*Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования*

|  |  |
| --- | --- |
|  | ***«Московский государственный технический университет  имени Н.Э. Баумана»***  ***(МГТУ им. Н.Э. Баумана)*** |

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*ФАКУЛЬТЕТ «ИНФОРМАТИКА И СИСТЕМЫ УПРАВЛЕНИЯ»*

*КАФЕДРА «ИУ-10»*

**Отчет по экзаменационной работе.**

**Дисциплина:** Объектно-ориентированное программирование

*Выполнили:*

Студенты 1 курса группы ИУ10-22

Локотков Кирилл Алексеевич

Щербин Никита Дмитриевич

*Проверили:* Буркацкий Кирилл Александрович

Шингарёв Александр Иванович

МОСКВА, 2022

**Оглавление:**

Цель работы:

* Разработать проект «Солнечная система». Смоделировать в плоском приближении и визуализировать движение планет Солнечной или любой другой системы.

Задачи:

* Пользовательский интерфейс должен позволять запускать и приостанавливать ход времени.
* Столкновение планет друг с другом и с «Солнцем» моделировать не требуется. Начальные данные о положении планет, их массах и начальных скоростях считываются из файла.
* По запросу пользователя данные о текущем положении, массах и скоростях планет должны сохраняться в файл.
* Возможность вывода траектории планет

Код программы:

* Модуль solar\_model.py - модуль, отвечающий за моделирование физических объектов.

# coding: utf-8

# license: GPLv3

GravConst = 6.67408E-11

"""Гравитационная постоянная Ньютона G"""

def calculate\_force(body, space\_objects):

*"""*

*Вычисляет силу, действующую на тело.*

*Параметры:*

*\*\*body\*\* — тело, для которого нужно вычислить дейстующую силу.*

*\*\*space\_objects\*\* — список объектов, которые воздействуют на тело.*

*"""*

body.Fx = body.Fy = 0

for obj in space\_objects:

if body == obj:

continue # тело не действует гравитационной силой на само себя!

dx = obj.x - body.x

dy = obj.y - body.y

r = (dx \*\* 2 + dy \*\* 2) \*\* 0.5

F = GravConst \* body.m \* obj.m / r \*\* 3

body.Fx += dx \* F

body.Fy += dy \* F

def move\_space\_object(body, dt):

*"""Перемещает тело в соответствии с действующей на него силой.*

*Параметры:*

*\*\*body\*\* — тело, которое нужно переместить.*

*\*\*dt\*\* — временной интервал.*

*"""*

ax = body.Fx / body.m

ay = body.Fy / body.m

body.Vx += ax \* dt

body.Vy += ay \* dt

body.prev\_x = body.x

body.prev\_y = body.y

body.x += body.Vx \* dt

body.y += body.Vy \* dt

def recalculate\_space\_objects\_positions(space\_objects, dt):

*"""Пересчитывает координаты объектов.*

*Параметры:*

*\*\*space\_objects\*\* — список оьъектов, для которых нужно пересчитать координаты.*

*\*\*dt\*\* — шаг по времени*

*"""*

for body in space\_objects:

calculate\_force(body, space\_objects)

for body in space\_objects:

move\_space\_object(body, dt)

if \_\_name\_\_ == "\_\_main\_\_":

print("This module is not for direct call!")

* Модуль solar\_input.py  - модуль, реализующий чтение и запись в конфигурационные файлы

# coding: utf-8

# license: GPLv3

from solar\_objects import Planet

def read\_space\_objects\_data\_from\_file(input\_filename):

*"""Cчитывает данные о космических объектах из файла, создаёт сами объекты*

*и вызывает создание их графических образов*

*Параметры:*

*\*\*input\_filename\*\* — имя входного файла*

*"""*

objects = []

with open(input\_filename) as input\_file:

for line in input\_file:

if len(line.strip()) == 0 or line[0] == '#':

continue # пустые строки и строки-комментарии пропускаем

planet = Planet()

parse\_planet\_parameters(line, planet)

objects.append(planet)

return objects

def parse\_planet\_parameters(line, planet):

*"""Считывает данные о планете из строки.*

*Предполагается такая строка:*

*Входная строка должна иметь слеюущий формат:*

*Planet <радиус в пикселах> <цвет> <масса> <x> <y> <Vx> <Vy>*

*Здесь (x, y) — координаты планеты, (Vx, Vy) — скорость.*

*Пример строки:*

*Planet 10 red 1000 1 2 3 4*

*Параметры:*

*\*\*line\*\* — строка с описание планеты.*

*\*\*planet\*\* — объект планеты.*

*"""*

args = line.split()

if len(args) != 8:

print("Ты ввел хуйню!")

print(line)

else:

planet.name = args[0]

planet.R = float(args[1])

planet.color = args[2]

planet.m = float(args[3])

planet.x = float(args[4])

planet.y = float(args[5])

planet.Vx = float(args[6])

planet.Vy = float(args[7])

def write\_space\_objects\_data\_to\_file(output\_filename, space\_objects: list[Planet]):

*"""Сохраняет данные о космических объектах в файл.*

*Строки должны иметь следующий формат:*

*Star <радиус в пикселах> <цвет> <масса> <x> <y> <Vx> <Vy>*

*Planet <радиус в пикселах> <цвет> <масса> <x> <y> <Vx> <Vy>*

*Параметры:*

*\*\*output\_filename\*\* — имя входного файла*

*\*\*space\_objects\*\* — список объектов планет и звёзд*

*"""*

with open(output\_filename, 'w') as out\_file:

for obj in space\_objects:

print(f"{obj.name} {obj.R} {obj.color} {obj.m} {obj.x} {obj.y} {obj.Vx} {obj.Vy}", file=out\_file)

if \_\_name\_\_ == "\_\_main\_\_":

print("This module is not for direct call!")

* Модуль solar\_vis.py -модуль, отвечающий за интерфейс пользователя

# coding: utf-8

# license: GPLv3

from PIL import ImageDraw

"""Модуль визуализации.

Нигде, кроме этого модуля, не используются экранные координаты объектов.

Функции, создающие графические объекты и перемещающие их на экране, принимают физические координаты

"""

header\_font = "Arial-16"

"""Шрифт в заголовке"""

window\_width = 800

"""Ширина окна"""

window\_height = 800

"""Высота окна"""

scale\_factor = None

"""Масштабирование экранных координат по отношению к физическим.

Тип: float

Мера: количество пикселей на один метр."""

def calculate\_scale\_factor(max\_distance):

*"""Вычисляет значение глобальной переменной \*\*scale\_factor\*\* по данной характерной длине"""*

global scale\_factor

scale\_factor = 0.4\*min(window\_height, window\_width)/max\_distance

print('Scale factor:', scale\_factor)

def scale\_x(x):

*"""Возвращает экранную \*\*x\*\* координату по \*\*x\*\* координате модели.*

*Принимает вещественное число, возвращает целое число.*

*В случае выхода \*\*x\*\* координаты за пределы экрана возвращает*

*координату, лежащую за пределами холста.*

*Параметры:*

*\*\*x\*\* — x-координата модели.*

*"""*

return int(x\*scale\_factor) + window\_width//2

def scale\_y(y):

*"""Возвращает экранную \*\*y\*\* координату по \*\*y\*\* координате модели.*

*Принимает вещественное число, возвращает целое число.*

*В случае выхода \*\*y\*\* координаты за пределы экрана возвращает*

*координату, лежащую за пределами холста.*

*Направление оси развёрнуто, чтобы у модели ось \*\*y\*\* смотрела вверх.*

*Параметры:*

*\*\*y\*\* — y-координата модели.*

*"""*

return int(y\*scale\_factor) + window\_height//2

def create\_planet\_image(space, planet):

*"""Создаёт отображаемый объект звезды.*

*Параметры:*

*\*\*space\*\* — холст для рисования.*

*\*\*star\*\* — объект звезды.*

*"""*

x = scale\_x(planet.x)

y = scale\_y(planet.y)

r = planet.R

planet.image = space.create\_oval([x - r, y - r], [x + r, y + r], fill=planet.color)

def update\_object\_position(space, body):

*"""Перемещает отображаемый объект на холсте.*

*Параметры:*

*\*\*space\*\* — холст для рисования.*

*\*\*body\*\* — тело, которое нужно переместить.*

*"""*

x = scale\_x(body.x)

y = scale\_y(body.y)

r = body.R

if x + r < 0 or x - r > window\_width or y + r < 0 or y - r > window\_height:

space.coords(body.image, window\_width + r, window\_height + r,

window\_width + 2\*r, window\_height + 2\*r) # положить за пределы окна

space.coords(body.image, x - r, y - r, x + r, y + r)

def draw\_track(img: ImageDraw.ImageDraw, body):

img.line([scale\_x(body.prev\_x), scale\_y(body.prev\_y), scale\_x(body.x), scale\_y(body.y)], fill=body.color, width=1)

if \_\_name\_\_ == "\_\_main\_\_":

print("This module is not for direct call!")

* Модуль solar\_objects.py- описание объектов

# coding: utf-8

# license: GPLv3

class Planet:

*"""Тип данных, описывающий планету.*

*Содержит массу, координаты, скорость планеты,*

*а также визуальный радиус планеты в пикселах и её цвет*

*"""*

name = ""

"""Название планеты"""

m = 0

"""Масса планеты"""

x = 0

"""Координата по оси \*\*x\*\*"""

y = 0

"""Координата по оси \*\*y\*\*"""

Vx = 0

"""Скорость по оси \*\*x\*\*"""

Vy = 0

"""Скорость по оси \*\*y\*\*"""

Fx = 0

"""Сила по оси \*\*x\*\*"""

Fy = 0

"""Сила по оси \*\*y\*\*"""

prev\_x = 0

"""Предыдущая координата по \*\*x\*\*"""

prev\_y = 0

"""Предыдущая координата по \*\*y\*\*"""

R = 5

"""Радиус планеты"""

color = "green"

"""Цвет планеты"""

image = None

"""Изображение планеты"""

* Модуль solar\_main.py - главный модуль

# coding: utf-8

# license: GPLv3

import tkinter

from PIL import Image, ImageTk, ImageDraw

from tkinter.filedialog import \*

from solar\_vis import \*

from solar\_model import \*

from solar\_input import \*

perform\_execution = False

"""Флаг цикличности выполнения расчёта"""

physical\_time = 0

"""Физическое время от начала расчёта.

Тип: float"""

displayed\_time = None

"""Отображаемое на экране время.

Тип: переменная tkinter"""

time\_step = None

"""Шаг по времени при моделировании.

Тип: float"""

space\_objects = []

"""Список космических объектов."""

call\_count = 0

black\_img = Image.new("RGBA", (window\_width, window\_height), (0, 0, 0, 255))

show\_tracks = False

def execution():

*"""Функция исполнения -- выполняется циклически, вызывая обработку всех небесных тел,*

*а также обновляя их положение на экране.*

*Цикличность выполнения зависит от значения глобальной переменной perform\_execution.*

*При perform\_execution == True функция запрашивает вызов самой себя по таймеру через от 1 мс до 100 мс.*

*"""*

global physical\_time

global displayed\_time

global track\_drawer

global black\_img

global tracks

global tracks\_img

global tracks\_container

global call\_count

global space

recalculate\_space\_objects\_positions(space\_objects, time\_step.get())

call\_count += 1

if call\_count % 2 == 0 and show\_tracks:

tracks = Image.blend(tracks, black\_img, 0.02)

track\_drawer = ImageDraw.Draw(tracks, "RGBA")

for body in space\_objects:

update\_object\_position(space, body)

if show\_tracks:

draw\_track(track\_drawer, body)

if show\_tracks:

space.delete(tracks\_container)

tracks\_img = ImageTk.PhotoImage(tracks)

tracks\_container = space.create\_image(0, 0, image=tracks\_img, anchor=tkinter.NW)

space.tag\_lower(tracks\_container, space.find\_all()[0])

physical\_time += time\_step.get()

displayed\_time.set("%.1f" % physical\_time + " seconds gone")

if perform\_execution:

space.after(101 - int(time\_speed.get()), execution)

def start\_execution():

*"""Обработчик события нажатия на кнопку Start.*

*Запускает циклическое исполнение функции execution.*

*"""*

global perform\_execution

perform\_execution = True

start\_button['text'] = "Stop"

start\_button['command'] = stop\_execution

execution()

print('Started execution...')

def stop\_execution():

*"""Обработчик события нажатия на кнопку Start.*

*Останавливает циклическое исполнение функции execution.*

*"""*

global perform\_execution

perform\_execution = False

start\_button['text'] = "Start"

start\_button['command'] = start\_execution

print('Paused execution.')

def toggle\_track():

global show\_tracks

global track\_button

global track\_drawer

global black\_img

global tracks

global tracks\_container

global space

if show\_tracks:

show\_tracks = False

track\_button['text'] = "Show track"

space.delete(tracks\_container)

tracks = Image.blend(tracks, black\_img, 1)

track\_drawer = ImageDraw.Draw(tracks, "RGBA")

else:

show\_tracks = True

track\_button['text'] = "Put away track"

def open\_file\_dialog():

*"""Открывает диалоговое окно выбора имени файла и вызывает*

*функцию считывания параметров системы небесных тел из данного файла.*

*Считанные объекты сохраняются в глобальный список space\_objects*

*"""*

global space\_objects

global perform\_execution

perform\_execution = False

for obj in space\_objects:

space.delete(obj.image) # удаление старых изображений планет

in\_filename = askopenfilename(filetypes=(("Text file", ".txt"),))

space\_objects = read\_space\_objects\_data\_from\_file(in\_filename)

max\_distance = max([max(abs(obj.x), abs(obj.y)) for obj in space\_objects])

calculate\_scale\_factor(max\_distance)

for obj in space\_objects:

if type(obj) is Planet:

create\_planet\_image(space, obj)

else:

raise AssertionError()

def save\_file\_dialog():

*"""Открывает диалоговое окно выбора имени файла и вызывает*

*функцию считывания параметров системы небесных тел из данного файла.*

*Считанные объекты сохраняются в глобальный список space\_objects*

*"""*

out\_filename = asksaveasfilename(filetypes=(("Text file", ".txt"),))

write\_space\_objects\_data\_to\_file(out\_filename, space\_objects)

def main():

*"""Главная функция главного модуля.*

*Создаёт объекты графического дизайна библиотеки tkinter: окно, холст, фрейм с кнопками, кнопки.*

*"""*

global physical\_time

global displayed\_time

global time\_step

global time\_speed

global space

global start\_button

global tracks

global tracks\_img

global tracks\_container

global tracks\_container

global track\_drawer

global track\_button

print('Modelling started!')

physical\_time = 0

root = tkinter.Tk()

# космическое пространство отображается на холсте типа Canvas

space = tkinter.Canvas(root, width=window\_width, height=window\_height, bg="black")

space.pack(side=tkinter.TOP)

tracks = Image.new("RGBA", (window\_width, window\_height))

track\_drawer = ImageDraw.Draw(tracks, "RGBA")

tracks\_img = ImageTk.PhotoImage(tracks)

tracks\_container = space.create\_image(0, 0, image=tracks\_img, anchor=tkinter.NW)

# нижняя панель с кнопками

frame = tkinter.Frame(root)

frame.pack(side=tkinter.BOTTOM)

start\_button = tkinter.Button(frame, text="Start", command=start\_execution, width=6)

start\_button.pack(side=tkinter.LEFT)

time\_step = tkinter.DoubleVar()

time\_step.set(1)

time\_step\_entry = tkinter.Entry(frame, textvariable=time\_step)

time\_step\_entry.pack(side=tkinter.LEFT)

time\_speed = tkinter.DoubleVar()

scale = tkinter.Scale(frame, variable=time\_speed, orient=tkinter.HORIZONTAL)

scale.pack(side=tkinter.LEFT)

load\_file\_button = tkinter.Button(frame, text="Open file", command=open\_file\_dialog)

load\_file\_button.pack(side=tkinter.LEFT)

save\_file\_button = tkinter.Button(frame, text="Save file", command=save\_file\_dialog)

save\_file\_button.pack(side=tkinter.LEFT)

track\_button = tkinter.Button(frame, text="Show track", command=toggle\_track, width=12)

track\_button.pack(side=tkinter.LEFT)

displayed\_time = tkinter.StringVar()

displayed\_time.set(str(physical\_time) + " seconds gone")

time\_label = tkinter.Label(frame, textvariable=displayed\_time, width=30)

time\_label.pack(side=tkinter.RIGHT)

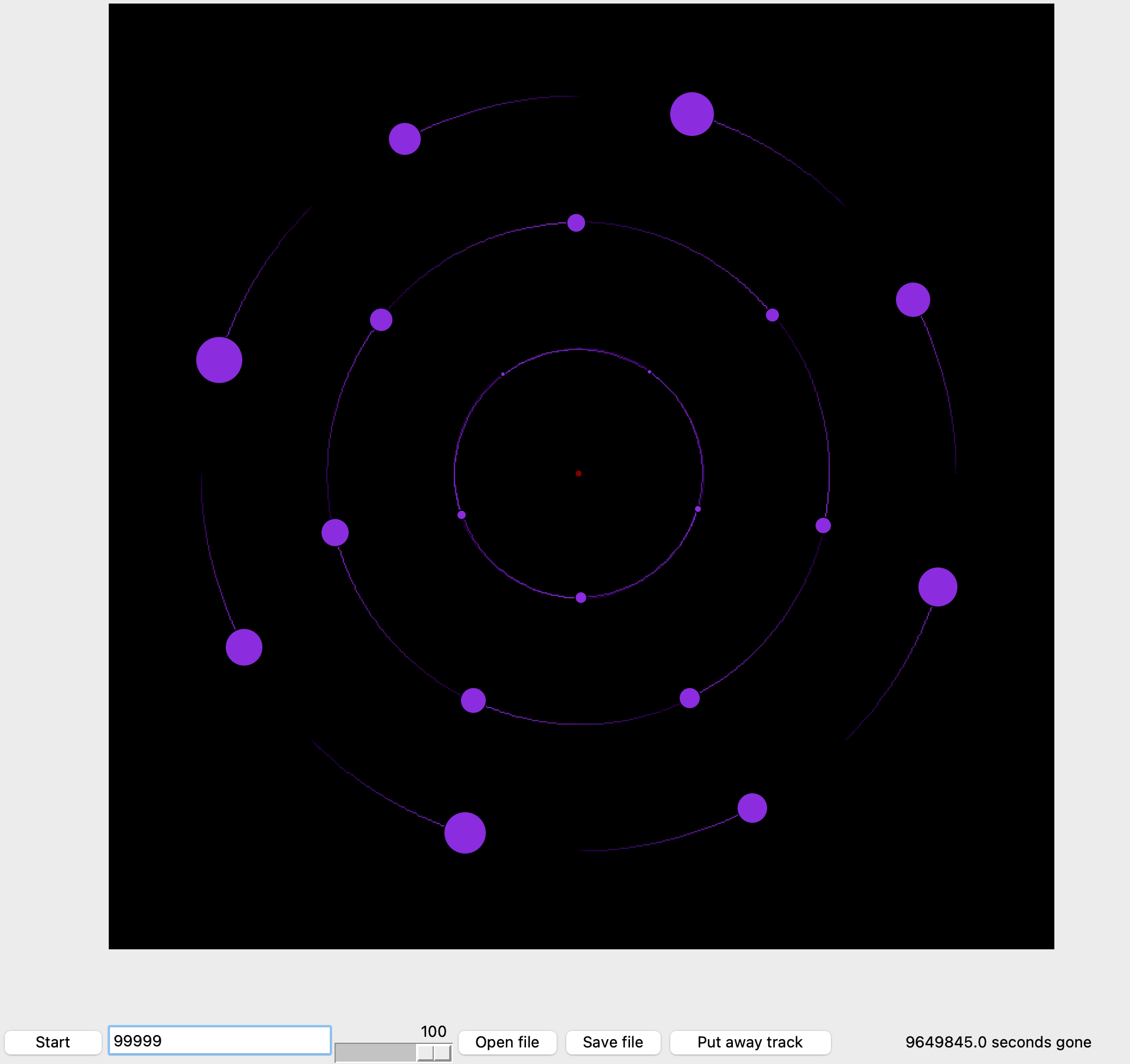
root.mainloop()

print('Modelling finished!')

if \_\_name\_\_ == "\_\_main\_\_":

main(

Тестирование:



Заключение:

Все поставленные задачи выполнены , в том числе и дополнительные задачи , по реализации 3-х систем.

Были закреплены основы объектно-ориентированного подхода, а также возможности библиотеки tkinter.