|  |
| --- |
| **Министерство науки и высшего образования Российской Федерации**  **Федеральное государственное бюджетное образовательное учреждение Высшего Образования «Московский Государственный технический университет имени Н.Э. Баумана (Национальный Исследовательский Университет)»**  **(МГТУ им. Н.Э. Баумана)** |

ФАКУЛЬТЕТ Информатика и системы управления

КАФЕДРА Теоретическая информатика и компьютерные технологии

**РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**К КУРСОВОЙ РАБОТЕ**

**ПО ДИСЦИПЛИНЕ**

**«АЛГОРИТМЫ КОМПЬЮТЕРНОЙ ГРАФИКИ»**

**НА ТЕМУ:**

«Визуализация движения двойного маятника»

Студент ИУ9-51Б Резепин Н. И.

*(группа)* *(подпись, дата)* *(фамилия, и. о.)*

Научный руководитель Каганов Ю. Т.

*(подпись, дата)* *(фамилия, и. о.)*

Консультант

*(подпись, дата)* *(фамилия, и. о.)*

Москва, 2023 г.

**СОДЕРЖАНИЕ**

ВВЕДЕНИЕ3

1 Обзор предметной области5

* 1. Двойной маятник как физическая система5
  2. Численное моделирование движения маятника7

2 Проектирование и разработка продукта13

2.1 Проектирование приложения14

2.1.1 Настольное приложение14

2.1.2 Адаптация приложения14

2.1.2 Структура приложения15

2.2 Выбор инструментов разработки16

2.2.1 Язык программирования Python16

2.2.2 Стек технологий17

3 Реализация продукта18

3.1 Главное окно19

3.2 Виджет настроек21

3.3 Визуализация движения модели24

3.4 Вспомогательные структуры27

ЗАКЛЮЧЕНИЕ29

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ30

ПРИЛОЖЕНИЕ А. Демонстрация работы продукта31

ПРИЛОЖЕНИЕ Б. Листинг исходного кода продукта 34

ПРИЛОЖЕНИЕ В. Сборка и запуск продукта 47

**ВВЕДЕНИЕ**

В настоящее время, время информационных технологий, открываются новые возможности для обучения. Информационные образовательные системы дают возможность интерактивного взаимодействия с предметом изучения, наглядный и понятный интерфейс, находящийся под рукой.

Внедрение и применение информационных систем обучения изменяет подход к ведению образовательной деятельности, меняет функции участников образовательного процесса и совершенствует способы получения и анализа информации.

Уже существует множество информационных систем обеспечения образовательного процесса, создающих единое интерактивное пространство, помогающее человеку работать с большим количеством разнообразной информации.

В связи с вышеперечисленным возникает острая необходимость в образовательных интерактивных приложениях для повышения качества образования путем непосредственного взаимодействия с изучаемой предметной областью.

Особую ценность представляют приложения, позволяющие симулировать те или иные физические процессы или явления, так как они, в отличие от «живых» лабораторных работ, не требуют никакого сложного и специфического оборудования, доступны в любое время и позволяют делать более точные замеры и подробнее наблюдать за предметом изучения.

Двойной математический маятник обладает сильной чувствительностью к изменениям начальных условий. В реальных условиях почти невозможно создать полностью идентичные условия для повторения того или иного характера движения данной физической системы. Электронная же версия двойного маятника дает возможность воспроизводить полностью идентичные начальные условия для повторения эксперимента, а также с огромной точностью менять начальные параметры для изучения стохастичности системы.

Целью данной курсовой работы является разработка программного обеспечения для визуализации движения двойного математического маятника с гибкой настройкой начальных условий.

Поставленной цели соответствуют следующие задачи:

- изучить двойной математический маятник как физическую систему;

- изучить численные методы вычисления приближенных решений систем дифференциальных уравнений первого и второго порядка;

- спроектировать удобный программный продукт для интерактивной работы с физической системой двойного математического маятника;

- изучить необходимый стек технологий для разработки и реализации программного обеспечения;

- разработать и реализовать программное обеспечение для интерактивного изучения физической системы двойного математического маятника.

В процессе работы мной были использованы следующие методы исследования:

- моделирование,

- формализация,

- анализ,

- систематизация.

**1 Обзор предметной области**

**1.1 Двойной маятник как физическая система**

В физике математическим маятником называется механическая система, состоящая из материальной точки, находящейся на конце невесомой нерастяжимой нити или легкого однородного стержня. Другой конец нити или стержня неподвижен. Данная система находится в однородном поле сил тяготения, а именно в поле силы тяжести.

Двойной математический маятник – это механическая система, состоящая из материальной точки А, находящейся на конце невесомой нерастяжимой нити или однородного стержня. Другой конец, точка О, этой нити или стрежня неподвижен. К материальной точке А прикреплена нерастяжимая нить или однородный стержень, на другом конце которого находится материальная точка Б. Данная система также находится в однородном поле сил тяготения. Схема двойного маятника показана на рисунке 1.

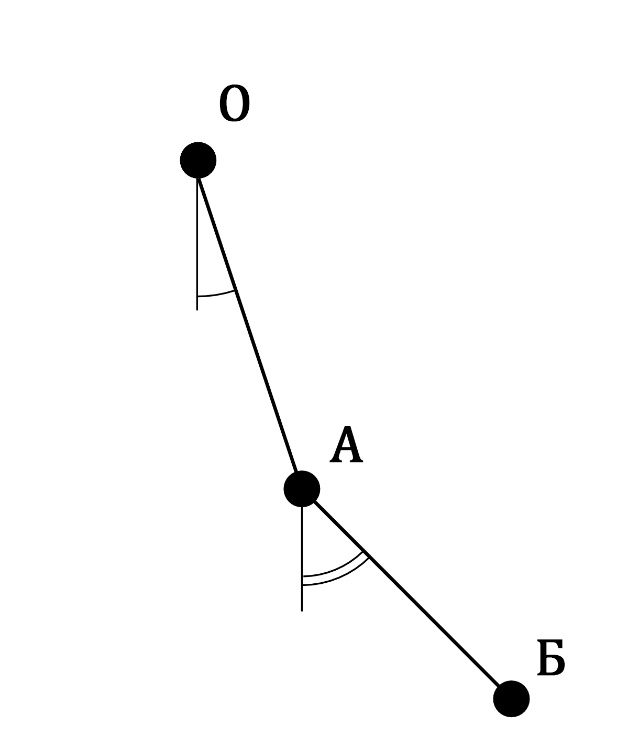


Рисунок 1 – Двойной математический маятник

Двойной маятник является физической системой, которая проявляет разнообразное динамическое поведение со значительной зависимостью от начальных параметров, то есть при изменении начального угла отклонения даже одной из нитей двойного маятника на сотые доли радиана итоговый характер движения будет сильно отличаться от исходного характера движения при фиксированном выбранном угле [1].

Однако хаотической системой двойной маятник является при определенном диапазоне параметров (масс материальных точек, длин стержней и начальных углов отклонения).

Движение двойного маятника является отличным примером колебательных движений механической системы с двумя степенями свободы. Уравнение движения в таких системах описывается системой дифференциальных уравнений. Для двойного маятника это система из четырех дифференциальных уравнений, которая не имеет аналитического решения, что значит она должна быть решена численно.

**1.2 Численное моделирование движения маятника**

Для определенности положим, что модель двойного маятника состоит из двух материальных точек и с массами и соответственно. Материальные точки соединены однородными стержнями с длинами и соответственно, причем массы стержней много меньше масс материальных точек.

Модель двойного маятника находится в однородном поле силы тяжести, силы трения и сопротивления воздуха отсутствуют. Кроме того, из вышесказанного следует, что массами стержней при вычислениях мы можем пренебречь. Описанная модель представлена на рисунке 2.

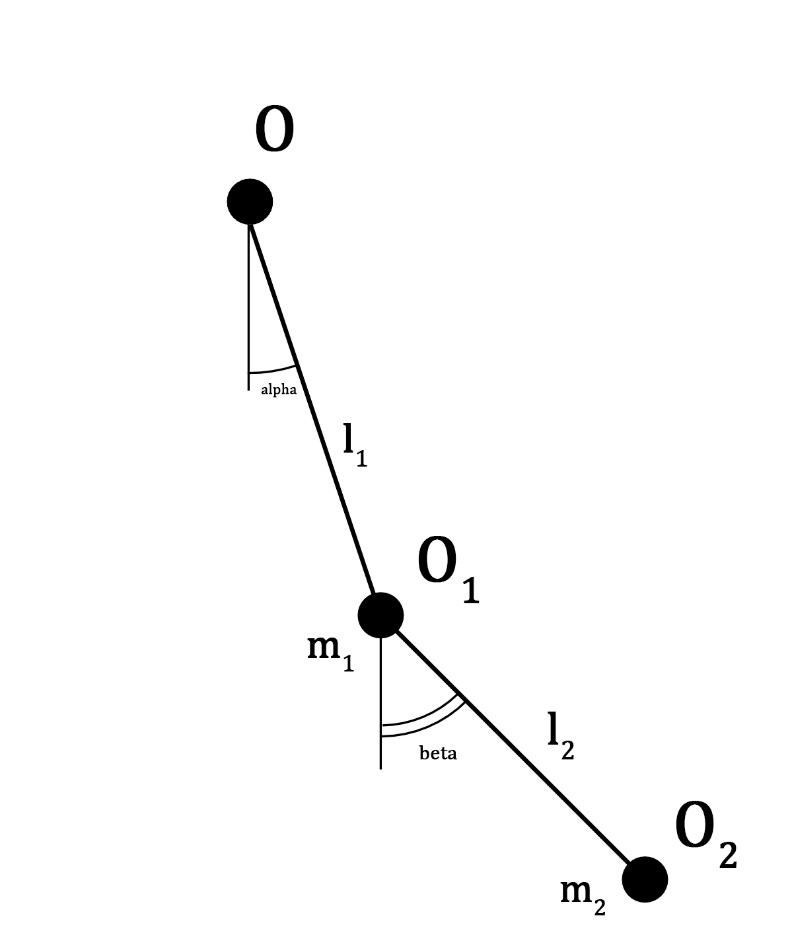


Рисунок 2 – Модель двойного маятника

В лагранжевой механике для описания механической системы используются обобщенные координаты и обобщенные скорости. В качестве обобщенных координат данной системы выберем углы α и β, которые составляют соответственно стержни и с вертикалью, что также показано на рисунке 2. В качестве обобщенных скоростей выберем угловые скорости и материальных точек и соответственно [2].

Посчитаем кинетическую энергию системы, состоящую из суммы кинетических энергий первого и второго маятников.

Кинетическая энергия первого маятника:

(1)

где

Кинетическая энергия второго маятника:

(2)

где

Вектор скорости второй материальной точки можно рассматривать как сумму вектора вращательной скорости точки вокруг точки и вектора собственной скорости точки .

Так как мы считаем, что система идеализирована и углы отклонения стержней от вертикали малы, то можем положить, что .

Откуда следует, что:

(3)

где

Подставим полученное значение скорости точки из (3) в (2) и получим:

(4)

Теперь найдем потенциальную энергию системы. Так как модель находится под действием сил тяжести, то потенциальная энергия системы будет выражаться как:

(5)

где

Зная кинетическую и потенциальную энергию системы возможно записать уравнение Лагранжа второго рода, найти из него уравнение частот и уравнения главных колебаний, из которых получаются уравнения движения модели маятника [3]:

(6)

(7)

(8)

где

Для описания движения модели двойного маятника необходимо решить уравнения (6) – (8) при условии времени .

Нам необходимо решить данную систему с учетом колебаний с произвольной амплитудой. Такая система уравнений не решается аналитически, поэтому мы будем использовать численные методы поиска и приближения решений.

Существует множество методов численного решения систем дифференциальных уравнений, однако для данной ситуации наиболее удобным и точным будет метод Рунге-Кутта четвертого порядка, который заключается в вычислении приближенного значения итерационным путем с постоянным шагом.

На данный момент у нас есть нелинейная система двух дифференциальных уравнений второго порядка [4]:

+

(9)

(10)

Для удобства численного приближения перейдем от уравнений Лагранжа к форме канонических уравнений Гамильтона. В результате вместо двух уравнений второго порядка мы будем работать с системой из четырех уравнений первого порядка.

Для начала перейдем к обобщенным координатам и обобщенным импульсам от обобщенных координат с обобщенными скоростями. Первые остаются неизменными – и . Введем обобщенные импульсы и следующим образом:

(11)

Теперь мы можем перейти от лагранжевой формы к гамильтоновой форме уравнений с помощью преобразования Лежандра [5], получим систему:

(12)

где

Теперь мы можем воспользоваться методом Рунге-Кутта. Запишем систему в векторной форме:

(13)

где

.

Вектор составлен из четырех канонических переменных данной системы (12), а компоненты вектора соответствуют правым частям дифференциальных уравнений.

Метод Рунге-Кутта предполагает на каждом шаге последовательное вычисление четырех промежуточных векторов:

(14)

(15)

(16)

(17)

Значение вектора в следующем временном узле вычисляется по формуле:

. (18)

Суммарная ошибка данного алгоритма на конечном интервале имеет порядок O(τ4), то есть точность вычислений возрастает в 16 раз при уменьшении временного шага τ в два раза.

**2 Проектирование и разработка продукта**

Процесс создания конечного продукта включает в себя несколько этапов:

- проектирование (создание общего представления о конечном продукте, связях и зависимостях в продукте, выполняемых им функциях и т.д.);

- разработка (выбор инструментов реализации, определение конкретного приложения проекта с учетом ограничений и функционала выбранных инструментов и т.д.);

- реализация (непосредственно написание программного кода продукта, его тестирование и доработка).

**2.1 Проектирование приложения**

В силу того, что основной функцией разрабатываемого продукта является образовательная и демонстрационная, оптимальным решением является разработка настольного (*desktop*) приложения.

Такой подход может обеспечить удобное распространение и установку демонстрационного приложения в компьютерных классах или на рабочих ПК в учебных классах.

**2.1.1 Настольное приложение**

Главной особенностью и главным преимуществом настольных приложений является их автономность – для работы такого приложения не нужно подключение к сети Интернет, необходимы только операционная система на настольном устройстве и предварительная установка продукта.

Кроме того, отсутствие зависимости в логике программы от подключения к удаленному серверу повышает скорость работы, а также функциональность приложения.

Важным требованием для настольного приложения является кроссплатформенность, то есть способность продукта запускаться под различными операционными системами и не зависеть от среды развертывания. Продукт, разработанный мной, полностью удовлетворяет этому требованию – приложение можно запустить под любой операционной системой, имеющей графическую оболочку рабочего стола.

**2.1.2 Адаптация приложения**

Настольное приложение должно иметь *GUI* (англ. *graphical user interface*) – графический интерфейс для удобной работы пользователя с продуктом. Кроме того, графический интерфейс должен быть адаптирован для всех операционных систем, имеющих графическую оболочку рабочего стола.

Существует два основных решения данной проблемы:

а) разработка отдельных версий приложений для *MacOS*, *Unix* и *Windows* систем с обеспечением адаптации интерфейса к особенностям каждой из операционных систем;

б) разработка кроссплатформенного интерфейса, который будет использовать базовые шаблоны виджетов окна каждого из типов операционных систем.

Для данного продукта мной был выбран второй вариант.

**2.1.3 Структура приложения**

Приложение должно включать в себя управляющее окно, на котором будут отображаться виджеты запуска и остановки визуализации движения модели двойного математического маятника, таймер, виджет настроек физических характеристик модели, а также виджет выбора типа отслеживания траектории движения.

Для данного набора функциональных виджетов нет смысла создавать более одного окна, как и нет смысла создавать справочные окна, так как интерфейс в силу своей минималистичности не требует дополнительных пояснений.

Вышеперечисленные виджеты будут расположены в окне программы, как показано на рисунке 3.

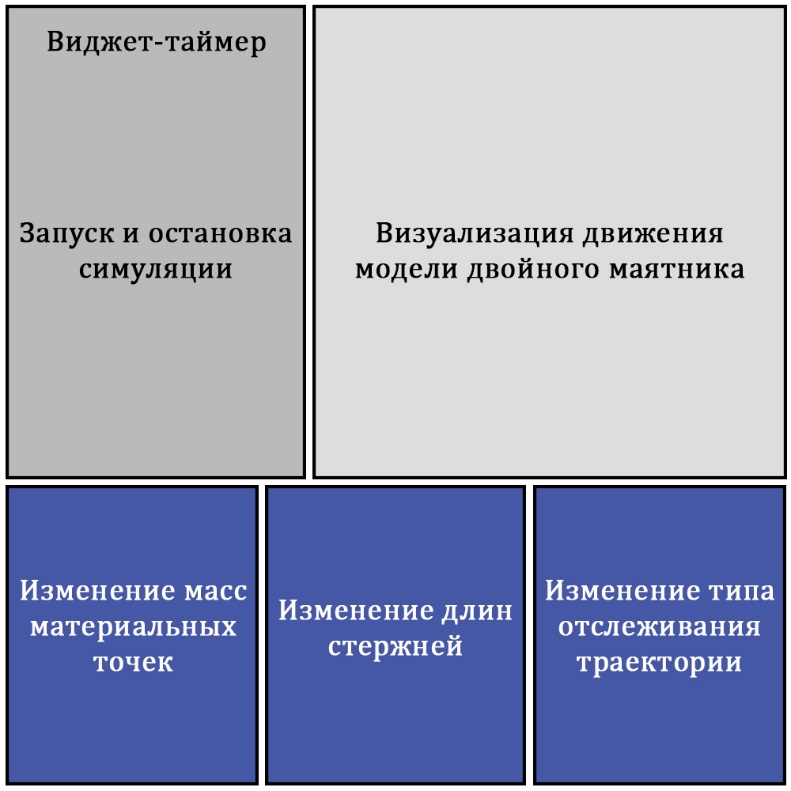


Рисунок 3 – Макет окна приложения

**2.2 Выбор инструментов разработки**

**2.2.1 Язык программирования Python**

*Python* — высокоуровневый язык программирования общего назначения с динамической строгой типизацией и автоматическим управлением памятью, ориентированный на повышение производительности разработчика, читаемости кода и его качества, а также на обеспечение переносимости написанных на нём программ.



Рисунок 4 – Логотип Python

Язык является полностью объектно-ориентированным в том плане, что всё является объектами, однако с точки зрения оформления кода и возможностей разработки *python* является мультипарадигменным языком программирования, поддерживающим императивное, процедурное, структурное, объектно-ориентированное программирование, метапрограммирование и функциональное программирование.

Стандартная библиотека включает большой набор полезных переносимых функций, начиная с возможностей для работы с текстом и заканчивая средствами для написания сетевых приложений.

Python стал одним из самых популярных языков, он используется в анализе данных, машинном обучении, *DevOps* и *web*-разработке, а также в других сферах, включая разработку игр.

Язык использует динамическую типизацию вместе с подсчётом ссылок и циклический сборщик мусора для менеджмента памяти. Также есть динамические разрешения имён (динамическое связывание), которые связывают имена методов и переменных во время выполнения программы.

**2.2.2 Стек технологий**

Для обеспечения кроссплатформенности мной был выбрана библиотека *tkinter*, для математических расчетов – *numpy* и некоторые другие стандартные библиотеки python.

*Tkinter* (от англ. *Tk interface*) — кроссплатформенная событийно-ориентированная графическая библиотека на основе средств *Tk* (широко распространённая в мире *GNU/Linux* и других *UNIX*‐подобных систем, портирована также и на *Microsoft Windows*).

*Tkinter* является продолжением идеи *Tk* – некоторого набора инструментов (*toolbox*), который вне зависимости от операционной системы предоставляет возможность оперировать базовыми элементами графического интерфейса для создания приложений, что полностью решает вопрос с адаптацией продукта под различные операционные системы.

Минусами такого решения является достаточно простой интерфейс, состоящий из базовых виджетов окон операционной системы, однако для данной задачи (а именно задачи создания обучающего демонстрационного пособия) этот инструмент подходит идеально.

**3 Реализация продукта**

Как было сказано ранее, приложение состоит из одного основного управляющего окна. Графический пользовательский интерфейс приложения разработан с помощью *Tkinter*.

При открытии приложения перед пользователем открывается окно, на котором в правом верхнем углу располагается модель двойного маятника в статичном положении, занимающая большую часть окна, панель запуска и остановки симуляции с таймером в левом верхнем углу. Под ними горизонтально расположены ползунки настроек и кнопка для применения изменений. Конечную версию приложения можно увидеть на рисунке 5.

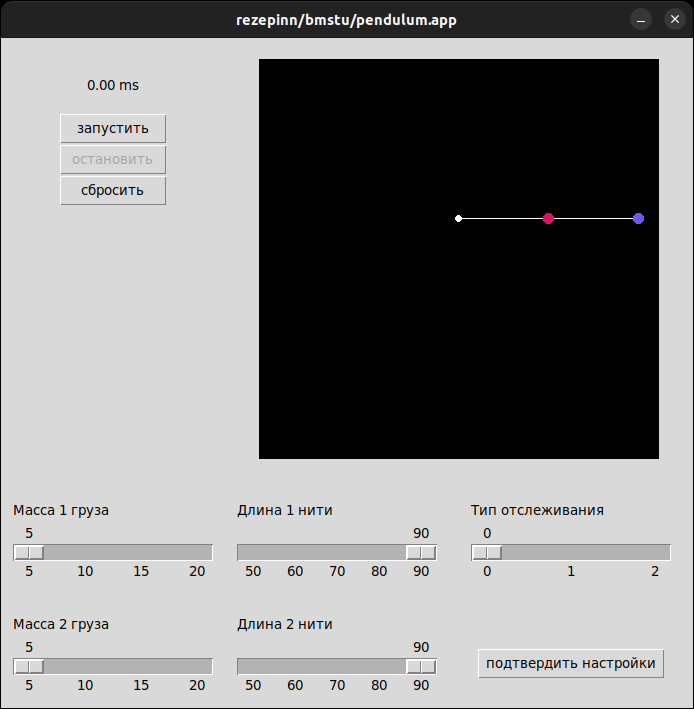


Рисунок 5 – Конечная версия приложения

Примечание – Все снимки экрана приложения и этапов его разработки сделаны под операционной системой *Linux Ubuntu 22.04 LTS*. Примеры того, как выглядит продукт при работе с другими операционными системами, представлены в Приложении А.

У пользователя есть возможность запустить движение модели, изменить начальные настройки маятника, сбросить настройки до начальных значений по умолчанию.

**3.1 Главное окно**

Если мы посмотрим на файл запуска, то увидим в нем ровно 2 строки.

from app import App  
  
App()

Рисунок 6 – Листинг файла main.py

В этих строках создается экземпляр класса *App*, который отвечает за создание, отображение и работу с главным окном приложения.

При инициализации нового объекта происходит создание и настройка нового окна; создание полей, хранящих объекты-виджеты (таймер – *class Timer*, двойной маятник – *class Pendulum*, настройки – *class Settings*), инициализация соответствующих виджетов и инициализация управляющих кнопок.

class App:  
 def \_\_init\_\_(self):  
 # initiate root window  
 self.root = tk.Tk()  
 self.root.title('rezepinn/bmstu/pendulum.app')  
 self.root.resizable(False, False)  
  
 # create widgets  
 self.timer = Timer(self.root)  
 self.pendulum = Pendulum(self.root)  
 self.settings = Settings(self.root, self.pendulum)  
  
 # initiate command buttons  
 self.button\_start = tk.Button()  
 self.button\_stop = tk.Button()  
 self.button\_reset = tk.Button()  
 self.init\_and\_grid\_buttons()  
  
 # start listening  
 self.root.mainloop()

Рисунок 7 – Инициализация объекта класса App

Каждая из кнопок выполняет свою функцию. Кнопки инициализируются и размещаются в окне приложения с помощью отдельного метода *init\_and\_grid\_buttons()*.

Кнопка *button\_start* отвечает за запуск таймера, начало движения двойного маятника, блокировку кнопок запуска, сброса и применения настроек, однако после нажатия на нее активируется кнопка остановки симуляции *button\_stop*.

Кнопка *button\_stop* отвечает за остановку таймера, прекращение движения двойного маятника, блокировку кнопки остановки, однако после нажатия на нее активируются кнопки запуска, сброса и применения настроек.

Кнопка *button\_reset* отвечает за сброс настроек маятника до начального значения по умолчанию. В соответствующее положение приводятся и ползунки настройки маятника, находящиеся в нижней части окна. Кроме того, маятник возвращается в начальное положение.

Над управляющими кнопками находится таймер, реализованный в отдельном классе. Расположение управляющих кнопок и таймера представлено на рисунке 8.

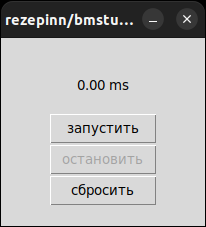


Рисунок 8 – Управляющие кнопки и таймер

Реализация таймера тривиальна и не нуждается в дополнительных комментариях. Стоит только отметить тот факт, что при инициализации объекта класса *Timer* в конструктор передается объект *tkinter.Tk* [7]. Это сделано для того, чтобы дать возможность объекту класса Timer при инициализации расположить необходимые виджеты внутри главного окна.

**3.2 Виджет настроек**

Виджет настроек, представленный объектом класса *Settings*, инициализируется при инициализации соответствующего объекта класса *App*.

Данный виджет предоставляет пользователю возможность:

а) изменять массу первой материальной точки,

б) изменять массу второй материальной точки,

в) изменять длину первого стержня,

г) изменять длину второго стержня,

д) изменять тип отображения траектории движения маятника.

Логически и визуально ползунки настроек разбиты на три группы: настройка масс, настройка длин, настройка трекинга. Расположение ползунков можно увидеть на рисунке 9.

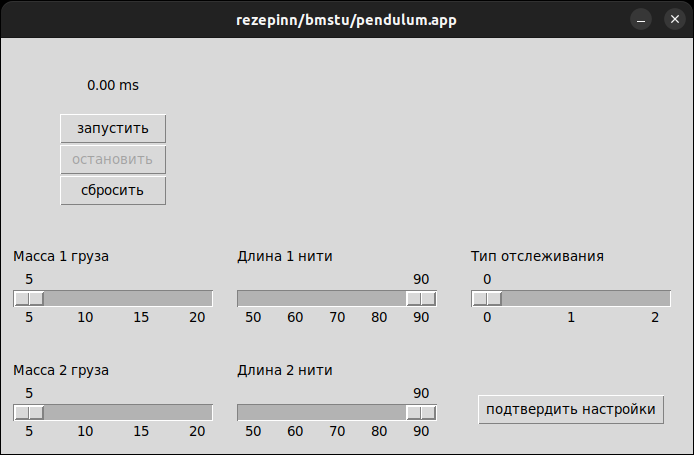


Рисунок 9 – Настройки

Для удобства пользователя в качестве переключателей был выбран объект класса *tkinter.Scale* [7], то есть шкала. Данный выбор был сделан из соображений создания интуитивно понятного интерфейса. Пользователю нужно лишь перетащить ползунок на нужное ему значение.

При перетаскивании ползунка по шкале не происходит никаких изменений, для применения настроек необходимо использовать кнопку «подтвердить настройки».

Кроме того, данная кнопка становится активной только в случае, если маятник остановлен. Менять настройки маятника в процессе его движения не получится. Данная функция сделана для защиты от случайного переключения настроек пользователем в процессе наблюдения.

При нажатии на кнопку «подтвердить настройки» сначала выполняется применение новых настроек (а именно вызывается соответствующие методы объекта класса Pendulum, который передается в объект класса Settings при инициализации), после чего картинка в окне приложения обновляется.

Логика программы выстроена таким образом, что вне зависимости от характера смены типа отслеживания движения предыдущая траектория движения маятника, если она отображалась, удаляется с окна, после чего создается новая.

Всего существует три типа отслеживания движения маятника:

а) положение «0» - траектория движения не отображается,

б) положение «1» - отображается траектория движения нижней материальной точки,

в) положение «2» - отображается траектория движения обеих материальных точек.

Различные типы отслеживания движения маятника показаны на рисунках 10-12:

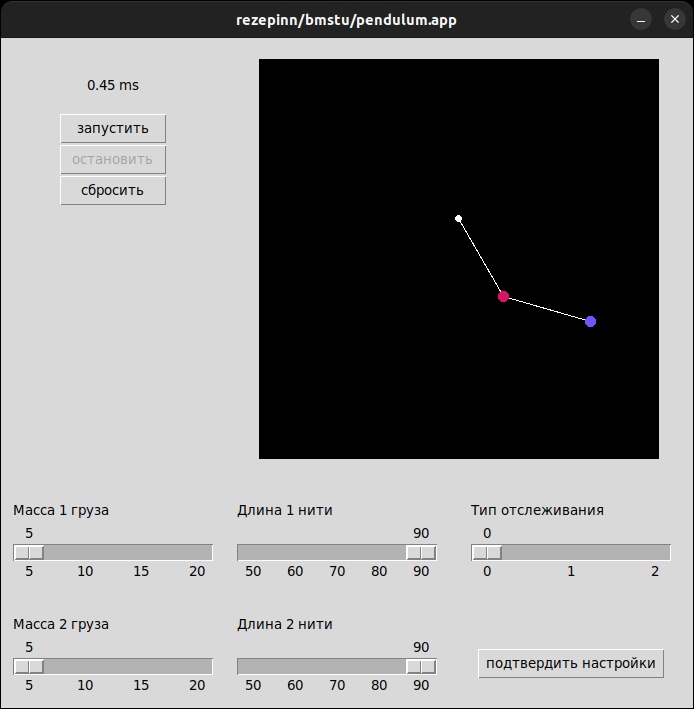


Рисунок 10 – Движение без отслеживания траектории движения

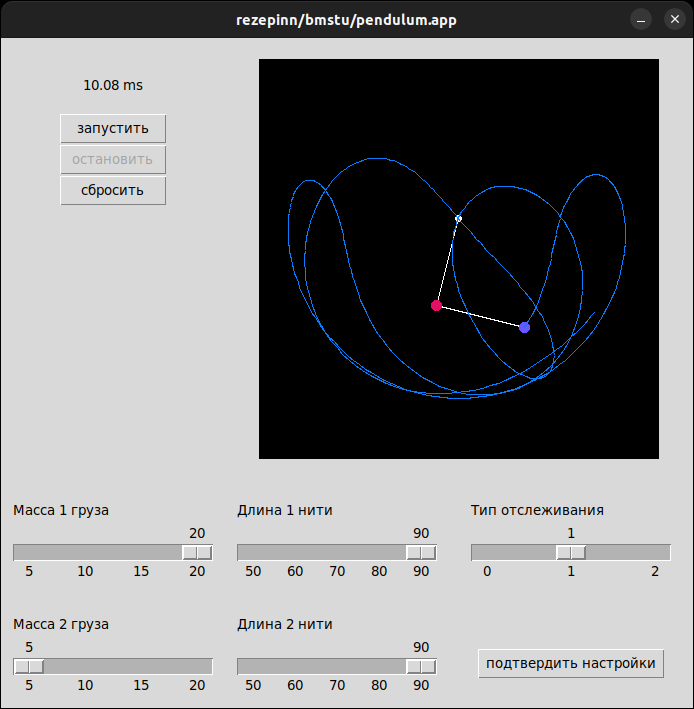


Рисунок 11 – Движение с отслеживанием траектории нижней точки

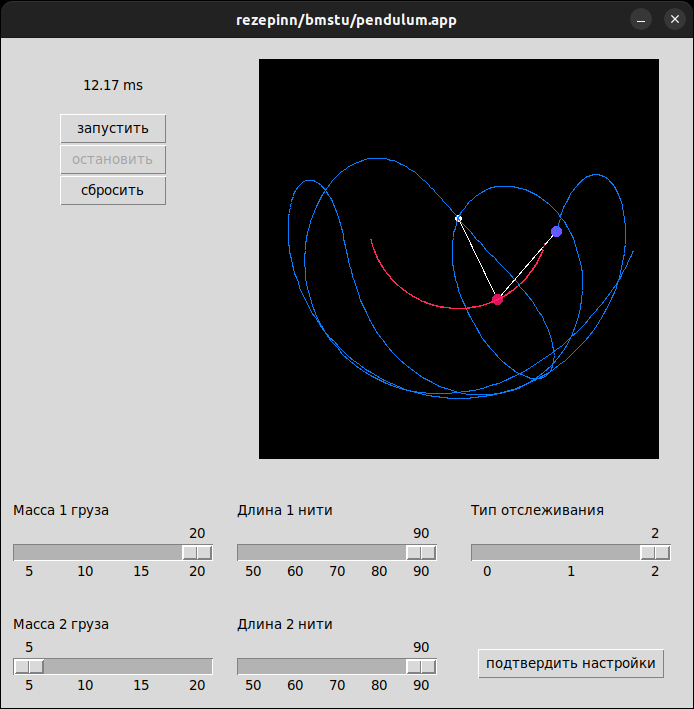


Рисунок 12 – Движение с отслеживанием траекторий обеих точек

**3.3 Визуализация движения модели**

Виджет-симулятор движения, представленный объектом класса *Pendulum*, инициализируется при инициализации соответствующего объекта класса *App*.

class Pendulum:  
 def \_\_init\_\_(self, root):  
 # set root window  
 self.root = root  
  
 # initiate pendulum canvas  
 self.canvas = tk.Canvas()  
 self.initiate\_and\_grid\_canvas()  
  
 # initiate pendulum settings  
 self.length1 = 90  
 self.length2 = 90  
 self.mass1 = 5.0  
 self.mass2 = 5.0  
 self.alpha = np.pi / 2  
 self.beta = np.pi / 2  
 self.d\_alpha = 0.0  
 self.d\_beta = 0.0  
  
 # set center pendulum coordinates  
 self.center\_coords = Point(CANVAS\_WIDTH / 2, 2 \* CANVAS\_HEIGHT / 5)  
  
 # initiate pendulum bob's coordinates  
 self.bob1\_coords = Point(self.center\_coords.get\_x() + self.length1, self.center\_coords.get\_y())  
 self.bob2\_coords = Point(self.bob1\_coords.get\_x() + self.length2, self.bob1\_coords.get\_y())  
  
 # initiate tracer  
 self.bob1\_tracer = tracer.Tracer()  
 self.bob2\_tracer = tracer.Tracer()  
 self.initiate\_tracers()  
  
 # pendulum movement and drawing flags  
 self.is\_active = False  
 self.tracing\_mode = 0  
  
 # set start state  
 self.draw\_pendulum()

Рисунок 13 – Инициализация объекта класса Pendulum

При создании объекта класса *Pendulum* задаются начальные настройки двойного маятника: массы материальных точек, длины стержней, начальные углы отклонения, начальные угловые скорости материальных точек.

Объект класса *Pendulum* хранит информацию о координатах точки подвеса, координатах материальных точек, объекты класса *Tracer* для каждой из материальных точек, информацию о текущем состоянии маятника, а именно происходит ли на данный момент движение, какой тип отображения траектории выбран.

Основная задача данного виджета – отображение визуализации движения модели двойного математического маятника. Данный класс отвечает за отрисовку самого маятника и отрисовку траекторий в зависимости от выбранного типа слежения.

Для вычисления координат каждой из материальных точек используется формула, использующая угол отклонения и длину стержня на данный момент времени, как это показано на рисунке 14.

def calculate\_bob1\_coords(self):  
 # set new coordinates of first bob  
 self.bob1\_coords = Point(  
 self.center\_coords.get\_x() + self.length1 \* np.sin(self.alpha),  
 self.center\_coords.get\_y() + self.length1 \* np.cos(self.alpha)  
 )  
  
 # add new coordinates to trace line  
 self.bob1\_tracer.add\_coord(  
 self.center\_coords.get\_x() + self.length1 \* np.sin(self.alpha),  
 self.center\_coords.get\_y() + self.length1 \* np.cos(self.alpha)  
 )  
  
def calculate\_bob2\_coords(self):  
 # set new coordinates of second bob  
 self.bob2\_coords = Point(  
 self.bob1\_coords.get\_x() + self.length2 \* np.sin(self.beta),  
 self.bob1\_coords.get\_y() + self.length2 \* np.cos(self.beta)  
 )  
  
 # add new coordinates to trace line  
 self.bob2\_tracer.add\_coord(  
 self.bob1\_coords.get\_x() + self.length2 \* np.sin(self.beta),  
 self.bob1\_coords.get\_y() + self.length2 \* np.cos(self.beta)  
 )

Рисунок 14 – Вычисление координат материальных точек

Значение длин стержней известно нам в каждый момент времени по определению. Значение углов отклонения в каждый момент времени вычисляется с помощью формул (14) – (18), выведенных ранее с использованием численных методов приближения системы (12). Соответствующий программный код представлен на рисунке 15:

def calculate\_pendulum\_angles(self):  
 # components for the 2nd derivative of alpha for first bob  
 component\_1 = (- G \* (2 \* self.mass1 + self.mass2) \* np.sin(self.alpha)) - \  
 (self.mass2 \* G \* np.sin(self.alpha - 2 \* self.beta))  
  
 component\_2 = (2 \* np.sin(self.alpha - self.beta) \* self.mass2) \* \  
 (self.d\_beta \*\* 2 \* self.length2 +  
 self.d\_alpha \*\* 2 \* self.length1 \* np.cos(self.alpha - self.beta))  
  
 component\_3 = (2 \* self.length1 \* self.mass1) + \  
 (self.length1 \* self.mass2 \* (1 - np.cos(2 \* self.alpha - 2 \* self.beta)))  
  
 # second derivative of alpha  
 dd\_alpha = (component\_1 - component\_2) / component\_3  
  
 # components for the 2nd derivative of beta for second bob  
 component\_1 = 2 \* np.sin(self.alpha - self.beta)  
  
 component\_2 = (self.d\_alpha \*\* 2 \* self.length1 \* (self.mass1 + self.mass2)) + \  
 (G \* (self.mass1 + self.mass2) \* np.cos(self.alpha)) + \  
 (self.d\_beta \*\* 2 \* self.length2 \* self.mass2 \* np.cos(self.alpha - self.beta))  
  
 component\_3 = (2 \* self.length2 \* self.mass1) + \  
 (self.length2 \* self.mass2) + \  
 (self.length2 \* self.mass2 \* np.cos(2 \* (self.alpha - self.beta)))  
  
 # second derivative of beta  
 dd\_beta = (component\_1 \* component\_2) / component\_3  
  
 # update d\_angles and angles values  
 self.d\_alpha += dd\_alpha \* DELTA  
 self.d\_beta += dd\_beta \* DELTA  
 self.alpha += self.d\_alpha \* DELTA  
 self.beta += self.d\_beta \* DELTA

Рисунок 15 – Вычисление углов отклонения

**3.4 Вспомогательные структуры**

Для реализации основного функционала приложения были созданы также вспомогательные структуры.

Для хранения информации о координатах материальных точек, точки подвеса шарнира, отслеживающих траекторию линий был создан класс *Point*, который реализует базовый функционал хранения точки на холсте (*tkinter.Canvas*[7]). Класс точки представлен на рисунке 16.

class Point:  
 def \_\_init\_\_(self, x, y):  
 self.x = x  
 self.y = y  
  
 def get\_x(self):  
 return self.x  
  
 def get\_y(self):  
 return self.y

Рисунок 16 – Класс Point

Для хранения информации о траекториях движения материальных точек двойного маятника был создан класс *Trace*, который реализует базовый функционал хранения истории движения материальной точки. Класс отслеживания траектории представлен на рисунке 17.

from point import Point  
  
  
class Tracer:  
 def \_\_init\_\_(self):  
 self.trace = []  
 self.history\_length = 100  
  
 def add\_coord(self, x, y):  
 if len(self.trace) >= self.history\_length:  
 self.trace = self.trace[1:]  
 self.trace.append(Point(x, y))  
  
 def get\_trace(self):  
 return self.trace

Рисунок 17 – Класс Tracer

Для хранения глобальных констант был выделен отдельный файл. Такое решение было выбрано для быстрого доступа к глобальным настройкам проекта, их изменения и структурирования кода. Файл с глобальными переменными представлен на рисунке 18.

BOB1\_COLOR = '#DD1265'  
BOB2\_COLOR = '#6B57FF'  
  
TRACE\_BOB1\_COLOR = '#FE2857'  
TRACE\_BOB2\_COLOR = '#087CFA'  
  
G = 9.8  
DELTA = 0.1  
  
CANVAS\_WIDTH = 400  
CANVAS\_HEIGHT = 400

Рисунок 18 – Глобальные настройки

В этих настройках указаны цвета материальных точек маятника, и цвета линий траекторий для каждой из материальных точек маятника, используемые при отрисовке, ускорение свободного падения и шаг численного приближения, используемые для вычислений, размеры холста, на котором отображается двойной маятник.

**ЗАКЛЮЧЕНИЕ**

В процессе работы мне удалось разработать программное обеспечение для интерактивного изучения физической системы двойного математического маятника с гибкой настройкой начальных состояний системы.

Я изучил процесс разработки обучающих информационных продуктов и повторил производственный цикл в процессе достижения конечного результата работы.

При выполнении работы были решены следующие задачи:

- изучена физическая система двойного математического маятника, аналитически получены его основные характеристики в системе лагранжевой механики;

- изучены численные методы вычисления приближенных решений систем дифференциальных уравнений, для решения системы из четырех дифференциальных уравнений первого порядка был выбран метод Рунге-Кутта четвертого порядка, обеспечивший достаточную точность численного нахождения углов отклонения стержней;

- спроектирован, разработан и реализован удобный программный продукт для интерактивной работы с физической системой двойного математического маятника с помощью таких инструментов как Python 3.11.1, Tkinter 3.11.1 и Numpy 1.24.0 вместе с IDE PyCharm.

Поставленная цель и соответствующие ей задачи были выполнены в полном объеме. Кроме того, разработанное программное обеспечение может масштабироваться и модифицироваться при необходимости.

**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. Учебник для вузов // В. И. Дронг, В. В. Дубинин, М. М. Ильин и др.; Под общ. ред. К. С. Колесникова. 3-е изд., стереотип. — М.: Изд-во МГТУ им. Н. Э. Баумана, 2005. — 736 е.: ил. (Сер. Механика в техническом университете; Т. 1).

2. Учебное пособие по теоретической механике МФТИ: [Электронный ресурс]. URL: <https://mipt.ru/upload/medialibrary/330/amelkin-lagranzheva-i-gamiltonova-mekhanika-ver2.pdf> (Дата обращения: 30.02.2023).

3. Голубева О.В. Теоретическая механика // Научная библиотека: [Электронный ресурс]. URL: http://lib.sernam.ru/book\_g\_tm.php?id=119. (Дата обращения: 29.02.2023).

4. Дифференциальные уравнения: [Электронный ресурс] // Двойной маятник. URL: http://www.math24.ru/двойной-маятник.html (Дата обращения: 29.01.2023).

5. Решение неконсервативных задач теории устойчивости / В.П. Радин, Ю.Н. Самогин, В.П. Чирков, А.В. Щугорев – М.: ФИЗМАТЛИТ, 2017. – 240с.

7. Документация Tkinter: [Электронный ресурс]. URL: <https://docs.python.org/3/library/tkinter.html>. (Дата обращения: 29.01.2023).

**ПРИЛОЖЕНИЕ А**

**Демонстрация работы продукта**

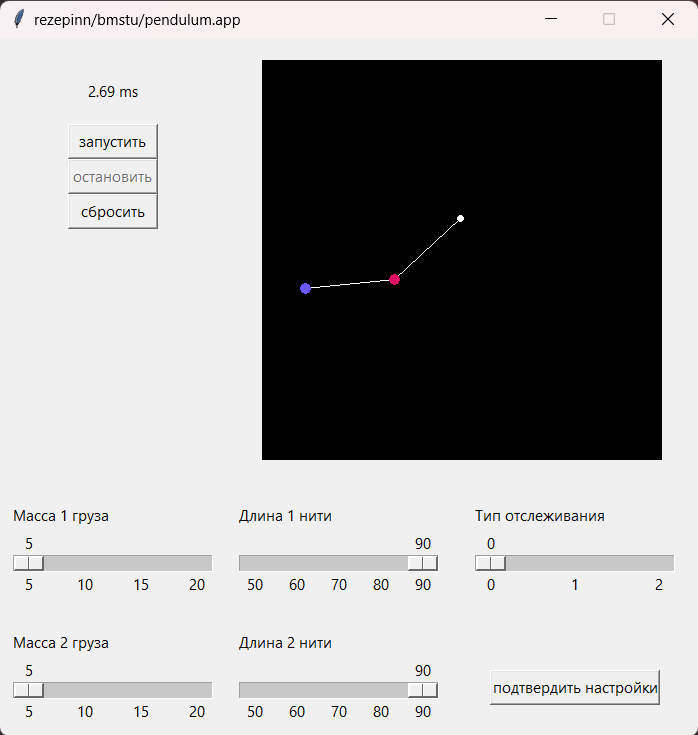


Рисунок А.1 – Работа приложения (ОС - Windows 11)

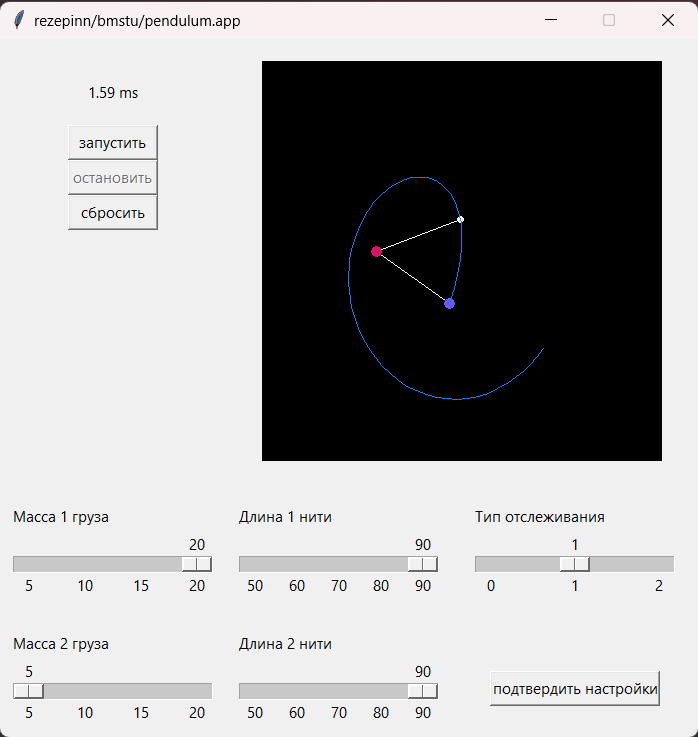


Рисунок А.2 – Работа приложения (ОС - Windows 11)

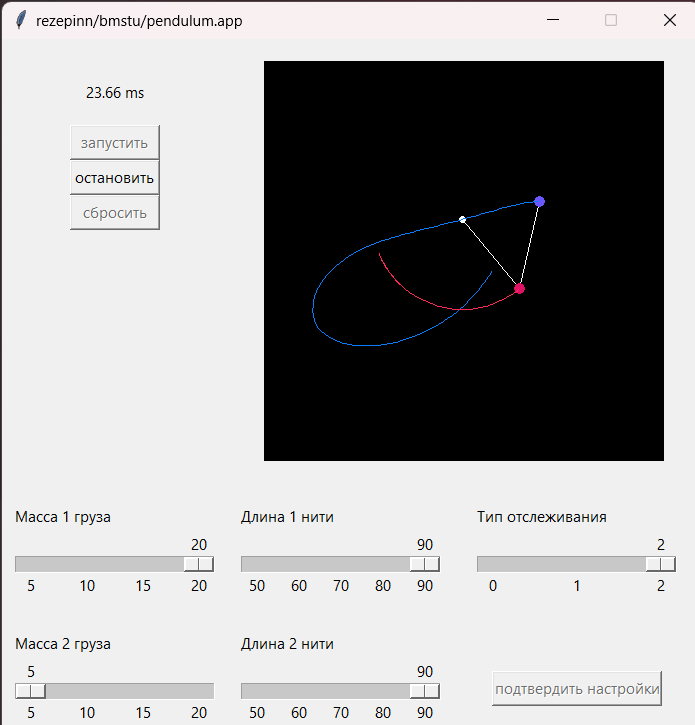


Рисунок А.3 – Работа приложения (ОС - Windows 11)

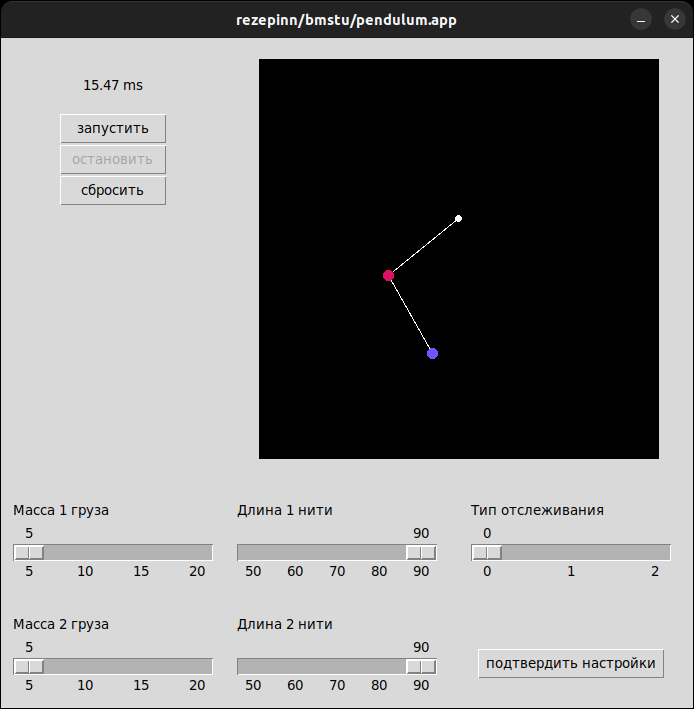


Рисунок А.4 – Работа приложения (ОС - Ubuntu 22.04)

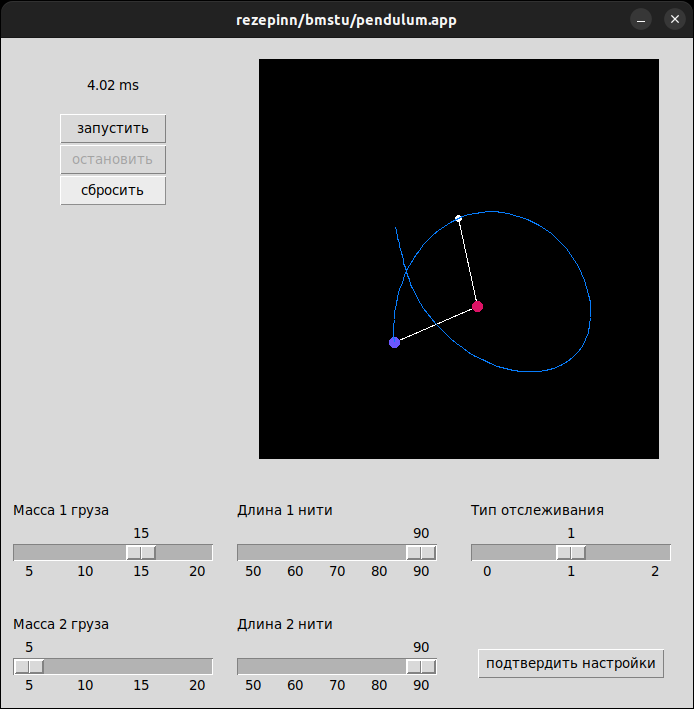


Рисунок А.5 – Работа приложения (ОС - Ubuntu 22.04)

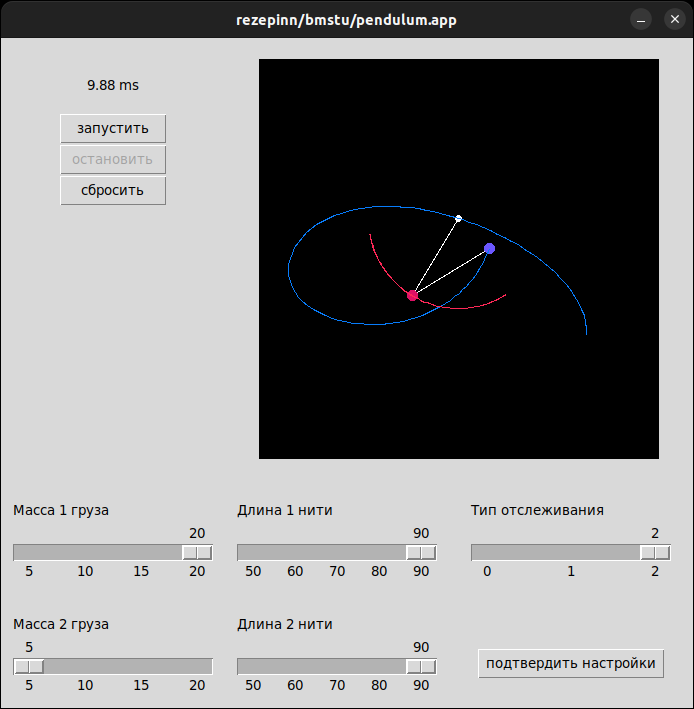


Рисунок А.6 – Работа приложения (ОС - Ubuntu 22.04)

Видео работы приложения (ОС - Windows 11) опубликовано по ссылке: <https://disk.yandex.ru/i/cienW0fyJmIPIQ>

**ПРИЛОЖЕНИЕ Б**

**Листинг исходного кода продукта**

**Б.1 Листинг файла main.py**

from app import App  
  
App()

**Б.2 Листинг файла app.py**

import tkinter as tk  
  
from pendulum import Pendulum  
from settings import Settings  
from timer import Timer  
  
  
class App:  
 def \_\_init\_\_(self):  
 # initiate root window  
 self.root = tk.Tk()  
 self.root.title('rezepinn/bmstu/pendulum.app')  
 self.root.resizable(False, False)  
  
 # create widgets  
 self.timer = Timer(self.root)  
 self.pendulum = Pendulum(self.root)  
 self.settings = Settings(self.root, self.pendulum)  
  
 # initiate command buttons  
 self.button\_start = tk.Button()  
 self.button\_stop = tk.Button()  
 self.button\_reset = tk.Button()  
 self.init\_and\_grid\_buttons()  
  
 # start listening  
 self.root.mainloop()  
  
 def start(self):  
 # set buttons state  
 self.button\_start['state'] = 'disabled'  
 self.button\_stop['state'] = 'normal'  
 self.button\_reset['state'] = 'disabled'  
 self.settings.confirm\_button['state'] = 'disabled'  
  
 # start pendulum  
 self.pendulum.start\_pendulum()  
  
 # start timer  
 self.timer.start\_timer()  
  
 def stop(self):  
 # set buttons state  
 self.button\_start['state'] = 'normal'  
 self.button\_stop['state'] = 'disabled'  
 self.button\_reset['state'] = 'normal'  
 self.settings.confirm\_button['state'] = 'normal'  
  
 # stop pendulum  
 self.pendulum.stop\_pendulum()  
  
 # stop timer  
 self.timer.stop\_timer()  
  
 def reset(self):  
 # reset pendulum  
 self.pendulum.reset\_pendulum()  
 self.pendulum.draw\_pendulum()  
 self.pendulum.draw\_traces()  
  
 # reset settings  
 self.settings.reset\_settings()  
  
 def init\_and\_grid\_buttons(self):  
 self.button\_start = tk.Button(  
 master=self.timer.get\_timer\_frame(),  
 text='запустить',  
 width=10,  
 height=1,  
 command=self.start  
 )  
 self.button\_start.grid(  
 row=1,  
 rowspan=1,  
 column=0,  
 columnspan=3  
 )  
  
 self.button\_stop = tk.Button(  
 master=self.timer.get\_timer\_frame(),  
 text='остановить',  
 width=10,  
 height=1,  
 command=self.stop,  
 state='disabled'  
 )  
 self.button\_stop.grid(  
 row=2,  
 rowspan=1,  
 column=0,  
 columnspan=3  
 )  
  
 self.button\_reset = tk.Button(  
 master=self.timer.get\_timer\_frame(),  
 text='сбросить',  
 width=10,  
 height=1,  
 command=self.reset  
 )  
 self.button\_reset.grid(  
 row=3,  
 rowspan=1,  
 column=0,  
 columnspan=3  
 )

**Б.3 Листинг файла pendulum.py**

import tkinter as tk  
import numpy as np  
from point import Point  
  
import tracer  
from consts import \*  
  
  
class Pendulum:  
 def \_\_init\_\_(self, root):  
 # set root window  
 self.root = root  
  
 # initiate pendulum canvas  
 self.canvas = tk.Canvas()  
 self.initiate\_and\_grid\_canvas()  
  
 # initiate pendulum settings  
 self.length1 = 90  
 self.length2 = 90  
 self.mass1 = 5.0  
 self.mass2 = 5.0  
 self.alpha = np.pi / 2  
 self.beta = np.pi / 2  
 self.d\_alpha = 0.0  
 self.d\_beta = 0.0  
  
 # set center pendulum coordinates  
 self.center\_coords = Point(CANVAS\_WIDTH / 2, 2 \* CANVAS\_HEIGHT / 5)  
  
 # initiate pendulum bob's coordinates  
 self.bob1\_coords = Point(self.center\_coords.get\_x() + self.length1, self.center\_coords.get\_y())  
 self.bob2\_coords = Point(self.bob1\_coords.get\_x() + self.length2, self.bob1\_coords.get\_y())  
  
 # initiate tracer  
 self.bob1\_tracer = tracer.Tracer()  
 self.bob2\_tracer = tracer.Tracer()  
 self.initiate\_tracers()  
  
 # pendulum movement and drawing flags  
 self.is\_active = False  
 self.tracing\_mode = 0  
  
 # set start state  
 self.draw\_pendulum()  
  
 def stop\_pendulum(self):  
 self.is\_active = False  
 self.initiate\_tracers()  
  
 def start\_pendulum(self):  
 self.is\_active = True  
 self.update\_pendulum()  
  
 def update\_pendulum(self):  
 if self.is\_active:  
 # calculating  
 self.calculate\_pendulum\_state()  
  
 # drawing  
 self.draw\_pendulum()  
 self.draw\_traces()  
  
 # next step  
 self.root.after(10, self.update\_pendulum)  
  
 def reset\_pendulum(self):  
 # reset pendulum settings  
 self.length1 = 90  
 self.length2 = 90  
 self.mass1 = 5.0  
 self.mass2 = 5.0  
 self.alpha = np.pi / 2  
 self.beta = np.pi / 2  
 self.d\_alpha = 0.0  
 self.d\_beta = 0.0  
  
 # reset tracing mode  
 self.tracing\_mode = 0  
  
 # reset coordinates  
 self.bob1\_coords = Point(self.center\_coords.get\_x() + self.length1, self.center\_coords.get\_y())  
 self.bob2\_coords = Point(self.bob1\_coords.get\_x() + self.length2, self.bob1\_coords.get\_y())  
  
 # reset tracers  
 self.initiate\_tracers()  
  
 def set\_mass1(self, mass1):  
 self.mass1 = mass1  
  
 def set\_mass2(self, mass2):  
 self.mass2 = mass2  
  
 def set\_length1(self, length1):  
 self.length1 = length1  
  
 def set\_length2(self, length2):  
 self.length2 = length2  
  
 def set\_tracing\_mode(self, code):  
 self.canvas.delete('trace')  
 self.tracing\_mode = code  
 self.initiate\_tracers()  
  
 def calculate\_pendulum\_state(self):  
 self.calculate\_pendulum\_angles()  
 self.calculate\_bob1\_coords()  
 self.calculate\_bob2\_coords()  
  
 def calculate\_pendulum\_angles(self):  
 # components for the 2nd derivative of alpha for first bob  
 component\_1 = (- G \* (2 \* self.mass1 + self.mass2) \* np.sin(self.alpha)) - \  
 (self.mass2 \* G \* np.sin(self.alpha - 2 \* self.beta))  
  
 component\_2 = (2 \* np.sin(self.alpha - self.beta) \* self.mass2) \* \  
 (self.d\_beta \*\* 2 \* self.length2 +  
 self.d\_alpha \*\* 2 \* self.length1 \* np.cos(self.alpha - self.beta))  
  
 component\_3 = (2 \* self.length1 \* self.mass1) + \  
 (self.length1 \* self.mass2 \* (1 - np.cos(2 \* self.alpha - 2 \* self.beta)))  
  
 # second derivative of alpha  
 dd\_alpha = (component\_1 - component\_2) / component\_3  
  
 # components for the 2nd derivative of beta for second bob  
 component\_1 = 2 \* np.sin(self.alpha - self.beta)  
  
 component\_2 = (self.d\_alpha \*\* 2 \* self.length1 \* (self.mass1 + self.mass2)) + \  
 (G \* (self.mass1 + self.mass2) \* np.cos(self.alpha)) + \  
 (self.d\_beta \*\* 2 \* self.length2 \* self.mass2 \* np.cos(self.alpha - self.beta))  
  
 component\_3 = (2 \* self.length2 \* self.mass1) + \  
 (self.length2 \* self.mass2) + \  
 (self.length2 \* self.mass2 \* np.cos(2 \* (self.alpha - self.beta)))  
  
 # second derivative of beta  
 dd\_beta = (component\_1 \* component\_2) / component\_3  
  
 # update d\_angles and angles values  
 self.d\_alpha += dd\_alpha \* DELTA  
 self.d\_beta += dd\_beta \* DELTA  
 self.alpha += self.d\_alpha \* DELTA  
 self.beta += self.d\_beta \* DELTA  
  
 def calculate\_bob1\_coords(self):  
 # set new coordinates of first bob  
 self.bob1\_coords = Point(  
 self.center\_coords.get\_x() + self.length1 \* np.sin(self.alpha),  
 self.center\_coords.get\_y() + self.length1 \* np.cos(self.alpha)  
 )  
  
 # add new coordinates to trace line  
 self.bob1\_tracer.add\_coord(  
 self.center\_coords.get\_x() + self.length1 \* np.sin(self.alpha),  
 self.center\_coords.get\_y() + self.length1 \* np.cos(self.alpha)  
 )  
  
 def calculate\_bob2\_coords(self):  
 # set new coordinates of second bob  
 self.bob2\_coords = Point(  
 self.bob1\_coords.get\_x() + self.length2 \* np.sin(self.beta),  
 self.bob1\_coords.get\_y() + self.length2 \* np.cos(self.beta)  
 )  
  
 # add new coordinates to trace line  
 self.bob2\_tracer.add\_coord(  
 self.bob1\_coords.get\_x() + self.length2 \* np.sin(self.beta),  
 self.bob1\_coords.get\_y() + self.length2 \* np.cos(self.beta)  
 )  
  
 def draw\_traces(self):  
 # delete previous trace line  
 self.canvas.delete('trace')  
  
 # tracing\_mode == 1 || 2  
 if self.tracing\_mode > 0:  
 self.draw\_trace(  
 self.bob2\_tracer.get\_trace(),  
 TRACE\_BOB2\_COLOR  
 )  
  
 # tracing\_mode == 2  
 if self.tracing\_mode > 1:  
 self.draw\_trace(  
 self.bob1\_tracer.get\_trace(),  
 TRACE\_BOB1\_COLOR  
 )  
  
 def draw\_trace(self, trace, color):  
 trace\_length = len(trace)  
 for i in range(trace\_length - 1):  
 self.canvas.create\_line(  
 trace[i].get\_x(),  
 trace[i].get\_y(),  
 trace[i + 1].get\_x(),  
 trace[i + 1].get\_y(),  
 fill=color,  
 tags=['trace']  
 )  
  
 def draw\_pendulum(self):  
 # delete previous pendulum state  
 self.canvas.delete('pendulum')  
  
 # center point of pendulum  
 self.canvas.create\_oval(  
 self.center\_coords.get\_x() - 3,  
 self.center\_coords.get\_y() - 3,  
 self.center\_coords.get\_x() + 3,  
 self.center\_coords.get\_y() + 3,  
 outline='white',  
 fill='white',  
 tags=['pendulum']  
 )  
  
 self.canvas.create\_line(  
 self.center\_coords.get\_x(),  
 self.center\_coords.get\_y(),  
 self.bob1\_coords.get\_x(),  
 self.bob1\_coords.get\_y(),  
 fill='white',  
 tags=['pendulum']  
 )  
  
 # first bob  
 self.canvas.create\_line(  
 self.bob1\_coords.get\_x(),  
 self.bob1\_coords.get\_y(),  
 self.bob2\_coords.get\_x(),  
 self.bob2\_coords.get\_y(),  
 fill='white',  
 tags=['pendulum']  
 )  
  
 self.canvas.create\_oval(  
 self.bob1\_coords.get\_x() - 5,  
 self.bob1\_coords.get\_y() - 5,  
 self.bob1\_coords.get\_x() + 5,  
 self.bob1\_coords.get\_y() + 5,  
 outline=BOB1\_COLOR,  
 fill=BOB1\_COLOR,  
 tags=['pendulum']  
 )  
  
 # second bob  
 self.canvas.create\_oval(  
 self.bob2\_coords.get\_x() - 5,  
 self.bob2\_coords.get\_y() - 5,  
 self.bob2\_coords.get\_x() + 5,  
 self.bob2\_coords.get\_y() + 5,  
 outline=BOB2\_COLOR,  
 fill=BOB2\_COLOR,  
 tags=['pendulum']  
 )  
  
 def initiate\_tracers(self):  
 self.bob1\_tracer = tracer.Tracer()  
 self.bob1\_tracer.add\_coord(self.bob1\_coords.get\_x(), self.bob1\_coords.get\_y())  
  
 self.bob2\_tracer = tracer.Tracer()  
 self.bob2\_tracer.add\_coord(self.bob2\_coords.get\_x(), self.bob2\_coords.get\_y())  
  
 def initiate\_and\_grid\_canvas(self):  
 self.canvas = tk.Canvas(  
 master=self.root,  
 background='black',  
 width=CANVAS\_WIDTH,  
 height=CANVAS\_HEIGHT  
 )  
 self.canvas.grid(  
 row=0,  
 rowspan=10,  
 column=3,  
 columnspan=10,  
 padx=20,  
 pady=20  
 )

**Б.4 Листинг файла timer.py**

import time  
import tkinter as tk  
  
  
class Timer:  
 def \_\_init\_\_(self, root):  
 # set root window  
 self.root = root  
  
 # initiate timer settings  
 self.is\_active = False  
 self.start\_time = time.time()  
  
 # initiate timer and pendulum start/stop buttons  
 self.timer\_label = tk.Label()  
 self.timer\_frame = tk.Label()  
 self.initiate\_and\_grid\_timer()  
  
 def get\_timer\_frame(self):  
 return self.timer\_frame  
  
 def initiate\_and\_grid\_timer(self):  
 # frame for timer and buttons  
 self.timer\_frame = tk.Frame(  
 master=self.root  
 )  
 self.timer\_frame.grid(  
 row=0,  
 rowspan=4,  
 column=0,  
 columnspan=3,  
 padx=20,  
 pady=20  
 )  
  
 self.timer\_label = tk.Label(  
 master=self.timer\_frame,  
 text='0.00 ms',  
 width=20,  
 height=3  
 )  
 self.timer\_label.grid(  
 row=0,  
 rowspan=1,  
 column=0,  
 columnspan=3  
 )  
  
 def start\_timer(self):  
 self.is\_active = True  
 self.start\_time = time.time()  
 self.update\_timer()  
  
 def update\_timer(self):  
 if self.is\_active:  
 delta = time.time() - self.start\_time  
 self.timer\_label['text'] = '{0:.2f}'.format(delta) + " ms"  
 self.root.after(90, self.update\_timer)  
  
 def stop\_timer(self):  
 self.is\_active = False  
  
 def reset\_timer(self):  
 self.timer\_label['text'] = '0.00 ms'

**Б.5 Листинг файла settings.py**

import tkinter as tk  
  
  
class Settings:  
 def \_\_init\_\_(self, root, pendulum):  
 # set root window and pendulum widget  
 self.root = root  
 self.pendulum = pendulum  
  
 # initiate pendulum settings scales  
 self.mass1\_scale = tk.Scale()  
 self.initiate\_and\_grid\_mass1\_scale()  
  
 self.mass2\_scale = tk.Scale()  
 self.initiate\_and\_grid\_mass2\_scale()  
  
 self.length1\_scale = tk.Scale()  
 self.initiate\_and\_grid\_length1\_scale()  
  
 self.length2\_scale = tk.Scale()  
 self.initiate\_and\_grid\_length2\_scale()  
  
 self.tracing\_mode\_scale = tk.Scale()  
 self.initiate\_and\_grid\_tracing\_mode\_scale()  
  
 self.confirm\_button = tk.Button()  
 self.initiate\_and\_grid\_confirm\_button()  
  
 def initiate\_and\_grid\_mass1\_scale(self):  
 # frame for label and scale  
 mass1\_frame = tk.Frame(  
 master=self.root  
 )  
 mass1\_frame.grid(  
 row=10,  
 rowspan=2,  
 column=0,  
 columnspan=3,  
 padx=10,  
 pady=10  
 )  
  
 tk.Label(  
 master=mass1\_frame,  
 text='Масса 1 груза'  
 ).grid(  
 row=0,  
 rowspan=1,  
 column=0,  
 columnspan=3,  
 sticky='w'  
 )  
  
 self.mass1\_scale = tk.Scale(  
 master=mass1\_frame,  
 orient=tk.HORIZONTAL,  
 length=200,  
 from\_=5,  
 to=20,  
 tickinterval=5,  
 resolution=1  
 )  
 self.mass1\_scale.grid(  
 row=1,  
 rowspan=1,  
 column=0,  
 columnspan=3  
 )  
  
 self.mass1\_scale.set(5)  
  
 def initiate\_and\_grid\_mass2\_scale(self):  
 # frame for label and scale  
 mass2\_frame = tk.Frame(  
 master=self.root  
 )  
 mass2\_frame.grid(  
 row=12,  
 rowspan=2,  
 column=0,  
 columnspan=3,  
 padx=10,  
 pady=10  
 )  
  
 tk.Label(  
 master=mass2\_frame,  
 text='Масса 2 груза'  
 ).grid(  
 row=0,  
 rowspan=1,  
 column=0,  
 columnspan=3,  
 sticky='w'  
 )  
  
 self.mass2\_scale = tk.Scale(  
 master=mass2\_frame,  
 orient=tk.HORIZONTAL,  
 length=200,  
 from\_=5,  
 to=20,  
 tickinterval=5,  
 resolution=1  
 )  
 self.mass2\_scale.grid(  
 row=1,  
 rowspan=1,  
 column=0,  
 columnspan=3  
 )  
  
 self.mass2\_scale.set(5)  
  
 def initiate\_and\_grid\_length1\_scale(self):  
 # frame for label and scale  
 length1\_frame = tk.Frame(  
 master=self.root  
 )  
 length1\_frame.grid(  
 row=10,  
 rowspan=2,  
 column=3,  
 columnspan=3,  
 padx=10,  
 pady=10  
 )  
  
 tk.Label(  
 master=length1\_frame,  
 text='Длина 1 нити'  
 ).grid(  
 row=0,  
 rowspan=1,  
 column=0,  
 columnspan=3,  
 sticky='w'  
 )  
  
 self.length1\_scale = tk.Scale(  
 master=length1\_frame,  
 orient=tk.HORIZONTAL,  
 length=200,  
 from\_=50,  
 to=90,  
 tickinterval=10,  
 resolution=1  
 )  
 self.length1\_scale.grid(  
 row=1,  
 rowspan=1,  
 column=0,  
 columnspan=3  
 )  
  
 self.length1\_scale.set(90)  
  
 def initiate\_and\_grid\_length2\_scale(self):  
 # frame for label and scale  
 length2\_frame = tk.Frame(  
 master=self.root  
 )  
 length2\_frame.grid(  
 row=12,  
 rowspan=2,  
 column=3,  
 columnspan=3,  
 padx=10,  
 pady=10  
 )  
  
 tk.Label(  
 master=length2\_frame,  
 text='Длина 2 нити'  
 ).grid(  
 row=0,  
 rowspan=1,  
 column=0,  
 columnspan=3,  
 sticky='w'  
 )  
  
 self.length2\_scale = tk.Scale(  
 master=length2\_frame,  
 orient=tk.HORIZONTAL,  
 length=200,  
 from\_=50,  
 to=90,  
 tickinterval=10,  
 resolution=1  
 )  
 self.length2\_scale.grid(  
 row=1,  
 rowspan=1,  
 column=0,  
 columnspan=3  
 )  
  
 self.length2\_scale.set(90)  
  
 def initiate\_and\_grid\_tracing\_mode\_scale(self):  
 # frame for label and scale  
 tracing\_mode\_frame = tk.Frame(  
 master=self.root  
 )  
 tracing\_mode\_frame.grid(  
 row=10,  
 rowspan=2,  
 column=6,  
 columnspan=3,  
 padx=20,  
 pady=20  
 )  
  
 tk.Label(  
 master=tracing\_mode\_frame,  
 text='Тип отслеживания'  
 ).grid(  
 row=0,  
 rowspan=1,  
 column=0,  
 columnspan=1,  
 sticky='w'  
 )  
  
 self.tracing\_mode\_scale = tk.Scale(  
 master=tracing\_mode\_frame,  
 orient=tk.HORIZONTAL,  
 length=200,  
 from\_=0,  
 to=2,  
 tickinterval=1,  
 resolution=1  
 )  
 self.tracing\_mode\_scale.grid(  
 row=1,  
 rowspan=1,  
 column=0,  
 columnspan=1  
 )  
  
 def initiate\_and\_grid\_confirm\_button(self):  
 self.confirm\_button = tk.Button(  
 master=self.root,  
 text='подтвердить настройки',  
 width=20,  
 command=self.set\_settings  
 )  
 self.confirm\_button.grid(  
 row=13,  
 rowspan=1,  
 column=6,  
 columnspan=3,  
 padx=20,  
 pady=20  
 )  
  
 def set\_settings(self):  
 # set new pendulum settings  
 self.pendulum.set\_mass1(self.mass1\_scale.get())  
 self.pendulum.set\_mass2(self.mass2\_scale.get())  
 self.pendulum.set\_length1(self.length1\_scale.get())  
 self.pendulum.set\_length2(self.length2\_scale.get())  
 self.pendulum.calculate\_bob1\_coords()  
 self.pendulum.calculate\_bob2\_coords()  
  
 # set new tracing settings  
 self.pendulum.set\_tracing\_mode(self.tracing\_mode\_scale.get())  
  
 # update picture on canvas  
 self.pendulum.draw\_pendulum()  
 self.pendulum.draw\_traces()  
  
 def reset\_settings(self):  
 self.mass1\_scale.set(5)  
 self.mass2\_scale.set(5)  
 self.length1\_scale.set(90)  
 self.length2\_scale.set(90)  
 self.tracing\_mode\_scale.set(0)

**Б.6 Листинг файла tracer.py**

from point import Point  
  
  
class Tracer:  
 def \_\_init\_\_(self):  
 self.trace = []  
 self.history\_length = 100  
  
 def add\_coord(self, x, y):  
 if len(self.trace) >= self.history\_length:  
 self.trace = self.trace[1:]  
 self.trace.append(Point(x, y))  
  
 def get\_trace(self):  
 return self.trace

**Б.7 Листинг файла point.py**

class Point:  
 def \_\_init\_\_(self, x, y):  
 self.x = x  
 self.y = y  
  
 def get\_x(self):  
 return self.x  
  
 def get\_y(self):  
 return self.y

**Б.8 Листинг файла consts.py**

BOB1\_COLOR = '#DD1265'  
BOB2\_COLOR = '#6B57FF'  
  
TRACE\_BOB1\_COLOR = '#FE2857'  
TRACE\_BOB2\_COLOR = '#087CFA'  
  
G = 9.8  
DELTA = 0.1  
  
CANVAS\_WIDTH = 400  
CANVAS\_HEIGHT = 400

**ПРИЛОЖЕНИЕ В**

**Сборка и запуск продукта**

**В.1 Сборка и запуск приложения под Windows**

Для сборки и запуска приложения из исходного кода необходимо скачать python версии 3.0 и выше. Для скачивания с официального сайта перейдите по ссылке: <https://www.python.org/downloads/>

Далее необходимо установить python на ваш компьютер. Для этого следуйте инструкциям установщика, который был скачан.

После чего необходимо открыть консоль и установить необходимые дополнительные библиотеки. Откройте консоль и напишите: «pip install numpy».

После завершения установки всех необходимых инструментов перейдите в папку проекта, откройте консоль и напишите: «python src/main.py».

**B.2 Сборка и запуск приложения под Linux (Debian)**

Для сборки и запуска приложения из исходного кода необходимо скачать python версии 3.0 и выше. Для скачивания необходимых пакетов откройте консоль и напишите: «sudo apt-get install python».

Далее необходимо установить необходимые дополнительные библиотеки. Откройте консоль и напишите: «pip install numpy».

После завершения установки всех необходимых инструментов перейдите в папку проекта, откройте консоль и напишите: «python src/main.py».