Московский государственный технический университет им. Н.Э. Баумана

Факультет «Информатика и системы управления»
Кафедра ИУ5 «Системы обработки информации и управления»

Курс «Парадигмы и конструкции языков программирования»

Отчет по ДЗ

Выполнил:
студент группы ИУ5-31Б
Шилина А.Ю.
Подпись и дата: 18.12.24

Проверил:
преподаватель каф. ИУ5
Гапанюк Е.Ю.
Подпись и дата:

ОГЛАВЛЕНИЕ

1. ОПИСАНИЕ ЗАДАНИЯ

Задание: разработать программу для симуляции полета ракеты с использованием графической библиотеки matplotlib. Программа должна учитывать различные параметры ракеты, такие как масса, сила тяги, коэффициент сопротивления и угол наклона. В результате работы программы отображается траектория полета ракеты с возможностью управления симуляцией через кнопки (пауза, ускорение, замедление).

2. ОПИСАНИЕ ПРОГРАММЫ

1. **Класс RocketSimulation:**
   Этот класс отвечает за управление симуляцией полета ракеты. Он инкапсулирует в себе:
   1. Инициализацию параметров ракеты (масса, сила тяги, угол наклона, коэффициент сопротивления и т.д.).
   2. Графическую отрисовку траектории полета ракеты.
   3. Обновление данных ракеты на каждом шаге симуляции.
   4. Управление симуляцией через кнопки: пауза, ускорение и замедление.
   5. Отображение информации о текущем времени, максимальной высоте и скорости ракеты.
2. **Методы:**
   1. **__init__(self, rocket)** – Конструктор класса, который инициализирует параметры ракеты, настраивает графику и добавляет кнопки для управления.
   2. **create_buttons(self)** – Метод для создания кнопок управления симуляцией.
   3. **toggle_pause(self, event)** – Включает и выключает паузу в симуляции.
   4. **speed_up(self, event)** – Ускоряет симуляцию.
   5. **slow_down(self, event)** – Замедляет симуляцию.
   6. **update_rocket_flight(self)** – Обновляет параметры ракеты на каждом шаге симуляции, вычисляя скорость и координаты.
   7. **simulate(self)** – Главный метод, который выполняет симуляцию, обновляя график и данные о ракете.
3. **Метод choose_rocket:**
   Данный метод позволяет пользователю выбрать модель ракеты (например, Falcon 9, Сатурн V, Терминатор-5 и другие). Пользователь может настроить параметры ракеты вручную, задавая такие значения как масса, сила тяги, угол наклона и другие.
4. **Графическое отображение:** Для отображения траектории полета используется библиотека matplotlib. График обновляется в реальном времени, и на нем отображается траектория полета ракеты. Также

выводится информация о текущем времени, максимальной высоте и скорости ракеты.

5. **Управление симуляцией:** В программе предусмотрены следующие кнопки управления:
    1. **Пауза** — ставит симуляцию на паузу или возобновляет ее.
    2. **Ускорение** — увеличивает скорость симуляции в 2 раза, максимум до 4x.
    3. **Замедление** — уменьшает скорость симуляции в 2 раза, минимум до 0.25x.
6. **Алгоритм расчета:** На каждом шаге симуляции рассчитывается сила тяги ракеты с учетом плотности воздуха (которая зависит от высоты), а также изменяются компоненты скорости по осям X и Y. Учитывается изменение массы ракеты в процессе расходования топлива.

3. ТЕКСТ ПРОГРАММЫ

```
4. import matplotlib.pyplot as plt
   import numpy as np
   from matplotlib.widgets import Button

   class RocketSimulation:
       def __init__(self, rocket):
           self.rocket = rocket
           self.x_coords = []
           self.y_coords = []
           self.max_height = 0
           self.max_speed = 0
           self.time_elapsed = 0
           self.running = True  # Управление паузой
           self.speed_factor = 1  # Управление скоростью
           self.finished = False  # Флаг завершения симуляции

           # Создаем график
           self.fig, self.ax = plt.subplots(figsize=(10, 6))
           self.line, = self.ax.plot([], [], label="Траектория полета
   ракеты", color="blue")
           self.ax.set_title("Траектория полета ракеты (в реальном
   времени)")
           self.ax.set_xlabel("Горизонтальная длина (м)")
           self.ax.set_ylabel("Высота (м)")
           self.ax.grid()
           self.ax.legend()

           # Счетчик информации
           self.info_box = self.fig.text(
               0.02, 0.95, "", fontsize=10, bbox=dict(facecolor="white",
   alpha=0.8)
           )

           # Добавляем кнопки
           self.create_buttons()

       def create_buttons(self):
           """Создает кнопки управления"""
           ax_pause = plt.axes([0.7, 0.01, 0.1, 0.05])  # Позиция кнопки
   паузы
```

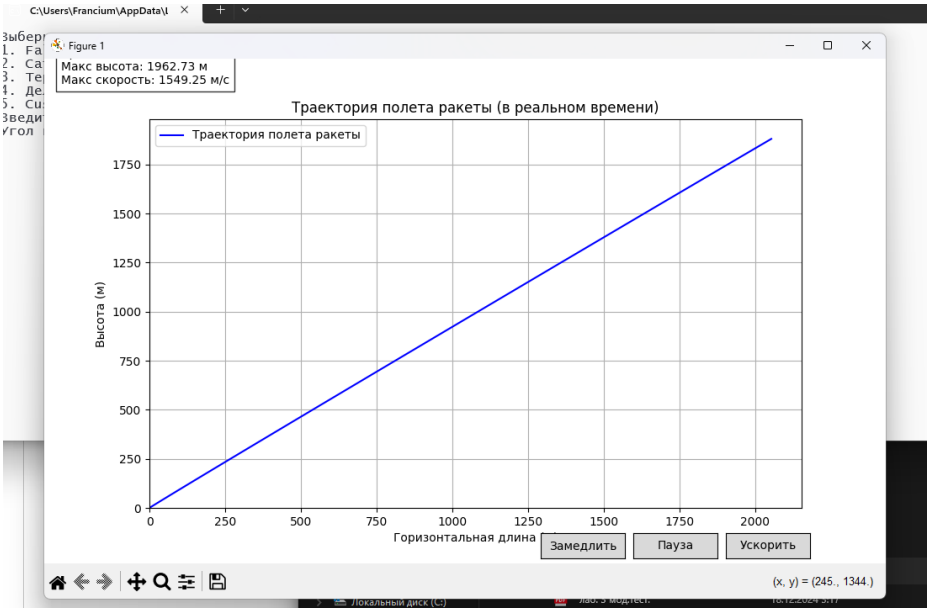
```
        ax_speed_up = plt.axes([0.81, 0.01, 0.1, 0.05])  # Позиция
кнопки ускорения
        ax_slow_down = plt.axes([0.59, 0.01, 0.1, 0.05])  # Позиция
кнопки замедления


        self.btn_pause = Button(ax_pause, "Пауза")
        self.btn_speed_up = Button(ax_speed_up, "Ускорить")
        self.btn_slow_down = Button(ax_slow_down, "Замедлить")

        # Привязываем функции к кнопкам
        self.btn_pause.on_clicked(self.toggle_pause)
        self.btn_speed_up.on_clicked(self.speed_up)
        self.btn_slow_down.on_clicked(self.slow_down)


    def toggle_pause(self, event):
        """Включение/выключение паузы"""
        self.running = not self.running
        self.btn_pause.label.set_text("Возобновить" if not self.running
else "Пауза")

    def speed_up(self, event):
        """Ускоряет симуляцию"""
        self.speed_factor = min(4, self.speed_factor * 2)  # Максимум
x4 скорости

    def slow_down(self, event):
        """Замедляет симуляцию"""
        self.speed_factor = max(0.25, self.speed_factor / 2)  # Минимум
x0.25 скорости


    def update_rocket_flight(self):
        """Обновляет параметры ракеты на каждом шаге симуляции"""
        g = 9.81  # Ускорение свободного падения
        air_density = 1.240 * np.exp(-0.00014 * self.rocket["height"])
# Плотность воздуха

        # Вычисляем силу тяги или сопротивления
        if self.rocket["full_weight"] <= self.rocket["weight"] -
self.rocket["fuel_weight"]:  # Если топливо закончилось
            force = -air_density * self.rocket["speed"] ** 2 / 2 *
self.rocket["resistance_coefficient"]
        else:  # Если двигатель работает
            force = self.rocket["engine_thrust"] - air_density *
self.rocket["speed"] ** 2 / 2 * self.rocket["resistance_coefficient"]

        # Обновляем скорости по X и Y
        self.rocket["speedX"] += force / self.rocket["full_weight"] *
np.cos(self.rocket["inclination_angle"]) * self.rocket["det_time"]
        self.rocket["speedY"] += (force / self.rocket["full_weight"] *
np.sin(self.rocket["inclination_angle"]) - g) * self.rocket["det_time"]

        # Если есть топливо, уменьшаем массу
        if self.rocket["full_weight"] > self.rocket["weight"] -
self.rocket["fuel_weight"]:
            self.rocket["full_weight"] -= self.rocket["fuel_weight"] /
self.rocket["engine_operating_time"] * self.rocket["det_time"]

        # Пересчитываем угол наклона ракеты
        self.rocket["inclination_angle"] =
np.arctan2(self.rocket["speedY"], self.rocket["speedX"])
```

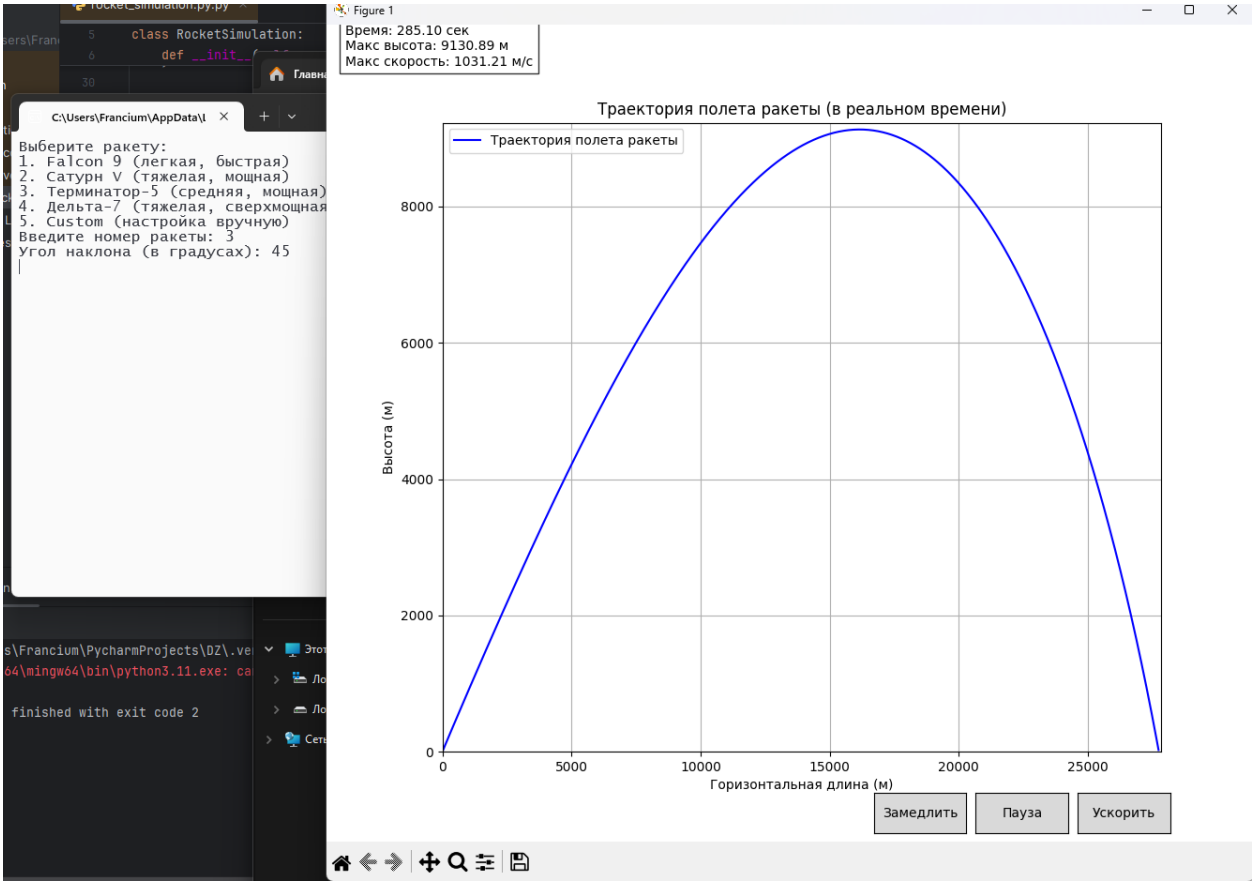
```
        # Общая скорость ракеты
        self.rocket["speed"] = np.sqrt(self.rocket["speedX"] ** 2 +
self.rocket["speedY"] ** 2)

        # Обновляем координаты
        self.rocket["height"] += self.rocket["speedY"] *
self.rocket["det_time"]
        self.rocket["length"] += self.rocket["speedX"] *
self.rocket["det_time"]

    def simulate(self):
        """Симулирует полет ракеты с реальной отрисовкой графика"""
        while self.rocket["height"] >= 0 and not self.finished:  # Пока
ракета не упала или не завершена
            if not self.running:  # Если пауза
                plt.pause(0.1)
                continue

            # Сохраняем текущие координаты
            self.x_coords.append(self.rocket["length"])
            self.y_coords.append(self.rocket["height"])

            # Обновляем состояние ракеты
            self.update_rocket_flight()

            # Обновляем максимальные значения
            self.max_height = max(self.max_height,
self.rocket["height"])
            self.max_speed = max(self.max_speed, self.rocket["speed"])
            self.time_elapsed += self.rocket["det_time"] *
self.speed_factor

            # Обновляем график
            self.line.set_data(self.x_coords, self.y_coords)
            self.ax.set_xlim(0, max(self.x_coords) + 100)
            self.ax.set_ylim(0, max(self.y_coords) + 100)

            # Обновляем информацию на счетчике
            self.info_box.set_text(
                f"Время: {self.time_elapsed:.2f} сек\n"
                f"Макс высота: {self.max_height:.2f} м\n"
                f"Макс скорость: {self.max_speed:.2f} м/с"
            )

            plt.pause(0.01 / self.speed_factor)  # Учитываем скорость
симуляции

        plt.ioff()  # Отключаем интерактивный режим
        plt.show()


def choose_rocket():
    """Меню для выбора ракеты"""
    print("Выберите ракету:")
    print("1. Falcon 9 (легкая, быстрая)")
    print("2. Сатурн V (тяжелая, мощная)")
    print("3. Терминатор-5 (средняя, мощная)")
    print("4. Дельта-7 (тяжелая, сверхмощная)")
    print("5. Custom (настройка вручную)")
    choice = input("Введите номер ракеты: ")
    if choice == "1":
        return {
```

```
            "weight": 100,  # Сухой вес ракеты (без топлива), кг
            "full_weight": 150,  # Полный вес ракеты (с топливом), кг
            "fuel_weight": 50,  # Вес топлива, кг
            "engine_operating_time": 1,  # Время работы двигателя, сек
            "engine_thrust": 80000,  # Сила тяги двигателя, Н
            "resistance_coefficient": 0.8 * 3.14 * pow(0.155, 2) / 4,
# Коэффициент сопротивления
            "inclination_angle": np.radians(float(input("Угол наклона
(в градусах): "))),
            # Угол, заданный пользователем
            "det_time": 0.1,  # Временной шаг симуляции, сек
            "height": 0,  # Начальная высота
            "length": 0,  # Горизонтальная длина
            "speed": 0,
            "speedX": 0,
            "speedY": 0,
        }
    elif choice == "2":
        return {
            "weight": 500,  # Сухой вес ракеты, кг
            "full_weight": 2000,  # Полный вес ракеты, кг
            "fuel_weight": 1500,  # Вес топлива, кг
            "engine_operating_time": 180,  # Время работы двигателя,
сек
            "engine_thrust": 3500000,  # Сила тяги двигателя, Н
            "resistance_coefficient": 0.5* 3.14 * pow(0.155, 2) / 4,
            "inclination_angle": np.radians(float(input("Угол наклона
(в градусах): "))),
            "det_time": 0.1,
            "height": 0,
            "length": 0,
            "speed": 0,
            "speedX": 0,
            "speedY": 0,
        }
    elif choice == "3":
        return {
            "weight": 300,  # Сухой вес ракеты, кг
            "full_weight": 300,  # Полный вес ракеты, кг
            "fuel_weight": 150,  # Вес топлива, кг
            "engine_operating_time": 2,  # Время работы двигателя, сек
            "engine_thrust": 1600 * 150 / 2,  # Сила тяги двигателя, Н
            "resistance_coefficient": 0.8 * 3.14 * pow(0.2, 2) / 4,
            "inclination_angle": np.radians(float(input("Угол наклона
(в градусах): "))),
            "det_time": 0.1,
            "height": 0,
            "length": 0,
            "speed": 0,
            "speedX": 0,
            "speedY": 0,
        }
    elif choice == "4":
        return {
            "weight": 600,  # Сухой вес ракеты, кг
            "full_weight": 600,  # Полный вес ракеты, кг
            "fuel_weight": 300,  # Вес топлива, кг
            "engine_operating_time": 3,  # Время работы двигателя, сек
            "engine_thrust": 1600 * 300 / 3,  # Сила тяги двигателя, Н
            "resistance_coefficient": 0.8 * 3.14 * pow(0.3, 2) / 4,
            "inclination_angle": np.radians(float(input("Угол наклона
(в градусах): "))),
            "det_time": 0.1,
```

```
            "height": 0,
            "length": 0,
            "speed": 0,
            "speedX": 0,
            "speedY": 0,
        }
    elif choice == "5":
        print("Настройка вручную:")
        return {
            "weight": float(input("Сухой вес ракеты (без топлива), кг:
")),
            "full_weight": float(input("Полный вес ракеты (с топливом),
кг: ")),
            "fuel_weight": float(input("Вес топлива, кг: ")),
            "engine_operating_time": float(input("Время работы
двигателя, сек: ")),
            "engine_thrust": float(input("Сила тяги двигателя, Н: ")),
            "resistance_coefficient": float(input("Коэффициент
сопротивления: ")),
            "inclination_angle": np.radians(float(input("Угол наклона
(в градусах): "))),
            "det_time": 0.1,
            "height": 0,
            "length": 0,
            "speed": 0,
            "speedX": 0,
            "speedY": 0,
        }
    else:
        print("Некорректный ввод.")
        return choose_rocket()


# Запуск симуляции
rocket = choose_rocket()
simulation = RocketSimulation(rocket)
simulation.simulate()
```

5. ТЕСТИРОВАНИЕ




class RocketSimulation:
def __init__
C:\Users\Francium\AppData\l
Выберите ракету:
1. Falcon 9 (легкая, быстрая)
2. Сатурн V (тяжелая, мощная)
3. Терминатор-5 (средняя, мощная)
4. Дельта-7 (тяжелая, сверхмощная
5. Custom (настройка вручную)
Введите номер ракеты: 3
Угол наклона (в градусах): 45
s\Francium\PycharmProjects\DZ\.ve
64\mingw64\bin\python3.11.exe: ca
finished with exit code 2
Figure 1
Время: 285.10 сек
Макс высота: 9130.89 м
Макс скорость: 1031.21 м/с
Траектория полета ракеты (в реальном времени)
Траектория полета ракеты
Высота (м)
0
2000
4000
6000
8000
0
5000
10000
15000
20000
25000
Горизонтальная длина (м)
Замедлить
Пауза
Ускорить