 体问题程序

在按钮页面上，四个按钮分别绑定在输入命令行参数运行 `Universe.py` 的方法上。这个方法实际上是对 `Universe.py` 里面的 `main` 函数加以改编得到的，即把命令提示符传参替换为方法调用传参。在命令行参数的传递方面，采用了三种思路：

1. 直接将命令行参数写在调用方法里面，作为其成员变量直接传入被调方法
2. 使用 `lambda` 表达式
3. 利用 `functools` 包里面的 `partial` 类，实例化一个中间参量作为关键字 `command` 的 `value`

此部分代码如下：

```
def create_buttons(self):
    self.background_label.place_forget()
    self.login.place_forget()

    self.TwoBodyTiny = tkinter.Button(self)
    self.TwoBodyTiny["text"] = "TwoBodyTiny"
    self.TwoBodyTiny["command"] = self.TwoBodyTiny
```

```
self.TwoBodyTiny['font'] = myFontHelvetica
self.TwoBodyTiny['bg'] = '#98FB98'
self.TwoBodyTiny['fg'] = '#800000'
self.TwoBodyTiny['height'] = 1
self.TwoBodyTiny['width'] = 15
self.TwoBodyTiny['command'] = self.execTwoTiny
self.TwoBodyTiny.pack(side=tkinter.TOP)

self.TwoBody = tkinter.Button(self)
self.TwoBody["text"] = "TwoBody"
self.TwoBody["command"] = self.TwoBody
self.TwoBody['font'] = myFontHelvetica
self.TwoBody['bg'] = '#87CEFA'
self.TwoBody['fg'] = '#8B008B'
self.TwoBody['width'] = 15
self.TwoBody['height'] = 1
self.TwoBody['command'] = self.execTwoBody
self.TwoBody.pack(side=tkinter.TOP)

self.ThreeBody = tkinter.Button(self)
self.ThreeBody["text"] = "ThreeBody"
self.ThreeBody["command"] = self.ThreeBody
self.ThreeBody['font'] = myFontHelvetica
self.ThreeBody['bg'] = '#DEB887'
self.ThreeBody['fg'] = '#191970'
self.ThreeBody['height'] = 1
self.ThreeBody['width'] = 15
self.ThreeBody['command'] = lambda: self.execNBody(
    '3body.txt', 20000, stddraw.MAGENTA)
self.ThreeBody.pack(side=tkinter.TOP)

self.FourBody = tkinter.Button(self)
self.FourBody["text"] = "FourBody"
self.FourBody["command"] = self.FourBody
self.FourBody['font'] = myFontHelvetica
self.FourBody['bg'] = '#FFDEAD'
self.FourBody['fg'] = '#228B22'
self.FourBody['height'] = 1
self.FourBody['width'] = 15
```

```
self.FourExec = partial(
    self.execNBody, '4body.txt', 20000, stddraw.ORANGE)
self.FourBody['command'] = self.FourExec
self.FourBody.pack(side=tkinter.TOP)

def execTwoTiny(self):
    filename = '2bodytiny.txt'
    dt = float(20000)
    PenColorTiny = stddraw.DARK_RED
    universe = Universe(filename)
    while True:
        universe.increaseTime(dt)
        stddraw.clear()
        universe.draw(PenColorTiny)
        stddraw.show(10)

def execTwoBody(self):
    filename = '2body.txt'
    dt = float(20000)
    PenColor = stddraw.DARK_GREEN
    universe = Universe(filename)
    while True:
        universe.increaseTime(dt)
        stddraw.clear()
        universe.draw(PenColor)
        stddraw.show(10)

def execNBody(self, fileName, calcFrequent, PenColor):
    self.filename = fileName
    self.dt = calcFrequent
    self.PenColor = PenColor
    self.universe = Universe(self.filename)
    while True:
        self.universe.increaseTime(self.dt)
        stddraw.clear()
        self.universe.draw(self.PenColor)
        stddraw.show(10)
```

4.3.4 更改球体颜色

调用 `std draw.py` 库的方法 `setPenColor()`，即可在 `Body.py` 中修改所有的球体的颜色（同时修改）。为了设定自定义颜色，需修改 `Body.py` 中 `Body.draw()` 方法，以及 `universe.py` 中 `draw()`，`main()` 的对应接口参数。对应代码如下：

```
# in body.py
def draw(self, ColorSet=std draw._DEFAULT_PEN_COLOR):
    std draw.setPenRadius(0.0125)
    std draw.setPenColor(ColorSet)
    std draw.point(self._r[0], self._r[1])

# in universe.py
def draw(self, ColorSet):
    for body in self._bodies:
        body.draw(ColorSet)
```

第 5 章 实验结果

下面放上运行截图，视频随压缩包附上

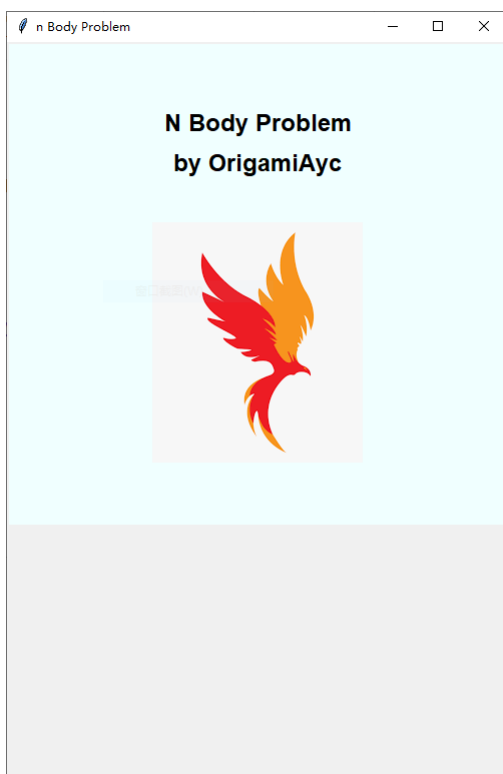


图 5.1: Author Page



图 5.2: Login Page

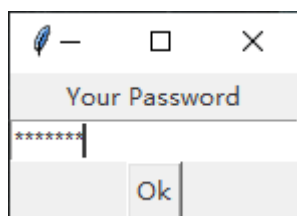


图 5.3: Password Entering

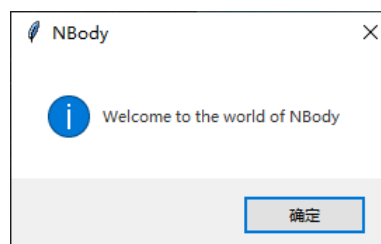


图 5.4: Password Verified



图 5.5: Buttons



图 5.6: ThreeBody Result

第 6 章 总结与收获

1. 对于 Tkinter 库的使用有了很多了解，可以自己设计并完成一个简单的有交互界面窗口的应用
2. 在复制粘贴代码的过程中对类的封装有了一些认识，可以通过封装类加上传递参数赋值的方式代替直接复制修改。可以增强代码可读性、增加简洁性、易于修改，类似于宏定义常参数的作用
3. python 入门，部分脱离 C 语言面向过程和涉及底层的影响（比如当我想开多线程的时候就不能直接系统调用）

第 7 章 参考资料及文献

1. 《21 天学通 python 第二版》
2. *Learing Tkinter*