«Санкт-Петербургский государственный электротехнический университет

«ЛЭТИ» им. В. И. Ульянова (Ленина)»

(СПбГЭТУ «ЛЭТИ»)

Направление подготовки: 09.03.01 «Информатика и вычислительная техника»

Профиль: «Вычислительные машины, комплексы, системы и сети»

Факультет компьютерных технологий и информатики
Кафедра Вычислительной техники

К защите допустить
Заведующий кафедрой
д. т. н., профессор М. С. Куприянов

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА БАКАЛАВРА

Тема: Симулятор физических моделей из школьного курса физики

Студент _____ С. А. Дубенков

Руководитель к. т. н., доцент _____ Я. А. Бекенева

Консультант от кафедры
к. т. н., доцент, с. н. с. _____ И. С. Зуев

Консультант по экономическому
обоснованию, к. э. н., доцент _____ А. С. Прошкина

Санкт-Петербург
2023

Санкт-Петербургский государственный электротехнический университет "ЛЭТИ" им. В. И. Ульянова (Ленина)
(СПбГЭТУ "ЛЭТИ")

Направление Информатика и вычислительная техника
Профиль Вычислительные машины, комплексы, системы и сети
Факультет компьютерных технологий и информатики
Кафедра вычислительной техники

УТВЕРЖДАЮ
Заведующий кафедрой ВТ
д. т. н., профессор
(М. С. Куприянов)
"___" _____ 2023 г.

ЗАДАНИЕ
на выпускную квалификационную работу

Студент Дубенков Семен Александрович Группа № 9308

1. Тема Симулятор физических моделей из школьного курса физики
(утверждена приказом № _____ от _____)
Место выполнения ВКР: Кафедра ВТ

2. Объект и предмет исследования

Настольное приложение для Windows 10, написанное на Java, с использованием фреймворка JavaFX

3. Цель

Цель работы: разработка настольного приложения – редактора физических моделей, обладающего насыщенным графическим интерфейсом, для исследования физических закономерностей тех или иных процессов в интерактивной форме. Для реализации данного проекта используется фреймворк JavaFX, который значительно упрощает создание настольных приложений с широким пользовательским функционалом.

Область применения приложения – это образовательные учреждения, где преподаватели могут использовать его для демонстрации физических моделей и проведения интерактивных уроков. Также приложение может использоваться школьниками для самостоятельного изучения разделов физики.

4. Исходные данные

Исходными данными для разработки являются русскоязычные и англоязычные статьи и видео в сети Интернет, документация к средствам разработки, методические указания.

5. Содержание

Изучение фреймворка JavaFX, создание технического задания и оценка возможностей приложения, изучение способов взаимодействия пользователя с приложением, разработка интерфейса приложения, создание прототипа, разработка функционала моделей, тестирование и стабилизация приложения.

6. Технические требования

Приложение должно быть легко расширяемым для добавления новых модулей и моделей.

В свойствах модулей должны быть прописаны имя и описание; в свойствах моделей должны быть прописаны имя, описание, путь до *.fxml файла, путь до иконки модели, булева переменная необходимости отображения сетки, причем первые 3 параметра являются обязательными.

Описание модели должно включать в себя описание модуля, краткое описание функциональности модели и расчеты, благодаря которым можно провести анализ самостоятельно.

Должно поддерживаться отображение разметочной сетки. Также должна быть добавлена настройка возможности отображения сетки в конфигурационном файле.

Модульное окно «Информация о модели» должно поддерживать парсинг html-кода для форматирования текста и вставки изображений.

Заголовок окна должен сочетаться с цветовой темой приложения. Использовать преимущественно сине-голубые оттенки. Окно приложения должно поддерживать только два размера: посередине монитора с соотношением сторон 11:9 и полноэкранный размер.

Дизайн должен быть адаптивным, т.е. при изменении размера экрана компоненты GUI подстраиваются под него.

Необходимо создать структуру пользовательского интерфейса, состоящую из заголовка, панели инструментов, сцены и строки состояния.

7. Дополнительные разделы

Экономическое обоснование ВКР

8. Результаты

Пояснительная записка, исходный код разработанного приложения на языке Java.

Дата выдачи задания — Дата представления ВКР к защите

«01» сентября 2022 г. — «13» июня 2023 г.

Руководитель

к. т. н., доцент _____ Я. А. Бекенева

Студент _____ С. А. Дубенков

Санкт-Петербургский государственный электротехнический университет “ЛЭТИ” им. В. И. Ульянова (Ленина)
(СПбГЭТУ “ЛЭТИ”)

Направление (09.03.01 «Информатика и вычислительная техника»)
Профиль («Вычислительные машины, комплексы, системы и сети»)
Факультет компьютерных технологий и информатики
Кафедра вычислительной техники

УТВЕРЖДАЮ
Заведующий кафедрой ВТ
д. т. н., профессор
(М. С. Куприянов)
“___” _____ 2023 г.

КАЛЕНДАРНЫЙ ПЛАН
выполнения выпускной квалификационной работы

Тема **Симулятор физических моделей из школьного курса физики**

Студент С. А. Дубенков Группа № **9308**

№ этапа	Наименование работ	Срок выполнения
1	Освоение материалов, необходимых для выполнения выпускной квалификационной работы	01.04 – 10.04
2	Обзор особенностей спецификации платформы JavaFX	11.04 – 20.04
3	Создание технического задания и оценка возможностей приложения	21.04 – 22.05
4	Изучение способов взаимодействия пользователя с приложением	23.04 – 27.04
5	Разработка интерфейса приложения	27.04 – 28.04
6	Создание прототипа	29.05 – 07.05
7	Разработка функционала моделей	08.05 – 25.05
8	Тестирование и стабилизация приложения	26.05 – 31.05
9	Предварительное рассмотрение работы	01.06.2023
10	Представление работы к защите	13.06.2023

Руководитель
к. т. н., доцент _____ Я. А. Бекенева
Студент _____ С. А. Дубенков

РЕФЕРАТ

Пояснительная записка содержит: 89 стр., 8 рис., 36 ист., 8 прил.

Цель работы: Симулятор физических моделей из школьного курса физики.

В выпускной квалификационной работе проводится обзор особенностей разработки приложения с выделением его ключевых особенностей.

Основная цель приложения – помочь школьникам научиться понимать и визуализировать физические законы и явления, используя компьютерную модель.

Объектом работы являются физические модели, включающие в себя различные законы и явления физики, такие как кинематика, динамика и оптика. Предметом работы является разработка графического интерфейса и программного кода, необходимых для визуализации физических моделей и их взаимодействия.

Методология выполнения работы включает в себя изучение основ физики, выбор наиболее важных и интересных моделей для школьников, разработку графического интерфейса и программного кода для визуализации моделей и их взаимодействия, а также тестирование и отладку приложения.

Результат работы – приложение, позволяющее школьникам визуализировать физические законы и явления, используя компьютерную модель. Приложение позволяет изменять параметры и условия взаимодействия, что делает процесс обучения более интересным и эффективным и упрощает понимание изучаемых тем.

ABSTRACT

Purpose of work: Simulator of physical models from the school course of physics.

In the final qualifying work, an overview of the features of the development of the application is carried out, highlighting its key features. The main purpose of the application is to help students learn to understand and visualize physical laws and phenomena using a computer model.

The object of the work are physical models that include various laws and phenomena of physics, such as kinematics, dynamics and optics. The subject of the work is the development of a graphical interface and program code necessary for the visualization of physical models and their interaction.

The methodology for doing the work includes studying the basics of physics, choosing the most important and interesting models for schoolchildren, developing a graphical interface and program code for visualizing the models and their interaction, as well as testing and debugging the application.

The result of the work is an application that allows students to visualize physical laws and phenomena using a computer model. The application allows you to change the parameters and conditions of interaction, which makes the learning process more interesting and effective and simplifies the understanding of the topics being studied.

СОДЕРЖАНИЕ

РЕФЕРАТ 6
ABSTRACT 7
СОДЕРЖАНИЕ 8
ОПРЕДЕЛЕНИЯ, ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ 10
ВВЕДЕНИЕ 12
1 Обзор средств разработки десктопного приложения 14
1.1 Обзор существующих способов цифрового обучения 15
1.1.1 Канал «MinutePhysics» 15
1.1.2 Algodoo 16
1.1.3 VisualMathStart 18
1.2 Выбор технологий для работы 19
1.2.1 Платформы для разработки веб-приложений 19
1.2.2 Qt 20
1.2.3 JavaFX 21
1.3 Сравнение языков C++ и Java 21
1.4 Основные требования для разработки приложения 22
1.4.1 Требования к функциональным характеристикам 23
1.4.2 Требования к дизайну 23
1.4.3 Рекомендуемые технические требования 24
1.4.4 Требования к корректности работы 25
1.5 Выводы 25
2 Реализация основного функционала приложения 26
2.1 Разбиение технического задания на задачи 26
2.1.1 Требования к конфигурационному файлу 26
2.1.2 Требования к дизайну 28
2.2 Разработка интерфейса приложения 30
2.3 Создание прототипа приложения 32
2.3.1 Реализация парсера конфигурационного файла 32

2.3.2 Реализация обработчика моделей 33
2.4 Разработка функционала моделей 36
2.4.1 Модель «Пушечное ядро» 37
2.4.2 Модель «Груз с пружиной» 39
2.4.3 Модель «Преломление через линзу» 41
2.5 Пример расширения функционала и кодовой базы приложения 43
3 Описание использования приложения 45
3.1 Взаимодействие с конфигурационным файлом 45
3.2 Взаимодействие с приложением 46
4 Экономическое обоснование дипломного проекта 50
4.1 Расчет затрат на оплату труда 50
4.2 Расчет накладных расходов 52
4.3 Расходы на материалы 53
4.4 Издержки на амортизацию ПК и оргтехники 53
4.5 Расходы на услуги сторонних организаций 54
4.6 Себестоимость выполнения дипломной работы 54
4.7 Вывод 55
ЗАКЛЮЧЕНИЕ 56
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 57
ПРИЛОЖЕНИЕ А. Полноэкранное отображение приложения 61
ПРИЛОЖЕНИЕ Б. Описание классов 62
ПРИЛОЖЕНИЕ В. Код класса MainController 63
ПРИЛОЖЕНИЕ Г. Код класса AbstractModelController 68
ПРИЛОЖЕНИЕ Д. Код класса Model 70
ПРИЛОЖЕНИЕ Е. Код класса CannonballController 73
ПРИЛОЖЕНИЕ Ё. Код класса WeightWithSpringController 80
ПРИЛОЖЕНИЕ Ж. Код класса RefractionThroughLensController 85

ОПРЕДЕЛЕНИЯ, ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

ПО – программное обеспечение.

ОС – операционная система.

ТЗ – техническое задание.

МФУ – многофункциональное устройство.

Модуль – раздел физики, в котором хранятся физические модели

Модель – упрощенное представление реальной физической системы, позволяющее прогнозировать ее поведение и результаты опытов.

Фреймворк – ПО, объединяющее различные компоненты большого программного проекта.

Десктоп – основное окно графической среды пользователя.

Десктопный – соотносящийся по значению с существительным десктоп.

Drag-and-drop – метод взаимодействия с пользовательским интерфейсом, в котором объекты могут быть перемещены мышью из одного места в другое.

Баг – ошибочное поведение программы или системы, которое приводит к нежелательным результатам.

Кастомизация – процесс изменения программного обеспечения для соответствия конкретным потребностям пользователя.

Front-end – часть разработки, отвечающая за создание пользовательского интерфейса и взаимодействие с пользователем в браузере.

Back-end – часть разработки, отвечающая за обработку запросов, работу с базой данных, обеспечение функциональности, невидимой для пользователя.

Кроссплатформенность – способность ПО работать на нескольких ОС без необходимости значительных изменений или перекомпиляции.

JVM (java virtual machine) – виртуальная Java-машина.

GUI (graphical user interface) – графический пользовательский интерфейс.

HTML (Hypertext Markup Language) – язык гипертекстовой разметки.

Скриншот – статичное изображение экрана компьютера, зафиксированное в определенный момент времени.

Виджет – графический компонент пользовательского интерфейса, добавляющийся внутрь рабочего окна или другого виджета для предоставления визуального и интерактивного взаимодействия.

Парсинг – сбор и структурирование информации для дальнейшего ее использования.

Контроллер – класс-обработчик, используемый в архитектуре приложения для обработки данных и посредничества между компонентами модели и ее представлением.

ВВЕДЕНИЕ

Изучение физики и других точных наук имеет важное значение для образования и подготовки будущих профессиональных кадров. В связи с ограничениями по проведению очных лабораторных в школах и дальнейшей цифровизации образования, симуляторы становятся более актуальными и необходимыми для обучения школьников точным наукам, в частности, физике. Благодаря дистанционному обучению образование стало более доступным.

Целью работы является разработка приложения для исследования физических закономерностей тех или иных процессов в интерактивной форме.

Оно знакомит с законами окружающего мира, обучает разделам физики из школьной программы, помогает закреплять материал, пройденный в классе. В конечном счете, приложение должно привить ученику любопытство и интерес к науке.

Объект и предмет исследования: настольное приложение для Windows 10 [15], написанное на Java [29], с использованием фреймворка JavaFX[5].

В 2007 году Sun Microsystems продемонстрировала на конференции технологию под названием JavaFX. Через год, в 2008 году, была выпущена версия 1.0, которая включала такие полезные инструменты, как компилятор и среда выполнения JavaFX, язык программирования JavaFX Script и различные библиотеки для графики, мультимедиа и веб-разработки. С тех пор эта платформа активно развивается.

Программный комплекс стремится исправить недостатки, связанные с непониманием различных тем и недостаточной подготовкой.

Преимущества использования моделирования в классе: возможность дистанционного обучения, более активное участие в процессе, а также удобство проведения экспериментов без необходимости использования дорогостоящего оборудования или материалов.

Для достижения цели требуется: разобраться в потребностях целевой аудитории; выбрать физические модели, которые максимально наглядно де-

монстрируют физические закономерности и которыми можно интуитивно управлять; собрать информацию для анализа; разработать архитектуру, используя паттерны проектирования для более удобной разработки и масштабирования проекта; отладка программных компонентов.

В первом разделе приводится обзор средств разработки десктопного приложения и выбор технологий для работы.

Во втором разделе описаны разработка приложения и реализация задач из технического требования.

В третьем разделе приводятся описание функционала и инструкция по использованию приложения.

В четвертом разделе представлен дополнительный раздел по экономическому обоснованию ВКР.

1 Обзор средств разработки десктопного приложения

В этом разделе рассмотрен набор задач, необходимых для разработки ПО, а также проведено сравнение различных технологий и способов разработки сложных систем.

Разработка симулятора физических моделей поможет ученикам понимать различные научные концепты в интерактивной форме. Приложение должно быть разработано с учетом потребностей и способностей школьников, чтобы оно было для них удобным и понятным. Также важно убедиться, что моделирование соответствует школьной программе и охватывает необходимые научные понятия. Наконец, важно учитывать масштабируемость проекта, поскольку в будущем может возникнуть потребность в обновлении или модификации, чтобы отобразить изменения в школьной программе или достижения в области информационных технологий.

Ниже приведено несколько задач, которые должны быть выполнены в ходе работы:

- разбор аналогов приложения и выявление их недостатков, которые можно решить в текущем проекте;
- определение технологического стека: он будет зависеть от поставленных задач и сложности поддержания программного продукта;
- определение разделов, научных концепций, которые необходимо охватить, и навыков, которым учащиеся должны будут обучиться;
- проектирование симуляции: включает в себя создание и расстановку графических компонентов, разработку пользовательского интерфейса и программирование симуляции;
- тестирование и уточнения моделирования: после разработки основной (функциональной) части важно тщательно протестировать приложение на критические баги, мешающие ходу работы.

1.1 Обзор существующих способов цифрового обучения

Методы цифрового обучения становятся все более популярными из-за их доступности, экономической эффективности и способности предоставлять персонализированный опыт обучения. Образовательные игры и симуляторы могут обеспечить увлекательное и интерактивное обучение, позволяя учащимся экспериментировать с различными сценариями и видеть последствия своего выбора в безопасной и контролируемой среде. В целом, разработка приложения для моделирования физических процессов является важным вкладом в сферу образования, поскольку она обеспечивает альтернативу традиционному образованию, а также предлагает ученикам и учителям увлекательный опыт обучения.

Главная задача обучения – донести мысль до ученика и сформировать профессиональную и компетентную с своей области личность, способную к самоопределению в условиях быстро меняющегося общества.

Еще одним плюсом цифрового обучения является относительная дешевизна, удобство проведения экспериментов без необходимости использования дорогостоящего оборудования, а также гибкость моделирования, которое позволяет учащимся исследовать различные сценарии и видеть, как различные переменные влияют на исход эксперимента.

Распространенными практиками цифрового обучения являются демонстрация видеороликов на платформе YouTube [14], а также приложения, выпускаемые под эгидой крупных образовательных учреждений, а именно:

1.1.1 Канал «MinutePhysics»

«MinutePhysics» [1] – популярный канал, специализирующийся на коротких, анимационных видео, и объясняющий сложные физические концепции в доступном для целевой аудитории виде. Канал ведется физиком, который использует нарисованную от руки анимацию и простой язык, чтобы разбивать сложные идеи на идеи поменьше.

Контент на MinutePhysics покрывает широкий спектр тем: от основ ньютоновской физики до квантовой механики и теории относительности.

В дополнение к объяснению физики, канал также занимается более широкими темами, такими как философия, экономика и дискретная математика. Видео длятся всего несколько минут, что позволяет легко их усваивать и распространять в социальных сетях.

Автор канала рисует анимации от руки, из-за чего между выходом видео проходит большой интервал времени: видео выходят приблизительно раз в месяц. Компьютерная симуляция должна решить проблему: создание графических компонентов и их связка с действиями пользователя обходится гораздо быстрее и дешевле, а еще такие модели можно поддерживать довольно долгое время.

Когда ученик просматривает видеоролики с демонстрацией модели, у него может развиться невнимательность из-за отсутствия вовлеченности в процесс: он не может манипулировать происходящим и изучать материал на основе личного опыта. С помощью симулятора физических процессов можно создать удобный пользовательский интерфейс взаимодействия с простотой использования.

1.1.2 Algodoo

Algodoo [2] – это программное обеспечение для моделирования физических явлений, представляющее из себя двумерную песочницу, средство обучения и компьютерную игру. На выбор дается большое количество моделей, разработанных командой разработчиков и дизайнеров.

Приложение доступно на платформах Windows [15] и MacOS [16] и используется преподавателями, студентами и любителями по всему миру для лучшего понимания физических концепций.

Одной из основных особенностей приложения является удобный интерфейс, позволяющий манипулировать различными объектами и средами; также оно поддерживает функционал drag-and-drop, что позволяет пользова-

телям выбирать объекты из библиотеки и перемещать их в текущую среду выполнения.

Ещё одной ключевой особенностью является способность моделировать различные физические явления, такие как гравитация, столкновения и трение. Физический движок имитирует поведение объектов в среде моделирования, что позволяет пользователям видеть взаимодействие объектов друг с другом и средой в реальном времени.

Хотя у Algodoo много сильных сторон, у этого программного продукта есть свои недостатки. Программа имеет сильные ограничения в 3D симуляции, и на этой платформе нецелесообразно создавать трехмерную модель из-за ограничений физического движка – это возможно только после установки стороннего ПО.

Последнее обновление приложения было выпущено более 10 лет назад, из-за чего приложение уже давно потеряло свою актуальность; оно уже не поддерживается на новых версиях операционных систем, из-за чего могут возникать проблемы оптимизации и баги.

Приложение было написано на C++[17], и из этого вытекает сразу несколько проблем: во-первых, это не кроссплатформенный язык, из-за чего для каждой платформы приходится писать отдельную версию приложения с тем функционалом, который доступен на этой ОС; во-вторых, C++ – очень сложный язык, который поддерживает указатели, ручное управление памятью, не имеет встроенной поддержки потоков и диспетчеризации, в нем отсутствует обратная совместимость – всё перечисленное означает, что безопасность выполнения программы лежит на ответственности программиста, из-за чего такой программный продукт поддерживать куда сложнее, если бы он был написан на языке более высокого уровня.

Так как это жанр приложения – песочница, то пользователю предлагается просто манипулировать объектами и средой, в которой они находятся, из-за чего у учащегося отсутствует понимание того, как это всё работает:

приложение не предлагает ни научную базу, ни формулы, по которым делались расчеты симуляции.

Закрытый исходный код: его недостатки изложены в пункте 1.1.3.

1.1.3 VisualMathStart

VisualMathStart[3] – программный продукт, предоставляющий доступ к набору программ для моделирования физических процессов и затрагивающий большинство разделов школьного курса физики. В набор входят: газодинамический, тепловой, гравитационный, электростатический симуляторы, программы для моделирования маятников, колебаний струн и продольных колебаний, тепловой энергии. Каждая программа имеет свои уникальные особенности и предназначена для моделирования конкретных физических процессов.

Первое, что попадается на глаза – непримечательный пользовательский интерфейс: такой же, как на большинстве офисных программ; отсутствует цветовая тема и стили. Также отсутствует разметочная сетка, по которой можно оценить масштабы.

В приложении присутствуют разделы, далекие от школьной программы физики: это мешает пониманию физических концепций и нагружает учащихся.

Программа состоит из закрытого кода, и это накладывает несколько ограничений:

- ограниченная кастомизация – поскольку исходный код недоступен, пользователи не могут модифицировать ПО в соответствии со своими конкретными потребностями;
- угрозы безопасности – без доступа к исходному коду сложно выявить и устранить уязвимости в коде приложения, из-за чего выявить критическую ошибку на ранней стадии не представляется возможным;
- зависимость от производителя: пользователи ПО с закрытым исходным кодом зависят от производителя в отношении поддержки, обновлений и

исправлении ошибок; если поставщик выходит из бизнеса или прекращает поддержку ПО, пользователи останутся с потенциально уязвимым программным продуктом;

• отсутствие прозрачности – без доступа к исходному коду пользователи не смогут убедиться, что приложение выполняет ровно те функции, о которых оно заявляет, и что оно не содержит вредоносного кода;

• ограниченные возможности: такое ПО разрабатывается ограниченной группой лиц, что может ограничить функционал приложения по сравнению с ПО с открытым исходным кодом, которое может получить преимущество от более широкого сообщества.

1.2 Выбор технологий для работы

Для достижения поставленных целей необходимо тщательно продумать стек технологий для проекта: выбор технологий для разработки зависит от сложности проекта, требуемого функционала, ожидаемой производительности и опыта работы разработчика. Для текущего проекта важно, чтобы его было легко поддерживать спустя долгое время, легкость вхождения в бизнес-логику, насыщенный пользовательский интерфейс, возможность поддержки большого количества библиотек.

Существует несколько способов решения текущей задачи.

1.2.1 Платформы для разработки веб-приложений

В данный список входят такие фреймворки как React [18], Angular [19], Vue [20] и другие, которые основаны на JavaScript [21] и используются для front-end разработки. Их можно использовать в самых разных проектах, от небольших до крупномасштабных приложений.

Однако не стоит использовать такие решения, и вот почему:

• ограниченная гибкость: такие фреймворки имеют определенные ограничения, которые могут не подходить для всех проектов или групп разработчиков – они ограничивают возможность настройки и расширения кодовой базы в соответствии с текущими требованиями;

• управление зависимостями: приложения, написанные на основе множества фреймворков, часто зависят от других библиотек и инструментов, что может усложнить процесс разработки;

• удаленный доступ: данные приложения запускаются на удаленных серверах, из-за чего производительность приложения зависит от производительности серверов, на которых приложение загружено, а также требуется постоянный доступ в Интернет, из-за чего невозможно пользоваться ПО без доступа к глобальной сети;

• используемость: большинство фреймворков для front-end разработки предназначены только для отображения контента в браузере, научная база используется довольно редко; там нет большинства функций, необходимых для реализации симуляции моделей, к тому же, в основном, для back-end разработки сайтов используется JavaScript, который сильно ограничен в функциональности, если и делать back-end, то тогда придется подключать сборщики проектов и языки более низкого уровня.

1.2.2 Qt

Qt[4] – это популярный кроссплатформенный фреймворк для написания десктоп-приложений и мобильных разработок, написан на языке C++. В нем присутствует большое количество полезных и многофункциональных модулей для разработки больших проектов, включая поддержку GUI, работы в сети, трехмерной графики. Однако и у такого ПО есть свои недостатки:

• высокий порог вхождения: для изучения фреймворка и его особенностей могут понадобиться большие время и усилия, чтобы начать использовать фреймворк для работы, к тому же платформа предоставляет не только создание GUI, но и собственные контейнеры, структуры данных, пространства классов и прочее;

• отсутствие кроссплатформенности исполняемых файлов: разработчики платформы уверяют, что Qt не зависит от исполняемой ОС, однако в ходе разработки придется использовать множество макросов, которые исполняют-

ся только на одной ОС, поэтому программе приходится узнавать параметры ОС прямо во время исполнения, из-за чего код становится менее «чистым»;

• использование C++: описание его недостатков представлено в п. 1.1.2.

1.2.3 JavaFX

JavaFX[5] – это популярная платформа для создания многофункциональных кроссплатформенных графических пользовательских интерфейсов на языке Java [29]. Она будет использоваться в текущей работе по следующим причинам:

• простота использования: фреймворк предоставляет только создание графических компонентов, сама же логика пишется исключительно на языке Java, то есть для входа в разработку достаточно знать язык Java;

• поддержка мультимедиа, 3D-графики и анимаций, позволяющих создавать максимально отзывчивый интерфейс;

• открытый исходный код: он постоянно обновляется сообществом разработчиков, это означает, что сложно будет не найти модуль, который будет подходить под определенную задачу;

• кроссплатформенность: в отличие от Qt, исполняемые файлы Java можно запустить на любой ОС, поддерживающие JVM, что позволяет запускать приложение на первом попавшемся компьютере;

Таким образом, JavaFX – это мощная и гибко настраиваемая среда графического интерфейса, которая подходит для создания современных приложений с богатыми графическими возможностями.

1.3 Сравнение языков C++ и Java

В графе описания характеристик языка Java курсивом обозначены преимущества, которые выдвигают его на первый план, в графе описания характеристик языка C++ курсивом обозначены недостатки, из-за которых он был выдвинут на задний план.

Java:

- Ограниченная поддержка указателей
- Компилируемый и интерпретируемый язык
- *Автоматическое управление памятью*
- *Имеет встроенную поддержку потоков*
- *Поддерживает JavaDoc [30]*
- *Безопасность использования памяти и потоков*
- *Строго типизированный язык*
- *Огромное количество фреймворков и библиотек*
- *Продолжительная поддержка ПО*

C++:

- Мощная поддержка указателей
- Компилируемый язык
- *Ручное управление памятью*
- *Не имеет встроенной поддержки потоков*
- *Не имеет собственных инструментов введения документации*
- *Безопасность на ответственности программиста*
- *Не строго типизированный язык*
- *Малое количество фреймворков*
- *Отсутствие обратной совместимости*

Таким образом, для более комфортной разработки и уменьшения создания приложения был выбран язык Java.

1.4 Основные требования для разработки приложения

Техническое задание для выпускной квалификационной работы состоит в разработке приложения для симуляции физических моделей, используя среду разработки JavaFX и язык Java. Приложение должно быть легко рас-

ширяемым для добавления новых модулей и моделей. Программе должен быть предоставлен доступ к конфигурационному файлу, в котором будет можно модифицировать перечисления модулей и моделей, а также исправлять информацию об их свойствах.

1.4.1 Требования к функциональным характеристикам

Ниже представлен список функциональных требований для приложения.

- Исходный код программы должен быть написан на Java, в нем необходимо использовать объектно-ориентированный подход.
- Для управления приложением должна использоваться исключительно компьютерная мышь.
- Должно поддерживаться отображение разметочной сетки, в которой размер ячейки равен 100*100 пикселей. Также должна быть добавлена настройка возможности отображения сетки в конфигурационном файле.
- В свойствах модулей должны быть прописаны имя и описание; в свойствах моделей должны быть прописаны имя, описание, путь до *.fxml файла, путь до иконки модели, булева переменная необходимости отображения сетки, причем первые три параметра являются обязательными.
- Описание модели должно включать в себя описание модуля, в котором модель находится, краткое описание функциональности модели и расчеты, благодаря которым можно провести анализ самостоятельно.
- Модульное окно “Информация о модели” должно поддерживать парсинг html-кода для форматирования текста и вставки изображений.
- Для увеличения производительности приложения загружать модели не при загрузке программы, а при первом запросе от пользователя.

1.4.2 Требования к дизайну

Ниже представлен список требований для интерактивного и простого в использовании дизайна приложения.

• Заголовок окна должен сочетаться с цветовой темой приложения. Использовать преимущественно сине-голубые оттенки.

• Окно приложения должно поддерживать только два размера: посередине монитора с соотношением сторон 11:9 и полноэкранный размер.

• Дизайн должен быть адаптивным, т.е. при изменении размера экрана компоненты GUI подстраиваются под него.

• Необходимо создать структуру пользовательского интерфейса, состоящую из заголовка, панели инструментов, сцены и строки состояния.

• Заголовок. В данном блоке необходимо расположить логотип приложения, пункты меню “Вид” и “Справка”, название приложение строго посередине заголовка, а также кнопки сворачивания, разворачивания и закрытия окна.

• Панель инструментов должна содержать кнопку отображения разметочной сетки, кнопку запуска демонстрации модели, выпадающий список с наименованиями модулей, а также кнопку, выдающую информацию о текущей модели, включая информацию о модуле.

• Сцена. На ней должна отображаться модель, которой можно манипулировать с помощью действий мыши, либо через панель настроек, находящейся справа и открывающейся при нажатии кнопки.

• Строка состояния – виджет, находящийся внизу сцены, сообщающий о каких-либо событиях, произошедших в программе.

1.4.3 Рекомендуемые технические требования

Ниже изложен список технических требований для запуска приложения.

• Операционная система: Windows 10 64-bit.

• Версия Java: 17 LTS и выше.

• Версия JavaFX: 17 и выше.

• Процессор: Intel Core i3 1115G4 и выше или эквивалентный по характеристикам от AMD.

- Оперативная память: 8 Гб.
- Свободное место на диске: 1 Гб.

1.4.4 Требования к корректности работы

Во время работы программы не должно вылетать критических ошибок и исключений, прерывающих ход приложения при запуске на Windows. Для достижения такого результата важно правильно управлять потоками, а также обеспечить безопасность при работе с ними.

1.5 Выводы

В данном разделе был проведен обзор существующих способов цифрового обучения, а также был произведен анализ существующих решений для текущей задачи.

Для удобной реализации проекта был выбран язык Java и платформа JavaFX.

2 Реализация основного функционала приложения

В этом разделе будет представлено описание процесса разработки и реализации основных возможностей приложения, включая создание физических моделей и их свойств, а также разработку пользовательского интерфейса и функциональности приложения. В разделе представлен подробный обзор технических аспектов проекта с выделением основных проблем и решений, возникших в процессе реализации.

2.1 Разбиение технического задания на задачи

Для выполнения технического задание в срок необходимо грамотно разбить его на технические задачи, чтобы определить их сложность, в какие временные интервалы их можно выполнить и превратить абстрактные требования в конкретные задачи. Разбиение на задачи помогает сделать работу слаженной и продуктивной, а также устраняет недопонимания между заказчиком и разработчиком.

2.1.1 Требования к конфигурационному файлу

Для того чтобы пользователю было удобно редактировать информацию о приложении и его содержимом, пользователю была представлена возможность редактировать ее, не заходя в исходный код программы. Для решения этой проблемы был создан конфигурационный файл с общей информацией о программе, описанием модулей, описанием моделей и их свойствами: они необходимы для определения поведения моделей.

Модульное окно «Информация» поддерживает парсинг html-кода, благодаря чему в него можно вставлять тексты разных стилей, а также изображения.

В конфигурационный файл добавлены следующие свойства:

• modules: этот ключ необходим для доступа к перечислению модулей, помещенное в квадратные скобки. Каждая модель относится к модулю по общей приставке, то есть принадлежность модели к модулю определяется по приставке до первой заглавной буквы. Приставки должны быть уникальными

и указывать на конкретный модуль. Для редактирования или добавления информации о модуле необходимо использовать ключ modules.<Module>, где module – ключ с перечислением всех модулей, <Module> – конкретный модуль из этого перечисления, внутри себя содержит ключи moduleName и moduleDescription;

- moduleName: ключ, необходимый для определения названия модуля;
- moduleDescription: ключ, необходимый для определения описания модуля;
- models: этот ключ необходим для доступа к перечислению моделей, помещенное в квадратные скобки. По приставке до первой заглавной буквы определяется принадлежность модели к модулю. Для редактирования или добавления информации о модели необходимо использовать ключ models.<Model>, где model – ключ с перечислением всех моделей, <Model> – конкретная модель из этого перечисления, внутри себя содержит ключи modelName, modelDescription, modelFilePath, iconPath и isGridNeeded;
- modelName: ключ, необходимый для определения названия модели;
- modelDescription: ключ, необходимый для определения описания модели;
- modelFilePath: ключ, в котором указывается путь до файла, определяющего структуру для создания пользовательского интерфейса, отдельного от логики приложения;
- iconPath: ключ, указывающий на путь до иконки модели. Если этот параметр указывается в конфигурации явно, то иконка модели появляется во вкладке модели над сценой и в диалоговом окне, иначе вместо нее показывается иконка приложения, а во вкладке модели ничего не указывается;
- isGridNeeded: параметр, указывающий, необходима ли для текущей модели разметочная сетка: если параметр указывается явно, тогда в панели инструментов активируется кнопка отображения сетки и пользователь смо-

жет пользоваться сеткой для замеров расстояний, иначе кнопка остается неактивной;

• programDescription: ключ, определяющий описание программы, отображающееся пользователю.

Для реализации представленного функционала был создан класс-парсер, который был выполнен в стиле шаблона проектирования Синглтон[22], предоставляющий возможность единожды создать объект класса и использовать ссылку на него, вызывая каждый раз метод получения экземпляра. Описание функционала класса представлено в разделе 2.3.1.

2.1.2 Требования к дизайну

К сожалению, с помощью программных средств Java [29] нельзя изменить цвет заголовка окна, чтобы он сочетался с цветовой темой приложения, так как это в значительной степени зависит от операционной системы: она решает, как отображаются заголовки и границы. Поэтому для решения проблемы этот заголовок необходимо убрать, добавить свой заголовок, придать ему границы, стиль и обработку событий при нажатии и его перемещении, а затем нужно добавить кнопки свертывания окна, увеличения на весь экран и закрытия приложения. Для реализации такого функционала был использован вспомогательный материал с системы вопросов и ответов о программировании StackOverFlow [26]. Также должны быть добавлены пункты меню, один из которых управляет видимостью сетки, второй отображает информацию о программном комплексе и о текущей модели.

Разметочная сетка является удобным инструментом для анализа изменения расстояния в условных единицах. Для отображения сетки будет использоваться отдельное полотно, видимость которого будет контролироваться из панели инструментов. Оно не должно мешать обработке событий при взаимодействии мыши.

В программе отсутствует возможность произвольного изменения размера окна – это было сделано с целью упрощения разработки и избеганию

обработки действий при изменении размер экрана: могут быть случаи, когда при изменении размеров рабочей области меняются координаты объектов, поэтому их надо будет пересчитывать заново.

Структура пользовательского интерфейса должна состоять из заголовка, панели инструментов, сцены и строки состояния. Принцип работы заголовка был описан ранее.

Панель инструментов включает в себя кнопку управления видимостью разметочной сетки, которая может быть неактивна, если пользователь не задавал в конфигурационном файле такую настройку; кнопку обработки начала исполнения модели, которая после начала выполнения и до конца действия модели должна оставаться неактивной для избегания коллизий действий; выпадающий список модулей, по выбору из которых открывается первая модель из выбранного модуля; кнопка «Информация о модели», по нажатию которой всплывает диалоговое окно с подробной информацией о текущей модели, которую можно редактировать в конфигурационном файле. При изменении размеров окна последняя кнопка остается в крайнем правом углу, остальные – на другом конце экрана.

Объект сцены представляет из себя набор вкладок, в которых находятся отображения моделей. Их можно переключать при нажатии на вкладку, тогда же должна загружаться модель, чтобы она не загружалась при инициализации программы, чтобы она не занимала места в оперативной памяти, если бы она ни разу не использовалась за время выполнения программы.

Строка состояния должна отображать изменения, произошедшие в программе, о которых необходимо знать пользователю: если ситуация некритичная, сообщение отображается именно там, иначе всплывает диалоговое окно, блокирующее действия пользователя. Критичность ситуации определяет разработчик модели. Также оно необходимо для подсвечивания подсказок при наведении мыши на объект.

2.2 Разработка интерфейса приложения

Для реализации окна приложения использовались графические шаблоны, представленные фреймворком JavaFX [5]. Полотно BorderPane [31] позволяет располагать виджеты по краям экрана и в центре: вверху расположено полотно, в котором вертикально находятся виджеты заголовка и панели инструментов, посередине находится полотно StackPane [32], позволяющее располагать свои дочерние элементы поверх друг друга, то есть в виде стека; в нем располагаются приветствующая надпись, выдвигающаяся панель настроек, панель разметочной сетки и полотно TabPane [33], располагающее в себе отображения моделей, добавленных разработчиком программно.

Заголовок расположен в полотне, располагающем свои виджеты горизонтально: в него входят иконка приложения, пункты меню, название приложения и кнопки управления видимостью окна. Надпись с названием приложения была помещена в такой же виджет, чтобы она располагалась по центру родительского виджета. Аналогично были расположены кнопки управления видимостью окна, чтобы они располагались в правом углу заголовка.

Панель инструментов была расположена в таком же полотне. Стоит отметить, что для расположения кнопки «Информация о модели» требовалось поместить ее в отдельное полотно и задать в его настройках расположение всех его элементов справа с конца.

В центральном полотне StackPane элементы GUI расположены в следующем порядке: приветствующая надпись, панель настроек, панель с разметочной сеткой и полотно, на котором будут располагаться модели, которые можно переключать при нажатии на вкладку. Особенность этого полотна заключается также в том, что все элементы внутри него пытаются занять максимальный размер, что очень удобно для отображения всех элементов центрального окна.

И наконец, строка состояния представляет из себя полотно HBox [34], в котором элементы располагаются в горизонтальном порядке. В него была

помещена надпись, меняющая свое содержимое в зависимости от событий в программе. В это полотно можно добавить панель быстрых настроек, ее реализацию можно отложить до первого финансирования.

Для формирования стилей GUI-компонентов необходимы глобальные методы и константы, но так как в Java в принципе невозможно существование переменных и методов вне классов, пришлось создать классы Global и Constants со статическими константными переменными и методами.

В классе Constants присутствуют следующие атрибуты:

- MIN_WIDTH – минимальная ширина экрана;
- MIN_HEIGHT – минимальная высота экрана;
- THEME_COLOR – цветовая тема приложения;
- BG_THEME_COLOR_PATTERN – паттерн отображения заднего фона;
- BG_THEME_PATTERN – паттерн отображения заднего фона;
- DEBUG – глобальная переменная, определяющая, выводить ли данные в консоль;
- MAIN_ICON_IMAGE – заглавная иконка приложения;
- HIDE_DELAY – время отображения пользовательского сообщения в строке состояния в миллисекундах;
- PIXELS_PER_UNIT – количество пикселей на одну условную единицу;
- g – гравитационная постоянная.

В классе Global есть метод getCSSThemeColor, определяющий цветовую гамму по входному коэффициенту: если он меньше 0, цвет виджета становится более насыщенным, если он больше 0, то менее насыщенным, иначе не меняется. Цвет всех компонентов зависит от константы Constants. THEME_COLOR. Цветовая гамма настроена таким образом, чтобы палитра цветов не казалась слишком резкой и контрастной, был осуществлён плавный переход оттенков.

Графическая схема приложения представлена на рисунке 2.1.

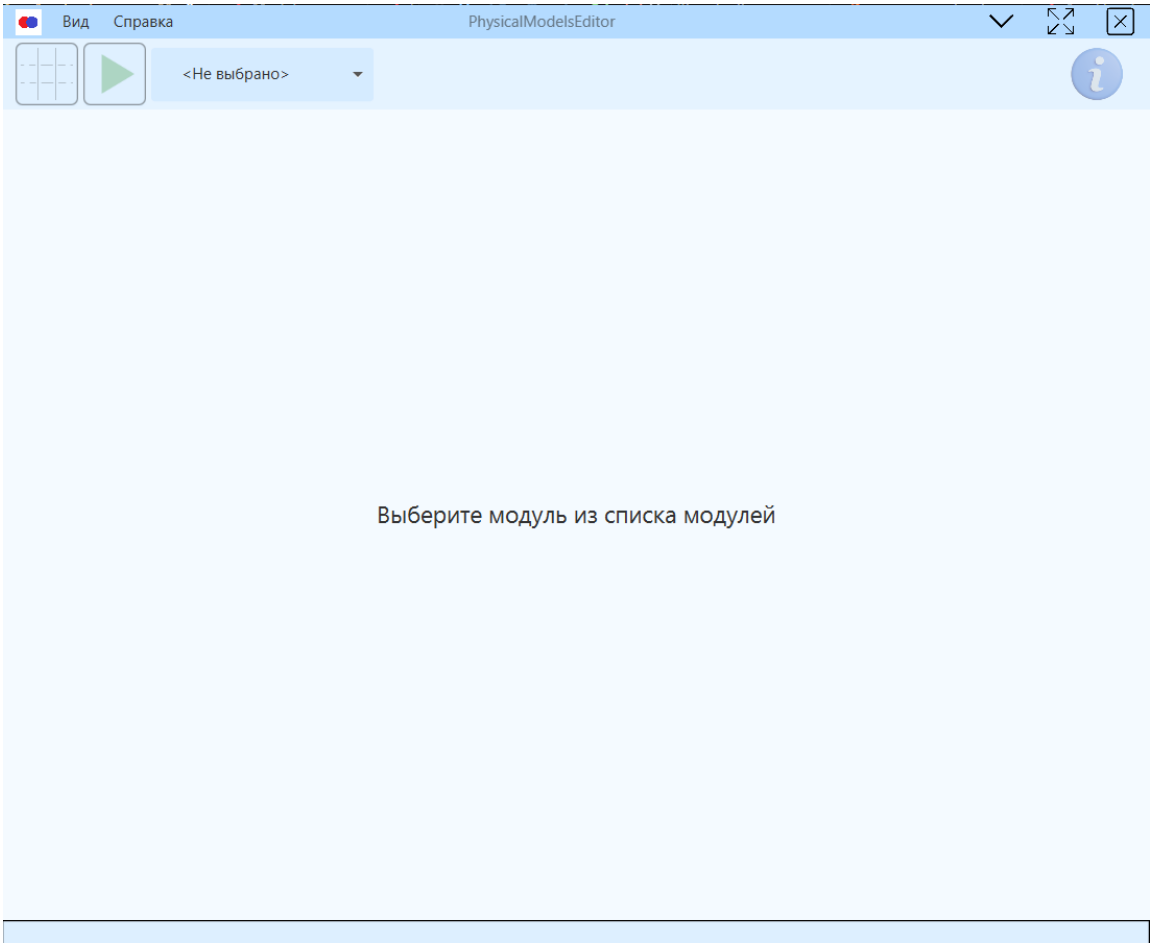

Рисунок 2.1 – Внешний вид приложения

Таким образом, внешний вид приложения настроен, можно приступать к реализации функционала приложения.

2.3 Создание прототипа приложения

В этом подразделе описан принцип работы приложения, а именно: реализацию парсера конфигурационного файла; обработку массива физических моделей.

2.3.1 Реализация парсера конфигурационного файла

Как говорилось ранее, класс-парсер реализован с помощью шаблона проектирования Синглтон для доступа к единственному экземпляру через статический метод его получения из любой точки программы. Конструктор класса вызывается при первом вызове метода getInstance, далее будет возвращаться готовая ссылка.

В конструкторе загружается ационный файл, затем к тексту из него применяются стили, а именно:

• обозначение формул: для такого текста использовался шрифт Comic sans MS [35], для указания формул в файле используется символ ‘`’;

• обозначение пути к изображению: для указания пути к изображению используется символ ‘$’.

Далее происходит инициализация модулей: сначала из свойства modules достаются названия модулей, затем значения из этого перечисления используются в качестве ключей для поиска информации о конкретных модулях; после парсинга модулей информация о них добавляется в структуру mModulesMapList.

Аналогично происходит инициализация моделей, так как их структура очень похожа; после парсинга моделей информация о них добавляется в структуру mModelsMapList.

В конце из конфигурационного файла достается информация о программном комплексе.

Информация, добытая из конфигурационного файла, используется в конструкторе ModuleFactory, в конструкторе Model, а также при установке описания программы в методе InfoDialog.setDescription.

2.3.2 Реализация обработчика моделей

Результатом текущего подраздела является каркас приложения без моделей, который может обрабатывать их при поступлении.

Главным контроллером является класс MainController, он обрабатывает события приложения над всеми моделями, а также регламентирует порядок инициализации различных компонентов. Например, метод initGUI инициализирует GUI-компоненты и задает их параметры, а именно их цветовую гамму и присваивает им обработчики действий. Также в нем идет инициализация объектов классов InfoDialog, SettingsToolbar, MarkingGrid и AppHeader. Ме-

тод initFactories инициализирует фабрики модулей типа ModuleFactory, информация из которых берется из конфигурационного файла.

Класс ModuleFactory является представлением модуля, хранящем внутри себя тематические модели. Внутри его конструктора происходит инициализация моделей класса Model и размещение их во внутренней структуре. Этот класс предоставляет доступ к моделям через метод modelAt, также возможно получение текущей модели через метод getCurrentModel.

Класс Model является представлением физической модели. Внутри конструктора происходит инициализация параметров модели из конфигурационного файла. Метод getScene вызывается при запросе из главного контроллера: если сцена модели не инициализирована, вызывается метод constructScene, который загружает сцену на экран и инициализирует контроллер модели, после чего в контроллер загружаются атрибуты сцены, и он используется по дальнейшему назначению. Если во время инициализации сцены модели происходит критическая ошибка, модель больше не загружается, а для разработчика выводится в консоли тип ошибки. Также класс хранит в себе объект класса AbstractModelController, являющийся представлением контроллера модели: внутри модели он необходим для загрузки настроек модели из контроллера в главный контроллер, а также для исполнения своего функционала по нажатию кнопки начала выполнения.

Класс InfoDialog является наследником класса диалогового окна платформы JavaFX и расширяет его функционал. Внутри себя содержит описание, объект сцены для связи с ее атрибутами и объект класса WebView, управляющий web-движком и отображающий его содержимое. Web-движок обладает способностью парсить html-код и отображать его со стилями. Для изучения способов взаимодействия с окном был использован материал из стороннего источника [27].

Класс SettingsToolbar является представлением выдвигающейся панели настроек, состоящей из кнопки, раскрывающей виджет, и из панели, вмеща-

ющей внутри себя перечисление настроек текущей модели с ключом названия настройки и значением объекта класса Control, благодаря которому можно вносить изменения в отображение модели в реальном времени. Сразу после загрузки программы панель настроек невидима, так как на тот момент ни одна модель не загружена, после выбора модуля из выпадающего списка панель становится видимой. При нажатии на кнопку панель выдвигается справа налево, кнопка блокируется до полного раскрытия панели во избежание коллизий и повышения отказоустойчивости приложения. При изменении модели панель настроек сворачивается автоматически: для ручного сворачивания необходимо снова нажать кнопку.

Класс MarkingGrid является представлением разметочной сетки. Сетка является удобным инструментом для анализа расчетов расстояния или расположения графических компонентов. Каждая клетка имеет размер 100*100 пикселей и представляет из себя количество пикселей на одну условную единицу: это может быть метр, сантиметр, дециметр и так далее, разработчик сам определяет меру условной единицы. Она отображается при нажатии на кнопку отображения сетки; если в конфигурационном файле не задана соответствующая настройка, кнопка становится неактивной, из-за чего активировать сетку невозможно. Если сетка отображена и пользователь переключает модель, она автоматически исчезает, несмотря на то, доступна ли сетка в выбранной модели или нет.

Класс AppHeader является представлением заголовка окна. Так как средствами языка Java невозможно изменить отображение заголовка, его необходимо создать самостоятельно. В конструктор переданы параметры панели заголовка, кнопки свертывания, развертывания и закрытия приложения: в нем были приданы стили компонентам и добавлены обработчики действий. Перетаскивание окна было реализовано следующим образом: сначала обработано события нажатия на заголовок, затем – событие перемещения заго-

ловка при зажатой мыши, после этого окно приложения перемещается по рассчитанным координатам относительно экрана монитора.

Абстрактный класс контроллера AbstractModelController необходим для реализации базовых возможностей контроллеров моделей-наследников. Класс предоставляет доступ к настройкам модели, установке атрибутов сцены и связывание текстового поля настройки с изменяемыми атрибутами модели. Контроллеры моделей, наследуемые от абстрактного класса, должны реализовывать абстрактные методы, такие как construct, в котором идет инициализация модели, createSettings, в котором инициализируется структура настроек, а также метод execute – функция активации модели; если модель не предполагает запуска какого-либо процесса, его можно не переопределять. Также абстрактный класс предоставляет возможность устанавливать всплывающие подсказки для панели настроек, если их название слишком большое и не влезает в панель, и возможность связывать текстовые поля настроек с изменяемыми атрибутами модели. Таким образом, весь перечисленный функционал доступен каждому наследнику этого класса.

2.4 Разработка функционала моделей

В выпускной квалификационной работе представлено четыре физические модели, а именно:

- модель «Пушечное ядро», позволяющая анализировать параболическое движение снаряда, а также найти время, дистанцию и максимальную высоту полёта снаряда;
- модель «Груз с пружиной», позволяющая исследовать закономерность между массой груза и растяжением пружины; также в ней можно прикладывать силу с помощью перемещения зажатой мыши;
- модель «Преломление через линзу», позволяющая исследовать закономерность между расстоянием до линзы и размером отображающегося изображения.

Каждая модель принадлежит конкретному модулю: так первая модель принадлежит модулю «Кинематика», вторая модель – модулю «Динамика», третья – модулю «Оптика».

2.4.1 Модель «Пушечное ядро»

Модель основана на теме «Движение тела, брошенного под углом к горизонту» [24]. В состав модели включены колесо, ствол орудия и снаряд: для вращения ствола использовалось аффинное преобразование [25].

Допустим, нам известна начальная скорость снаряда `V`, угол наклона ствола `α` и изначальная величина `h`. Тогда можно найти следующие величины.

1. Компоненты скорости: скорость снаряда состоит из горизонтальной и вертикальной составляющих:

$$V_x = V\cos\alpha;$$

$$V_y = V\sin\alpha.$$

2. Уравнение движения: пройденное расстояние по горизонтали можно выразить как $x = V_x t$, где t – время; вертикальное расстояние от земли описывается формулой $y = h + V_y t - \frac{gt^2}{2}$, где g – гравитационная постоянная, равная 9.8.
3. Время полета: полет заканчивается, когда снаряд падает на землю, это происходит, когда вертикальное расстояние от земли равно 0. В случае, когда начальная высота равна 0, уравнение можно описать так:

$$V_y t - \frac{gt^2}{2} = 0.$$

Тогда из этого уравнения находим, что что время полета равно:

$$t = \frac{2V_y}{g} = \frac{2Vsin\,\alpha}{g}.$$

Однако если предмет бросается с некоторой высоты, то получается следующее квадратное уравнение:

$$h + V_y t - \frac{gt^2}{2} = 0;$$

$$t = \frac{V_y + \sqrt{V_y^2 + 2gh}}{g}.$$

4. Дальность полета снаряда – это горизонтальное расстояние, пройденное за время полета. Формула имеет вид $D = V_x t$. Формула времени берется из предыдущего пункта.
5. Максимальная высота: когда снаряд достигает максимальной высоты, он перестает двигаться вверх и начинает падать. Это означает, что его вертикальная составляющая скорости меняется с положительной на отрицательную, другими словами, она равна 0 на короткий момент времени $t(V_y = 0)$. Если $V_y - gt(V_y = 0)$, можно переписать уравнение следующим образом:

$$h_{max} = V_y t(V_x = 0) - \frac{gt^2(V_y = 0)}{2} = \frac{V_y^2}{2g}.$$

К счастью, в случае запуска снаряда с некоторой начальной высоты h необходимо добавить это значение в итоговую формулу:

$$h_{max} = h + \frac{V_y^2}{2g}.$$

Для использования данной модели следует изменить угол ствола для прорисовки траектории и запустить демонстрацию модели, дважды нажав на ствол либо нажав на кнопку "Начать выполнение" в панели инструментов. За условную единицу принимается один метр.

Для реализации данного функционала был использован вспомогающий объект класса TrajectoryLine, который отвечает за прорисовку кривой траек-

тории и расчета ее координат. Начальная высота ствола – 0.675 метров, скорость по умолчанию – 6 метров в секунду, начальное положение ствола орудия – 0 градусов. Для изучения способов перемещения объектов по траектории был использован материал из стороннего источника [28].

В модели заданы следующие настройки: угол наклона ствола в градусах, начальная скорость снаряда в метрах в секунду, время полета в секундах, дистанция в метрах, изначальная высота в метрах, максимальная высота в метрах.

2.4.2 Модель «Груз с пружиной»

Эта модель демонстрирует зависимость растяжения пружины от веса подвешенного груза и приложенной силы. С ее помощью можно найти длину растянутой пружины и приложенную к ней силу. В состав модели включены пружина и груз.

Для реализации модели достаточно знаний из школьного курса физики, а именно знать единственную формулу $F = mg = kx$, где F – сила упругости в Ньютонах, m – масса груза в килограммах, g – гравитационная постоянная, k – жесткость пружины в Ньютонах на метр, x – удлинение пружины в метрах. Также в настройки модели добавлен параметр «Приложенная сила» в Ньютонах, позволяющий определять силу, с которой пользователь растягивает пружину.

Растяжение от груза задается через ввод настроек, ручное растяжение – при перетаскивании предмета. Итоговое растяжение получается из суммы двух растяжений. Ручное удлинение считается как разница между положением груза при перемещении и при расположении без перетягивания, деленная на количество пикселей на один метр.

В начальном положении груз с массой 1.52 килограмма висит на пружине с растяжением 0.05 метра и жесткостью 300 Ньютонов на метр. Для использования данной модели следует изменить параметры модели в панели

настроек и перемещать груз по вертикали с помощью зажатой мыши. За условную единицу принимается 0.05 метра.

В модели заданы следующие настройки: жесткость пружины в Ньютонах на метр, масса груза в килограммах, приложенная сила в Ньютонах, удлинение пружины в метрах: последние две настройки являются не редактируемыми.

Перемещение груза по координатам строится следующим образом:

• при ручном удлинении: когда пользователь держит груз зажатой мышью, тем самым растягивая пружину, высчитывается ручное растяжение принципом, описанным выше; после этого вызывается метод calculate, который рассчитывает остальные параметры;

• при введении значений в панель настроек: после изменения параметров ручное растяжение устанавливается в 0, после чего вызывается метод calculate.

Метод calculate рассчитывает растяжение от веса груза, положение груза по оси Y при задействованном растяжении от груза, приложенную пользователем силу, рассчитываемую как F = kx; итоговое растяжение пружины и устанавливает соответствующую координату по оси Y для груза и длину пружины.

Пружина может растягиваться как при перемещении вниз, так и при перемещении вверх: во втором случае приложенная сила становится отрицательной, так как меняется направление силы – абсолютная величина остается такой же.

Для повышения интерактивности модели была продумана анимация перемещения груза вдоль оси Y: после расжатия мыши груз начинает двигаться в противоположном направлении до рассчитанной величины, затем двигается в противоположную сторону, и так происходит до тех пор, пока расстояние перемещения не становится слишком маленьким. Для реализации изложенного алгоритма процесс перемещения груза и изменения длины пру-

жины был перемещен в отдельный поток. Каждое предстоящее расстояние рассчитывается как модуль разницы изначального положения груза и его текущего положения, умноженное на 1.5. Скорость перемещения груза за одно расстояние – 150 миллисекунд, однако был задан закон перемещения, по которому груз идет вверх в два раза быстрее, чем вниз – в будущем можно придумать более сложный закон перемещения. На каждой итерации груз перемещается на единицу вверх, либо вниз, на такую же величину меняется длина пружины.

2.4.3 Модель «Преломление через линзу»

Эта модель демонстрирует искажение предмета через линзу. С ее помощью можно найти проекцию предмета через тонкую линзу. В состав модели включены: линза, декартова система координат с 0 посередине, изображение.

Модель основана на теме «Построение изображения предмета в тонкой линзе» [23], для ее реализации используется формула D = 1/F, где D – оптическая сила линзы в диоптриях, F – фокусное расстояние в метрах. Для упрощения реализации было принято решение строить изображение предмета только в тонкой собирающей линзе, остальные виды линз будет реализовывать команда разработчиков.

В начальном расположении предмет находится на расстоянии 0.2 метра от линзы на расстоянии двух фокусов: при такой расстановке размер предмета равен размеру изображения. Начальная оптическая сила линзы равна 10 диоптрий, таким образом фокусное расстояние равно 1/10 = 0.1 метра. Предмет можно перемещать вдоль оси X с помощью перемещения зажатой мыши. При каждом изменении координаты предмета будет пересчитываться размер изображения. За условную единицу принимается 1 фокусное расстояние.

В модели заданы следующие настройки: расстояние предмета от линзы в метрах, оптическая сила линзы в диоптриях, доля от изначального размера в процентах: последняя настройка является не редактируемой.

Изображение в тонкой собирающей линзе строится следующим образом: от предмета идет перпендикулярный линзе луч, затем от точки пересечения этого луча и линзы строится другой луч A, проходящий через фокусное расстояние линзы; после этого строится луч B, излучающийся от предмета и проходящий через оптический центр линзы. Остается найти точку пересечения лучей A и B – полученная точка является точкой изображения.

При перемещении предмета перестраиваются параметры прямых и изображения, а именно: начало прямой от объекта к линзе, начало и конец прямой от объекта до изображения, начало и конец прямой от линзы до изображения, дистанция от предмета до линзы, доля от изначального размера, размер изображения.

Если предмет расположен до фокуса, то изображение будет действительным и перевернутым, иначе оно будет мнимым и прямым. Для реализации такого приема был реализован следующий подход: если предмет находится за фокусным расстоянием, изображение переворачивается на 180 градусов и перемещается на высоту оси X, иначе остается прямым и перемещается на высоту точки пересечения прямых.

Для этой модели существуют ограничения расположения предмета: его нельзя перемещать за линию линзы, также нельзя перемещать на фокусное расстояния, так как на такой дистанции линии до изображения не пересекаются. Для решения этой проблемы в методе arrowDragged был применен следующий подход: если объект близок к фокусному расстоянию на отрезок, равный 5 пикселам, то если объект перемещается слева направо, он автоматически сдвигается вправо на два отрезка, иначе сдвигается влево на то же расстояние. Пользователь может попробовать изменить расстояние через панель настроек, однако ему не удастся это сделать: если ввести расстояние не больше 0, высветится предупреждение «Дистанция до линзы должна быть больше 0», если ввести расстояние, близкое к фокусному расстоянию на отрезок, высветится предупреждение «Нельзя устанавливать объект на фокус-

ном расстоянии» – в обоих случаях после неудачного изменения параметра, текстовое поле запомнит предыдущее значение и выведет его.

2.5 Пример расширения функционала и кодовой базы приложения

В этом подразделе приводится пример добавления нового модуля и модели. Подразумевается, что этим будет заниматься сторонний разработчик, имеющий опыт разработки на языке Java.

Для добавления модуля необходимо добавить переменную с приставкой в перечисление modules и задать ее описание на примере остальных модулей.

Для добавления модели необходимо сделать следующее:

- добавить переменную с приставкой в перечисление models и задать ее имя, описание, путь и дополнительные характеристики на примере остальных моделей;
- создать файл с расширением *.fxml в папке resources/root/models и наполнить его содержимым – имя файла должно быть таким же, как и в конфигурационном файле;
- создать класс-обработчик модели в папке root/controllers, применив наследование от AbstractModelController и переопределив его методы;
- в *.fxml файле привязать созданный класс-обработчик с отображением модели во вкладке «Controller» слева внизу, если разработчик использует SceneBuilder.

Для добавления параметров модели в панель настроек необходимо создать настройку в методе createSettings и поместить ее в структуру mModelSettings: для демонстрационного варианта ушла одна строка кода.

Расширяемая модель готова: теперь можно реализовывать ее функционал. Пример добавленной модели представлен на рисунке 2.2.

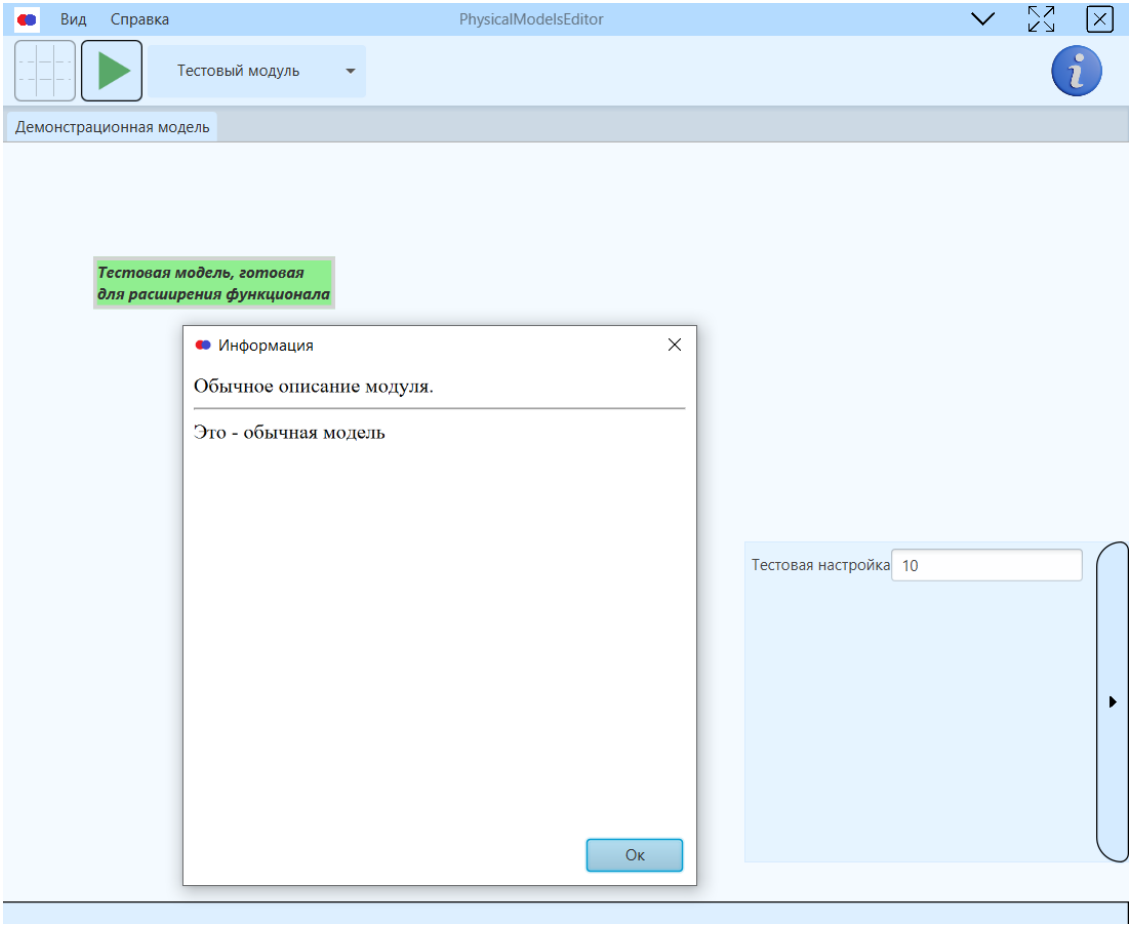

Рисунок 2.2 – Добавленная модель

Таким образом, был реализован основной функционал приложения, разработан его интерфейс, создан прототип приложения с парсером конфигурационного файла и обработчиком моделей, созданы демонстрационные физические модели, а также приведен пример расширения функционала приложения. Руководство по использованию приложения будет приведено в третьем разделе.

3 Описание использования приложения

В этом разделе представлена инструкция по использованию приложения и демонстрация возможностей программы.

3.1 Взаимодействие с конфигурационным файлом

Для редактирования информации о приложении и моделях используется конфигурационный файл: для взаимодействия с ним не требуется знания языков программирования.

Допустим, разработчик допустил ошибку в теории или не дописал важную для понимания учащихся информацию – в таком случае заказчик может внести правки самостоятельно.

Пример внесения дополнительной информации представлен на рисунке 3.1. В конфигурационный файл была добавлена строка « Здесь мог бы быть ваш текст »

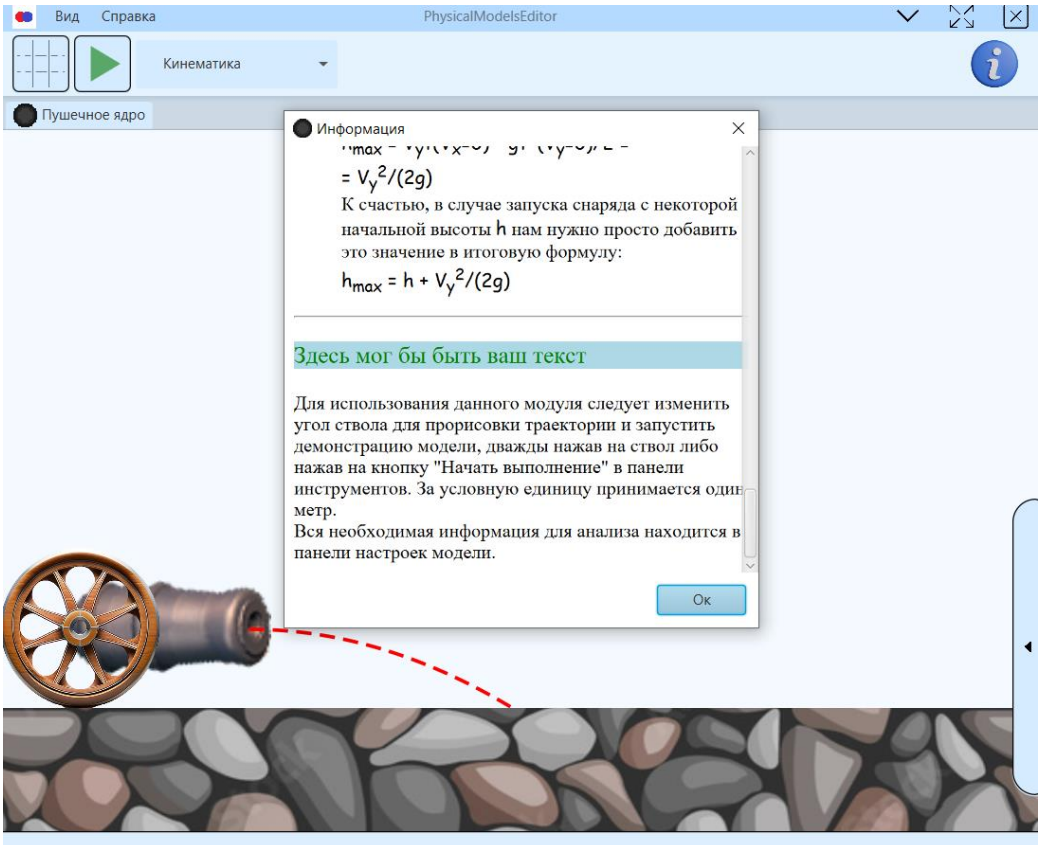

Рисунок 3.1 – Пример добавления информации в конфигурационный файл

3.2 Взаимодействие с приложением

Сразу после открытия приложения пользователю предлагается выбрать модель из выпадающего списка. В выборе пункта меню «Справка» → «О программе» отображается диалоговое окно с информацией о приложении. Также в этом меню есть опция «Вид» → «Включить сетку»: она активна, только если текущая модель поддерживает отображение разметочной сетки.

Первой моделью является «Пушечное ядро» из раздела «Кинематика». При изменении угла ствола пушки меняется кривая траектория движения снаряда, вследствие чего меняются остальные параметры модели. Менять угол ствола можно вручную либо через панель настроек. Угол должен быть в диапазоне от 0 до 90 градусов. Скорость снаряда можно задавать только через панель настроек.

После нажатия кнопки «Начать выполнение» пунктирная линия траектории исчезает, и по этой траектории летит снаряд; он летит с постоянной скоростью, так как реализация ускорения показалась сложной и невыполнимой в срок.

На рисунке 3.2 представлена демонстрация модели «Пушечное ядро».

Второй моделью является «Груз с пружиной» из раздела «Динамика». При зажатии мыши на грузе и попытке перемещения вверх или вниз в панели настроек меняются параметры модели. Его можно перемещать вниз до конца экрана монитора, вверх – только до изначального размера пружины без подвешенного груза (две клетки).

При изменении массы груза или жесткости пружины в панели настроек меняется растяжение пружины в отображении и в настройках.

После расжатия мыши груз возвращается в исходное положение, производя анимацию перемещения.

На рисунке 3.3 представлена демонстрация модели «Груз с пружиной».

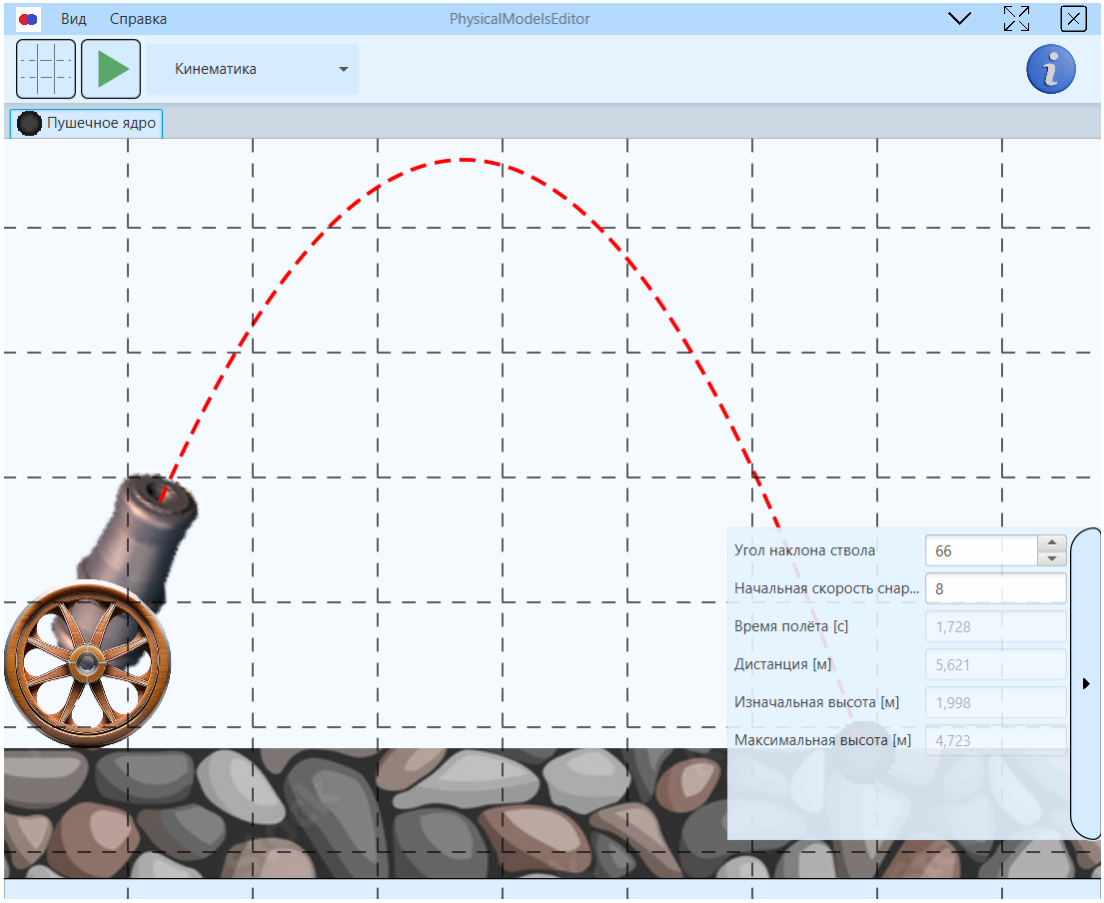

Рисунок 3.2 – Демонстрация модели «Пушечное ядро»

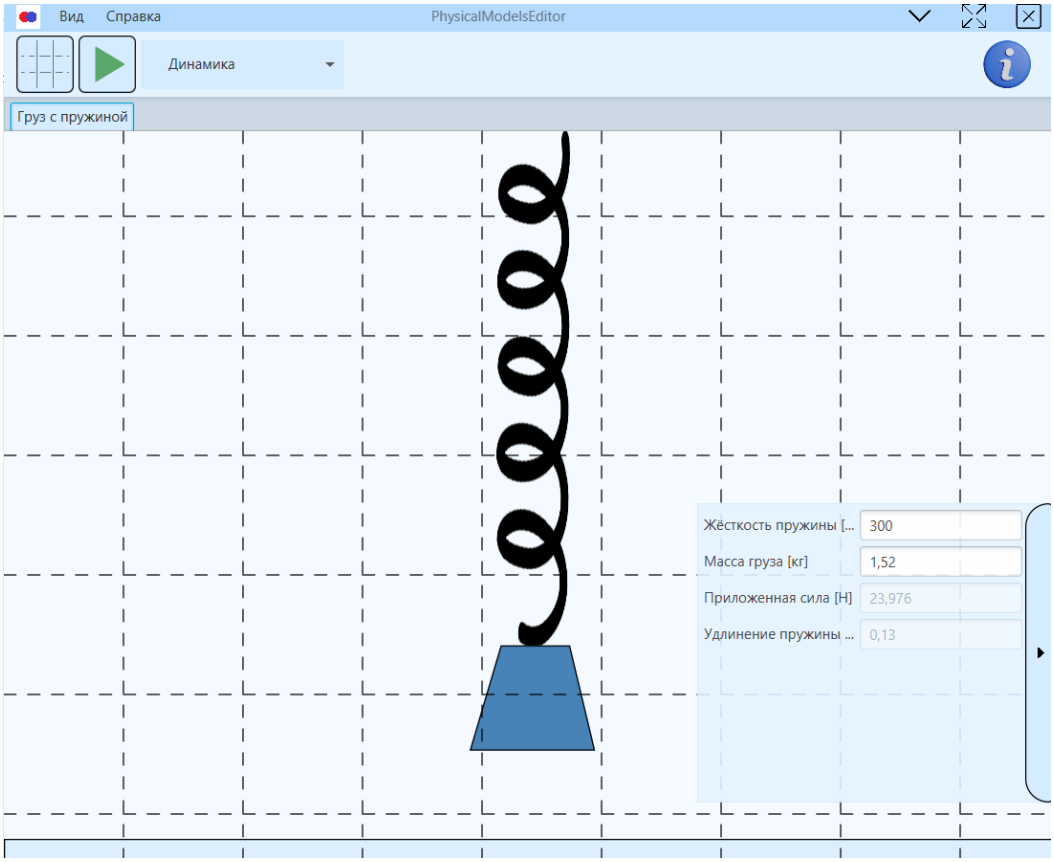

Рисунок 3.3 – Демонстрация модели «Груз с пружиной»

Третьей моделью является «Преломление через линзу». При перемещении предмета меняется отображение изображения через линзу, а также параметры в панели настроек. Его можно перемещать только по горизонтали, предмет не может перейти на другую сторону линзы, также его невозможно установить на фокусное расстояние: в таком случае предмет смещается относительно своего перемещения.

При изменении расстояния от линзы в панели настроек предмет смещается на указанную дистанцию, а при изменении оптической силы линзы отображение модели не меняется, однако фокусное расстояние становится другим, поэтому предмет находится на расстоянии такого же количества фокусов.

Если предмет находится за линией фокуса, изображение строится по другую сторону линзы и остается перевернутым, иначе изображение переходит на другую сторону линзы и переворачивается.

На рисунке 3.4 и 3.5 представлена демонстрация модели «Преломление через линзу».

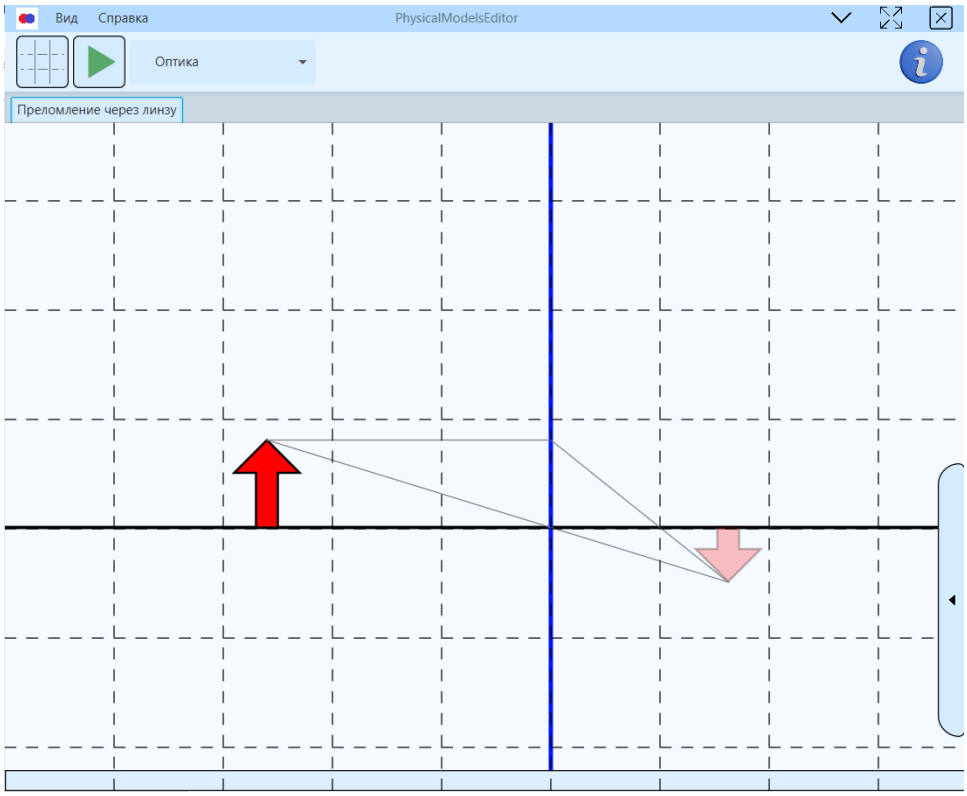

Рисунок 3.4 – Предмет находится за линией фокуса

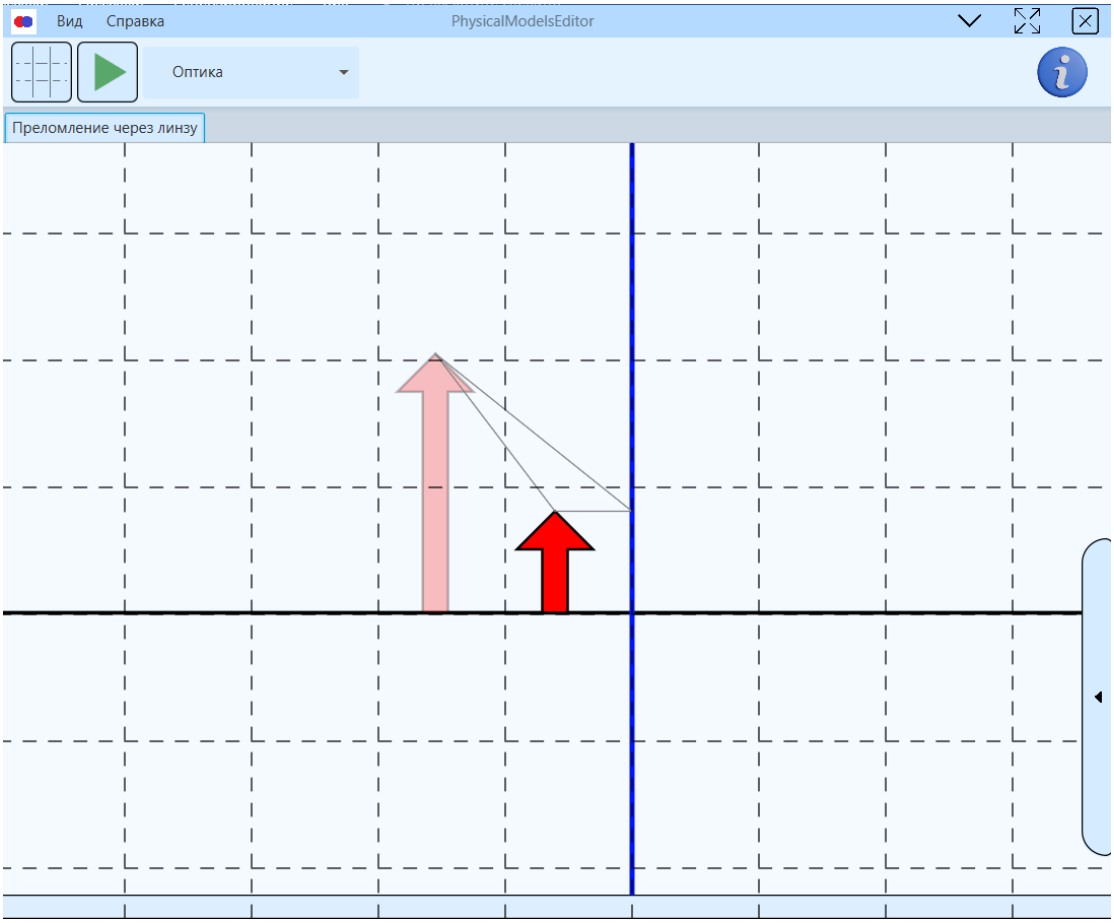

Рисунок 3.5 – Предмет расположен между фокусом и линзой

В данном подразделе было предоставлено описание использования приложения и демонстрация его возможностей, а именно показано редактирование конфигурационного файла и продемонстрирована работа моделей.

4 Экономическое обоснование дипломного проекта

В данной выпускной квалификационной работе разрабатывается приложение для симуляции физических моделей из школьного курса физики. Готовое приложение должно быть простым в использовании и в расширении функционала, если его будут дорабатывать сторонние разработчики.

Цель работы – разработка редактора физических моделей для исследования физических закономерностей и процессов в интерактивной форме с использованием JavaFX [5], внедряющий широкий пользовательский интерфейс.

Область применения приложения – образовательные учреждения, где преподаватели могут использовать его для демонстрации физических моделей и проведения интерактивных уроков. Также приложение может использоваться школьниками для самостоятельного изучения разделов физики.

4.1 Расчет затрат на оплату труда

Для расчета затрат на этапе проектирования необходимо правильно сформировать план-график проводимых работ. На его основе определяется трудоёмкость работ, далее рассчитывается рабочая ставка за день.

На сайте zp.hh.ru [6] для получения значения минимальной зарплаты разработчика в графу «Регион» следует вписать «Санкт-Петербург», в графу «Профессиональная область» – «Разработка программного обеспечения», в графу «Ваша роль в компании» – «Специалист», в «Стаж» – «Меньше года или отсутствует» – тогда сервис мониторинга зарплат выдает минимальное значение в 70000 рублей, для руководителя – 110000 рублей.

На основе полученной информации рассчитываются рабочие ставки программиста и руководителя:

$$C_{\text{прог}} = \frac{70000 \text{ рублей}}{21 \text{ день}} = 3333 \frac{\text{рублей}}{\text{день}}$$

$$C_{\text{рук}} = \frac{110000 \text{ рублей}}{21 \text{ день}} = 5238 \frac{\text{рублей}}{\text{день}}$$

Данные о трудоёмкости выполнения работ указаны в человеко-днях и приведены в таблице 4.1.

Таблица 4.1 – Трудоёмкость этапов разработки

№	Этапы и содержание выполняемых работ	Исполнитель	Трудоёмкость, чел/день	Ставка, руб/день
1	Разработка ТЗ и его согласование с заказчиком	Руководитель	4	5238
2	Изучение технологий для работы	Разработчик	4	3333
3	Создание прототипа	Разработчик	3	3333
4	Разработка пользовательского интерфейса	Разработчик	2	3333
5	Разработка функционала моделей	Разработчик	14	3333
6	Тестирование и стабилизация приложения	Разработчик	4	3333
7	Согласование с заказчиком и исправление замечаний	Разработчик	5	3333
		Руководитель	5	5238

Итоговая трудоёмкость руководителя:

$$T_{\text{рук}} = 9 \text{ чел/дней}$$

Итоговая трудоёмкость программиста:

$$T_{\text{прог}} = 32 \text{ чел/дня}$$

На основе информации от трудоёмкости руководителя и программиста, участвующих в рабочей деятельности, рассчитываются расходы на заработную плату: при выполнении расчетов в выпускной квалификационной работе норматив дополнительной заработной платы составляет 14%, а на социальные отчисления уходит 30,2% [7].

Основная заработная плата считается следующим образом:

$$З_{\text{осн.з/пл}} = T_{\text{рук}} * C_{\text{рук}} + T_{\text{прог}} * C_{\text{прог}} = 5238 * 9 + 3333*32 = 153798$$

рублей,

где T_i – время, затраченное i-м исполнителем на проведение работ (в человеко-днях);

C_i – ставка i-го исполнителя (в рублях в день).

Расходы на дополнительную заработную плату определяются по формуле:

$$З_{доп.з/пл} = З_{осн.з/пл} * \frac{Н_{доп}}{100},$$

где $З_{доп.з/пл}$ – расходы на дополнительную заработную плату исполнителей;

$З_{осн.з/пл}$ – расходы на основную заработную плату исполнителей;

$Н_{доп}$ – норматив дополнительной заработной платы.

Дополнительная заработная плата составляет:

$$З_{доп.з/пл} = З_{осн.з/пл} * \frac{Н_{доп}}{100} = 153798 * 0.14 = 21531{,}72 \text{ рублей}$$

Отчисления на страховые взносы на обязательное социальное, пенсионное и медицинское страхование определяются по сумме основных и дополнительных зарплат штата и равны:

$$З_{соц} = (З_{осн.з/пл} + З_{доп.з/пл}) * \frac{Н_{соц}}{100} = (153798 + 21531{,}72) * 0.302$$
$$= 52949{,}58 \text{ рублей}$$

4.2 Расчет накладных расходов

Расчет накладных расходов вычисляется по формуле:

$$С_{нр} = (З_{осн.з/пл} + З_{доп.з/пл}) * \frac{Н_{нак}}{100},$$

где $С_{нр}$ – накладные расходы, руб.;

$З_{осн.з/пл}$ – расходы на основную заработную плату исполнителей, руб.;

$З_{доп.з/пл}$ – расходы на дополнительную заработную плату исполнителей, руб.;

$Н_{нак}$ – норматив отчислений накладных расходов. В данной работе норматив накладных расходов равен 20% от суммы основной и дополнительной заработной платы. Рассчитаем данный показатель:

$$С_{нр} = (153798 + 21531{,}72) * 0.2 = 35065{,}94 \text{ рублей}$$

4.3 Расходы на материалы

Данные расчёта затрат на расходные материалы с учётом транспортно-заготовительных расходов занесены в таблицу 4.2. Согласно методическим материалам, норма транспортно-заготовительных материалов – 10%.

Т.к. купленное устройство для печати уже было заправлено картриджами, их стоимость в траты не включена.

Таблица 4.2 – Затраты на материалы

Материалы	Кол-во	Цена, руб.	Сумма, руб.
Бумага для офисной техники SVETOCOPY 500л [8]	1	349	349
Сетевой фильтр ЭРА [9]	1	449	449
Кабель для компьютера Belkin USB-A(M)/USB-B(M) [10]	2	699	1398
Патч-корд DEXP HtsPcUSt5E150 [11]	2	599	1198
Итого:			3394
Транспортно-заготовительные расходы (10%):			339,4
Всего:			3733,4

4.4 Издержки на амортизацию ПК и оргтехники

В ходе разработки приложения использовался ноутбук и МФУ. Стоимость ноутбука Ноутбук MSI Modern 14 – 34999 рублей [12], стоимость МФУ HP LaserJet M141w – 20499 рублей [13].

Сумма годовой амортизации вычисляется по формуле:

$$A_г = К_{об} * \frac{Н_{АМ}}{100},$$

где $К_{об}$ – стоимость оборудования, $Н_{АМ}$ – норма амортизации.

Согласно Постановлению Правительства РФ от 01.01.2002 №1 (ред. от 28.04.2018) «О Классификации основных средств, включаемых в амортизационные группы», компьютеры и печатающие устройства относятся ко второй группе, их срок полезного использования составляет от двух до трех лет.

Срок службы МФУ и ноутбука – 3 года. Норма амортизации – 33,3 %.

$$A_г = (34999 + 20499) * \frac{33{,}3}{100} = 18499{,}3 \text{ рублей}$$

Сумма амортизации за один рабочий день равна:

$$A_{\text{д}} = \frac{A_{\text{г}}}{N}.$$

В 2023 году 247 рабочих дней, поэтому:

$$A_{\text{д}} = \frac{18499{,}3}{247} = 74{,}9 \text{ рублей.}$$

Амортизационные отчисления за время работы над ВКР:

$$A_{\text{ВКР}} = A_{\text{д}} * 32 = 2396{,}8 \text{ рублей.}$$

4.5 Расходы на услуги сторонних организаций

Для разработки дипломного проекта требовался доступ в Интернет, поэтому была оформлена услуга подключения к Интернету у компании ООО "Ростелеком" за ежемесячную плату в размере 500 рублей.

Также была оказана услуга стороннего дизайнера, который рисовал формы, окружение и различные GUI-компоненты: за работу он взял 10000 рублей.

Таблица 4.3 – Затраты по работам, оказываемым сторонними организациями

Наименование	Кол-во	Цена, руб.	Сумма, руб.
Доступ в Интернет, мес.	2	650	1300
Работа дизайнера	1	10000	10000
Итого			11300

4.6 Себестоимость выполнения дипломной работы

Расчет себестоимости выполнения работы представлен в таблице 4.4.

Таблица 4.4 – Расчет себестоимости выполнения работ

Наименование статьи	Сумма, руб.	Структура себестоимости, %
Расходы на оплату труда	175329,72	62,4
Отчисления на социальные нужды	52949,58	18,9
Накладные расходы	35065,94	12,5
Материальные затраты	3733,4	1,3
Амортизационные отчисления	2396,8	0,85
Расходы на услуги сторонних организаций	11300	4
Итого:	280775,44	100

4.7 Вывод

В данном разделе была рассчитана итоговая себестоимость приложения для симуляции физических моделей с точки зрения вложенных в него финансовых средств.

При разработке проекта были учтены затраты на оплату труда, материалы, социальные отчисления, амортизационные отчисления и сторонние траты. Большая часть затрат пришлась на оплату труда, так как рабочий коллектив – это основной ресурс в любом проекте.

Исходя из расчётов, себестоимость конечного продукта составила 280775 рублей и 44 копейки. Основную часть расходов составляют расходы на оплату труда (62,4 %), отчисления на социальные нужды (18,9 %) и накладные расходы (12,5 %).

ЗАКЛЮЧЕНИЕ

В ходе выполнения выпускной квалификационной работы было разработано приложение для симуляции физических моделей, направленное для использования в образовательных учреждениях. Также был проведён сравнительный анализ существующих решений и технологий и создан демонстративный пример работы приложения. Программа предоставляет возможность пользователю изучить физические закономерности, используя среду и объекты приложения.

Для реализации программного обеспечения были использованы платформа для создания приложений с широким пользовательским функционалом JavaFX и язык Java. Для поддержки такой системы достаточно небольшое количество человек и вычислительных ресурсов. Приложение получилось с открытым исходным и задокументированным кодом, что даёт возможность добавлять различный функционал всем пользователям.

Развитием данного приложения является добавление новых физических моделей, доработка старых, уточнение теоретической базы, перенос приложения на различные платформы и ОС, улучшение графической составляющей. Наличие такого функционала может позволить приложению выйти на новую аудиторию и предоставить рынку нового конкурента.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Канал «MinutePhysics» [Электронный ресурс] URL: https://www.youtube.com/@MinutePhysics. (Дата обращения: 04.05.2023).
2. Симулятор [Электронный ресурс] URL: http://www.algodoo.com/. (Дата обращения: 04.05.2023).
3. Симулятор [Электронный ресурс] URL: https://visualmathstart.ru/. (Дата обращения: 04.05.2023).
4. Официальный сайт платформы Qt [Электронный ресурс] URL: https://www.qt.io/. (Дата обращения: 04.05.2023).
5. Официальный сайт платформы JavaFX [Электронный ресурс] URL: https://openjfx.io/. (Дата обращения: 04.05.2023).
6. Сервис для получения минимальной зарплаты [Электронный ресурс] URL: https://zp.hh.ru/. (Дата обращения: 04.05.2023).
7. Отчисления на соцнужды [Электронный ресурс] URL: https://ppt.ru/art/buh-uchet/otchisleniya-na-socnuzdi. (Дата обращения: 04.05.2023).
8. Бумага для офисной техники [Электронный ресурс] URL: https://www.bookvoed.ru/book?id=13600078. (Дата обращения: 04.05.2023).
9. Сетевой фильтр [Электронный ресурс] URL: https://www.mvideo.ru/products/setevoi-filtr-era-usf-4es-15m-w-50131692. (Дата обращения: 04.05.2023).
10. Кабель для подключения к принтеру [Электронный ресурс] URL: https://www.mvideo.ru/products/kabel-dlya-komputera-belkin-usb-am-usb-bm-18m-f3u154bt18m-50052063. (Дата обращения: 04.05.2023).
11. Кабель Ethernet [Электронный ресурс] URL: https://www.dns-shop.ru/product/b82e7604bd993361/patc-kord-dexp-htspcust5e150/. (Дата обращения: 04.05.2023).

12. Ноутбук для работы [Электронный ресурс] URL: https://www.mvideo.ru/products/noutbuk-msi-modern-14-c11m-9s7-14j312-046-30066723. (Дата обращения: 04.05.2023).

13. Лазерное МФУ [Электронный ресурс] URL: https://www.mvideo.ru/products/lazernoe-mfu-hp-laserjet-m141w-7md74a-30064634. (Дата обращения: 04.05.2023).

14. Видеохостинг [Электронный ресурс] URL: https://www.youtube.com/. (Дата обращения: 12.05.2023).

15. Операционная система Windows [Электронный ресурс] URL: https://ru.wikipedia.org/wiki/Windows. (Дата обращения: 12.05.2023).

16. Операционная система macOS [Электронный ресурс] URL: https://ru.wikipedia.org/wiki/MacOS. (Дата обращения: 12.05.2023).

17. Язык программирования C++ [Электронный ресурс] URL: https://ru.wikipedia.org/wiki/C++. (Дата обращения: 12.05.2023).

18. Платформа для разработки веб-приложений React [Электронный ресурс] URL: https://ru.wikipedia.org/wiki/React. (Дата обращения: 12.05.2023).

19. Платформа для разработки веб-приложений Angular [Электронный ресурс] URL: https://angular.io/. (Дата обращения: 12.05.2023).

20. Платформа для разработки веб-приложений Vue [Электронный ресурс] URL: https://vuejs.org/. (Дата обращения: 12.05.2023).

21. Язык программирования JavaScript [Электронный ресурс] URL: https://ru.wikipedia.org/wiki/JavaScript. (Дата обращения: 12.05.2023).

22. Шаблон проектирования Синглтон [Электронный ресурс] URL: https://ru.wikipedia.org/wiki/Одиночка_(шаблон_проектирования). (Дата обращения: 20.05.2023).

23. Построение изображения предмета в тонкой линзе [Электронный ресурс] URL: https://www.yaklass.ru/p/fizika/9-klass/svetovye-iavleniia-131515/linza-fokusnoe-rasstoianie-linzy-postroenie-izobrazhenii-163989/re-9a238ed5-04a4-4a28-82a7-8ba6c45343e1. (Дата обращения: 21.05.2023).

24. Движение тела, брошенного под углом к горизонту [Электронный ресурс] URL: https://online.mephi.ru/courses/physics/osnovi_-mehaniki/data/lecture/2/p7.html. (Дата обращения: 21.05.2023).

25. Учебное пособие по преобразованиям JavaFX [Электронный ресурс] URL: https://o7planning.org/11157/javafx-transformation. (Дата обращения: 27.05.2023).

26. Изменение цвета заголовка в фреймворке JavaFX [Электронный ресурс] URL: https://stackoverflow.com/questions/42583779/how-to-change-the-color-of-title-bar-in-framework-javafx. (Дата обращения: 27.05.2023).

27. Использование диалоговых окон в JavaFX [Электронный ресурс] URL: https://code.makery.ch/blog/javafx-dialogs-official/. (Дата обращения: 27.05.2023).

28. Пример переходов в JavaFX [Электронный ресурс] URL: https://examples.javacodegeeks.com/java-development/desktop-java/javafx/javafx-transition-example/. (Дата обращения: 27.05.2023).

29. Язык программирования Java [Электронный ресурс] URL: https://ru.wikipedia.org/wiki/Java. (Дата обращения: 27.05.2023).

30. Генератор документации в html-формате JavaDoc [Электронный ресурс] URL: https://ru.wikipedia.org/wiki/Javadoc. (Дата обращения: 27.05.2023).

31. Описание класса BorderPane [Электронный ресурс] URL: https://docs.oracle.com/javase/8/javafx/api/javafx/scene/layout/BorderPane.html. (Дата обращения: 27.05.2023).

32. Описание класса StackPane [Электронный ресурс] URL: https://docs.oracle.com/javase/8/javafx/api/javafx/scene/layout/ StackPane.html. (Дата обращения: 27.05.2023).

33. Описание класса TabPane [Электронный ресурс] URL: https://docs.oracle.com/javase/8/javafx/api/javafx/scene/control/TabPane.html. (Дата обращения: 27.05.2023).

34. Описание класса HBox [Электронный ресурс] URL: https://docs.oracle.com/javase/8/javafx/api/javafx/scene/layout/HBox.html. (Дата обращения: 27.05.2023).

35. Семейство шрифтов Comic Sans [Электронный ресурс] URL: https://ru.wikipedia.org/wiki/Comic_Sans. (Дата обращения: 27.05.2023).

36. Описание класса WebView [Электронный ресурс] URL: https://docs.oracle.com/javase/8/javafx/api/javafx/scene/web/WebView.html. (Дата обращения: 27.05.2023).

ПРИЛОЖЕНИЕ А. Полноэкранное отображение приложения

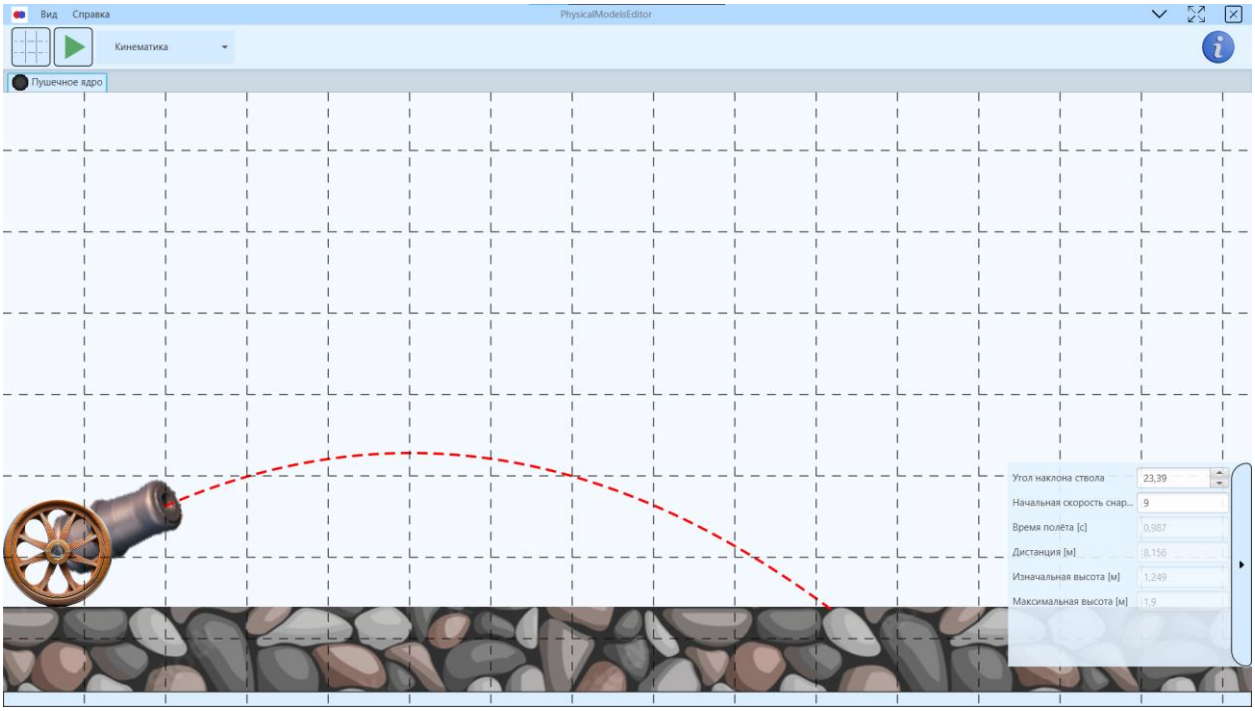

Рисунок А.1 – демонстрация работы приложения в полноэкранном масштабе

ПРИЛОЖЕНИЕ Б. Описание классов

Таблица Б.1 – Описание классов приложения

Класс	Описание класса
AbstractModelController	Абстрактный класс контроллера модели
CannonballController	Контроллер модели, анализирующей параболическое движение снаряда
RefractionThroughLensController	Контроллер модели, позволяющей исследовать закономерность между расстоянием до линзы и размером отображающегося изображения
WeightWithSpringController	Контроллер модели, описывающей зависимость растяжения пружины от силы упругости
AppHeader	Класс, определяющий поведение заголовка
InfoDialog	Класс диалогового окна с информацией о модели/программе
MarkingGrid	Класс, отвечающий за прорисовку разметочной сетки
PolygonArrow	Представление объекта стрелки в виде полигональной линии
SettingsToolbar	Класс, определяющий поведение панели настроек модели
StatusBarController	Класс, определяющий поведение строки состояния
Model	Класс, хранящий и обрабатывающий характеристики физической модели. Характеристики берутся из конфигурационного файла
ModuleFactory	Класс, являющийся модулем и хранящий в себе тематические модели. По сути, является фабрикой моделей
Constants	Хранилище глобальных констант
Global	Хранилище глобальных функций
LineWithOdds	Класс, расширяющий функционал класса Line. Хранит в себе коэффициенты прямой
Logger	Статический класс для вывода информации пользователю
Point	Представление точки
MainController	Главный контроллер, отвечающий за управление приложением
MainApplication	Точка входа в приложение для фреймворка JavaFX
Main	Точка входа в приложение
TrajectoryLine	Класс, отвечающий за прорисовку кривой траектории

ПРИЛОЖЕНИЕ В. Код класса MainController

```
/** Главный контроллер, отвечающий за управление приложением*/
public class MainController implements Initializable {
    @FXML
    private ComboBox<ModuleFactory> moduleTitlesComboBox;
    @FXML
    private Label moduleLabel;
    @FXML
    private ToolBar toolBar;
    @FXML
    private MenuBar menuBar;
    @FXML
    private Button infoButton;
    @FXML
    private Button gridButton;
    @FXML
    private Button executeButton;
    @FXML
    private MenuItem curModelInfoMenuItem;
    @FXML
    private MenuItem gridMenuItem;
    @FXML
    private MenuItem programInfoMenuItem;
    @FXML
    private TabPane modelsTabPane;
    @FXML
    private StackPane stackPane;
    @FXML
    private Label statusBar;
    @FXML
    private HBox statusBarBox;
    @FXML
    private Button settingsToolButton;
    @FXML
    private ScrollPane settingsToolBar;
    @FXML
    private HBox header;
    @FXML
    private Button collapseButton;
    @FXML
    private Button expandButton;
    @FXML
    private Button closeButton;
    @FXML
    private Pane markingGrid;

    /** Список фабрик (модулей)*/
    private final List<ModuleFactory> mFactories = new ArrayList<>();
    /** Информационное диалоговое окно*/
```

```
private InfoDialog mInfoDialog;
/** Панель управления моделью на текущей сцене*/
private SettingsToolbar mSToolbar;
/** Разметочная сетка*/
private MarkingGrid mMarkingGrid;

@Override
public void initialize(URL url, ResourceBundle resourceBundle) {
    initGUI();
    initFactories();
    moduleTitlesComboBox.getItems().addAll(mFactories);
}

/** Инициализация GUI-компонентов */
private void initGUI() {
    final var buttonStyle = Global.getCSSThemeColor()+
            "-fx-border-radius: 5px;" +
            "-fx-background-radius: 5px;" +
            "-fx-border-color: black;";

    infoButton.setShape(new Circle());
    gridButton.setStyle(buttonStyle);
    executeButton.setStyle(buttonStyle);
    toolBar.setStyle(Global.getCSSThemeColor(0.4));
    menuBar.setStyle(Global.getCSSThemeColor(-0.25));
    statusBarBox.setStyle(Global.getCSSThemeColor(0.2)+
            "-fx-border-color:black;-fx-border-width:1;");
    moduleTitlesComboBox.setStyle(Global.getCSSThemeColor());
    modelsTabPane.setStyle(Global.getCSSThemeColor(0.1, "back") +
            Global.getCSSThemeColor(0.75));

    infoButton.setOnAction(e->
 onInfoButtonClicked(DialogType.modelInfo));
    curModelInfoMenuItem.setOnAction(e->
 onInfoButtonClicked(DialogType.modelInfo));
    programInfoMenuItem.setOnAction(e->
 onInfoButtonClicked(DialogType.programInfo));
    gridButton.setOnAction(this::onGridButtonClicked);
    gridMenuItem.setOnAction(this::onGridButtonClicked);
    executeButton.setOnAction(this::onExecuteClicked);

    moduleTitlesComboBox.setOnAction(this::getModule);
    moduleTitlesComboBox.setConverter(new StringConverter<>() {
        @Override
        public String toString(ModuleFactory factory) {
            return factory.getModuleName();
        }

        @Override
        public ModuleFactory fromString(String s) {
```

```
            return null;
        }
    });

    mInfoDialog = new InfoDialog("Информация");
    mSToolbar     =     new     SettingsToolbar(settingsToolButton,
 settingsToolBar);
    mMarkingGrid = new MarkingGrid(markingGrid);
    Logger.setSBController(statusBar);
    new     AppHeader(header,     collapseButton,     expandButton,
 closeButton);

    StatusBarController.connectToStatusBar(infoButton);
    StatusBarController.connectToStatusBar(gridButton);
    StatusBarController.connectToStatusBar(executeButton);
}

/** Обработка нажатия на gridButton*/
private void onGridButtonClicked(ActionEvent actionEvent) {
    mMarkingGrid.setVisible(!mMarkingGrid.isVisible());
}

/** Обработка нажатия на executeButton*/
private void onExecuteClicked(ActionEvent actionEvent) {
    final   ModuleFactory   moduleFactory   =   moduleTitlesComboBox.
 getValue();
    final Model model = moduleFactory.getCurrentModel();
    model.execute();
}

/** Получение модуля из ComboBox*/
private void getModule(ActionEvent actionEvent) {
    infoButton.setDisable(false);
    executeButton.setDisable(false);
    curModelInfoMenuItem.setDisable(false);
    stackPane.getChildren().remove(moduleLabel);
    settingsToolButton.setVisible(true);
    modelsTabPane.getTabs().clear();

    final   ModuleFactory   moduleFactory   =   moduleTitlesComboBox.
 getValue();
    for (int i = 0; i < moduleFactory.size(); i++) {
        final var model = moduleFactory.modelAt(i);
        Tab tab = new Tab(model.getModelName());
        tab.setClosable(false);
        tab.setGraphic(model.getIcon());
        tab.setStyle(Global.getCSSThemeColor());
        if (i == moduleFactory.getCurrentModelIndex()) {
            modelChanged(tab, model);
        }
```

```
        modelsTabPane.getTabs().add(tab);

        tab.selectedProperty().addListener((choose) -> {
            boolean isSelected = (boolean)((ObservableValue<?>)
 choose).getValue();
            if (isSelected) { //если таб выбран, загрузить сцену
 (контент)

 moduleFactory.setCurrentModelIndex(modelsTabPane.getTabs().indexO
 f(tab));
                modelChanged(tab, model);
            }
        });
    }

 modelsTabPane.getSelectionModel().select(moduleFactory.getCurrent
 ModelIndex());
}

/** Событие при смене отображения текущей модели*/
private void modelChanged(Tab tab, Model model) {
    tab.setContent(model.getScene());
    gridButton.setDisable(!model.isGridNeeded());
    gridMenuItem.setDisable(!model.isGridNeeded());
    mSToolbar.setVisible(false);
    mMarkingGrid.setVisible(false);
    Platform.runLater(()->{ //Вызывается уже после установки сцены
        mSToolbar.setSettings(model.getSettings());
    });
}

/** Инициализация модулей */
private void initFactories(){
    //инициализация парсера
    DescriptionFileParser fileParser = DescriptionFileParser.
 getInstance();
    final var propertiesMap = getPropertiesMap();
    final var modulesMap = fileParser.getModulesMap();
    for (final var moduleHashMap: modulesMap) {
        var module = new ModuleFactory(moduleHashMap);
        module.setProperties(propertiesMap);
        mFactories.add(module);
    }
}

/** Получение параметров главной сцены*/
private Map<String, BooleanProperty> getPropertiesMap() {
    BooleanProperty execButtonProperty = executeButton.
 disableProperty();
```

```
        BooleanProperty expandButtonProperty = expandButton.
      disableProperty();

        Map<String, BooleanProperty> propertiesMap = new HashMap<>();
        propertiesMap.put("execButtonProperty", execButtonProperty);
        propertiesMap.put("expandButtonProperty",
      expandButtonProperty);
        return propertiesMap;
    }

    /** Обработка нажатия на infoButton*/
    private void onInfoButtonClicked(final DialogType type){
        final var module = moduleTitlesComboBox.getValue();
        final var model = module != null? module.getCurrentModel():
      null;
        //экономим ресурсы для снижения частоты запросов к движку html
        if (type == DialogType.modelInfo && model != null){
            if (mInfoDialog.hasChanged(model.getModelDescription())){
                //Использование шаблона Builder
                mInfoDialog.setDescription(
module.getModuleDescription()+"<hr>"+model.getModelDescription()).
                        setIcon(model.getIcon()).
                        updateContent();
            }
        }
        else if (type == DialogType.programInfo){

      mInfoDialog.setDescription(type).setIcon(null).updateContent();
        }

        mInfoDialog.showAndWait();
    }
}
```

ПРИЛОЖЕНИЕ Г. Код класса AbstractModelController

```
/** Абстрактный класс контроллера модели*/
abstract public class AbstractModelController implements Initializable
     {
    /** Структура, содержащая настройки модели*/
    protected    Map<Label,    Control>    mModelSettings    =    new
     LinkedHashMap<>();
    /** Структура, хранящая атрибуты кнопок сцены для управления ими
     изнутри*/
    protected Map<String, BooleanProperty> mPropertiesMap;

    /**Инициализирует LinkedHashMap с ключом названием настройки
    и значением - кастомным виджетом настройки*/
    abstract protected void createSettings();
    /** Инициализация модели */
    abstract protected void construct();

    @Override
    final    public    void    initialize(URL    url,    ResourceBundle
     resourceBundle) {
        Platform.runLater(()->{ //Вызывается уже после установки сцены
            construct();
            createSettings();
            __setToolTips__();
        });
    }

    /** Установка всплывающих подсказок для панели настроек*/
    private void __setToolTips__() {
        for (final var labelSetting: mModelSettings.keySet()){
            labelSetting.setTooltip(new
     Tooltip(labelSetting.getText()));
            StatusBarController.connectToStatusBar(labelSetting);
        }
    }
    /** Получение настроек*/
    final public Map<Label, Control> getSettings(){
        return mModelSettings;
    }
    /** Функция активации - подразумевается, что каждый контроллер её
     имеет, но может и не иметь  */
    public void execute() {}
    /** Установка атрибутов сцены*/
    public     void     setProperties(Map<String,     BooleanProperty>
     propertiesMap) {
        mPropertiesMap = propertiesMap;
    }
    /** Связывание текстового поля настройки с изменяемыми атрибутами
     модели property*/
```

```
    final public void bidirectionalBinding(TextField field,
     Property<Number> property) {
        //field - то, что зависит; property - то, от чего зависит
        bidirectionalBinding(field, property, true);
    }
    /**
     * {@code disable} по умолчанию true.
     *   @see   AbstractModelController#bidirectionalBinding(TextField,
     Property<Number>)
     */
    final public void bidirectionalBinding(TextField field,
     Property<Number> property, boolean disable) {
        Bindings.bindBidirectional(field.textProperty(), property, new
     NumberStringConverter());
        field.setDisable(disable);
    }
}
```

ПРИЛОЖЕНИЕ Д. Код класса Model

```
/** Класс, хранящий и обрабатывающий характеристики физической модели
 *  Характеристики берутся из конфигурационного файла
*Модели отличаются только параметрами характеристик, поэтому
     наследование избыточно*/
public class Model {
    /** Название модели*/
    private final String mModelName;
    /** Описание модели*/
    private final String mModelDescription;
    /** Путь до .fxml файле модели*/
    private final String mModelFilePath;
    /** Объект сцены*/
    private Node mScene;
    /** Иконка модели*/
    private ImageView mIcon;
    /** Необходима ли для этой модели сетка*/
    private final boolean mIsGridNeeded;
    /**Была ли совершена попытка загрузки модели*/
    private boolean mTriedToLoad = false;
    /**  Структура, содержащая настройки модели (для передачи
     контроллеру)*/
    private Map<Label, Control> mSettingsMap;
    /** Объект управления моделью*/
    private AbstractModelController mController = null;
    /** Структура, хранящая атрибуты кнопок сцены для управления ими
     изнутри*/
    private Map<String, BooleanProperty> mPropertiesMap;
    public Model(Map<String, String> model){
        mModelName = model.get("modelName");
        mModelDescription = model.get("modelDescription");
        mModelFilePath = model.get("modelFilePath");
        String iconPath = model.get("iconPath");
        mIsGridNeeded  =  Boolean.parseBoolean(model.get("isGridNeeded
     "));
        initIcon(iconPath);
    }
    /** Инициализация иконки*/
    private void initIcon(String iconPath) {
        try {
            var  icon  =  new  Image(getClass().getResourceAsStream(
     iconPath));
            mIcon = new ImageView(icon);
            mIcon.setFitWidth(20);
            mIcon.setFitHeight(20);
        }
        catch (Exception ignored){}
    }
    /** Установка сцены*/
```

```
public void setScene(Node root){
    mScene = root;
}
/** Получение сцены*/
public Node getScene() {
    if (mScene == null && !mTriedToLoad) {
        mTriedToLoad = true;
        constructScene();
    }
    return mScene;
}
/** Инициализация сцены*/
private void constructScene() {
    //Для того чтобы URL != null, нужно в папке ресурсов иметь
    //такую же папочную структуру, как и в папке проекта,
    //иначе выдает исключение IllegalStateException
    try {
        FXMLLoader loader = new FXMLLoader(getClass().
                getResource(mModelFilePath));
        mScene = loader.load();
        mController = loader.getController();
        if (mController != null){
            mSettingsMap = mController.getSettings();
            mController.setProperties(mPropertiesMap);
        }
    } catch (IOException e) {
        Logger.displayOnStatusBar("Не загрузилась модель");
        Logger.log("Не загрузилась модель " + mModelName +
 "\nПричина: " +Logger.formatStringWithLF(e.getCause().toString(),
 3) + "\nВместо сцены возвращается null");
    } catch (IllegalStateException e) {
        Logger.log("Программа не смогла найти путь до модели");
    } catch (NullPointerException e) {
        String message = "Какой-то объект при инициализации равен
 null";
        Logger.displayOnStatusBar(message);
        Logger.log(message);
        Logger.log("Модель: " + mModelName);
        Logger.log("Путь: " + mModelFilePath);
        e.printStackTrace();
    } catch (ClassCastException e) {
        Logger.log("Ошибка кастинга в " + e.toString().split(":")
 [1].trim().split (" ")[1]);
    } catch (Exception e) {
        String message = "Необработанное исключение :(";
        Logger.displayOnStatusBar(message);
        Logger.log(message);
        Logger.log(Logger.formatStringWithLF(e.toString(), 3));
    }
}
```

```
public String getModelName() {
    return mModelName;
}
public ImageView getIcon() {
    return mIcon;
}
public String getModelDescription(){
    return mModelDescription;
}
public boolean isGridNeeded(){
    return mIsGridNeeded;
}
public Map<Label, Control> getSettings(){
    return mSettingsMap;
}
/** Установка атрибутов сцены*/
public     void     setProperties(Map<String,     BooleanProperty>
 propertiesMap){
    mPropertiesMap = propertiesMap;
}
/** Функция активации*/
public void execute(){
    if (mController != null){
        mController.execute();
    }
    else   Logger.displayOnAlertWindow("Демонстрация   невозможна,
 т.к. mController == null");
}}
```

ПРИЛОЖЕНИЕ Е. Код класса CannonballController

```
/** Модель, анализирующая параболическое движение снаряда*/
public class CannonballController extends AbstractModelController {
    @FXML private ImageView wheel;
    @FXML private ImageView floor;
    @FXML private Pane borderPane;
    @FXML private Pane barrelPane;

    /** Координаты зажатой ЛКМ*/
    private final Point mStartP = new Point();
    /** Сущность для управления поворотом ствола*/
    private final Rotate mRotate = new Rotate();
    /** Объект, отвечающий за прорисовку кривой траектории снаряда*/
    private TrajectoryLine trLine;
    /** Точка, от которой начинается отсчет кривой траектории*/
    private final ImageView mTrackingPoint = new ImageView();
    /** Точка, совпадающая с точкой вращения*/
    private final ImageView mPivotPoint = new ImageView();
    /** Начался ли процесс бросания снаряда*/
    private boolean mIsTransitionStarted = false;
    /** Изображение снаряда*/
    private ImageView mProjectile;

    @Override
    protected void construct() {
        Logger.log("Загрузилась модель пушечного ядра");
        barrelPane.getTransforms().add(mRotate);
        mProjectile = getProjectileObject();

        //Определяем точку опоры
        mRotate.setPivotX(barrelPane.getWidth()/10);
        mRotate.setPivotY(barrelPane.getHeight()/2);

        setTrackingPoint(mTrackingPoint, true);
        setTrackingPoint(mPivotPoint, false);
        trLine = new TrajectoryLine();

        barrelPane.addEventHandler(MouseEvent.MOUSE_PRESSED,
     this::setMouse);
        // Когда край пушки перетаскивается, вращаем её
        barrelPane.addEventHandler(MouseEvent.MOUSE_DRAGGED,
     this::barrelDragged);
        barrelPane.setOnMouseClicked(e->{
            if (e.getButton().equals(MouseButton.PRIMARY)){
                if (e.getClickCount() == 2 && !mIsTransitionStarted){
                    execute();
                }
            }
        });
```

```
}

/** Обработчик перетаскивания ствола*/
private void barrelDragged(final MouseEvent event){
    if (!mIsTransitionStarted){
        /*
         Используется для получения положения крайнего угла
 объекта на сцене
         Аффинное преобразование - это линейное, которое переводит
 объект из двумерного(трех-) пространство в другое двумерное
 пространство, сохраняя неизменной прямолинейность и
 параллельность линий. Одна точка A с координатой (x, y, z) в
 трехмерном пространстве в позицию A' с координатой (x', y', z')
 путем матричного умножения

        final Transform localToScene =
 barrelPane.getLocalToSceneTransform();

        //конечное положение точки (на момент конца локального
 перемещения)
        //координаты берутся относительно берущегося предмета
        final double endX = event.getSceneX();
        final double endY = event.getSceneY();
        //Аффинные преобразования
        //Параметры tx, ty обозначают трансформированные точки для
 ствола
        final double px = mRotate.getPivotX() +
 localToScene.getTx();
        //Получает элемент преобразования координаты Y матрицы 3x4
 + координаты т. поворота (она постоянна)
        final double py = mRotate.getPivotY() +
 localToScene.getTy();

        // Определение углов поворота
        final double th1 = clockAngle(mStartP.x - px, mStartP.y -
 py);
        final double th2 = clockAngle(endX - px, endY - py);

        final double angle = mRotate.getAngle() + th2 - th1;
        //для изменения положения только в первой четверти
        if (angle <= 0 && angle >= -90){
            mRotate.setAngle(angle);
        }
        setMouse(event);
    }
}

private void setMouse(final MouseEvent e){
    mStartP.setCoord(e);
}
```

```
/** Определение угла поворота по двум точкам */
private double clockAngle(double dx, double dy) {
    double angle = Math.abs(Math.toDegrees(Math.atan2(dy, dx)));

    if (dy < 0) {
        angle = 360 - angle;
    }
    return angle;
}

/** Установка отслеживающих точек для расчета высоты*/
private void setTrackingPoint(ImageView point, boolean isTracking)
 {
    final int SIZE = 1;
    final int w = (int) barrelPane.getWidth();
    point.setFitWidth(SIZE);
    point.setFitHeight(SIZE);
    point.setLayoutX(isTracking ? w * 0.9 : mRotate.getPivotX());
    point.setLayoutY(mRotate.getPivotY());
    barrelPane.getChildren().add(point);
}

@Override
protected void createSettings() {
    final var angleSpinner = new Spinner<Double>();
    final var speedText = new TextField();
    final var durationField = new TextField();
    final var distanceField = new TextField();
    final var heightField = new TextField();
    final var maxHeightField = new TextField();

    mModelSettings.put(new     Label("Угол     наклона     ствола"),
 angleSpinner);
    mModelSettings.put(new   Label("Начальная   скорость   снаряда
 [м/с]"), speedText);
    mModelSettings.put(new      Label("Время      полёта      [с]"),
 durationField);
    mModelSettings.put(new Label("Дистанция [м]"), distanceField);
    mModelSettings.put(new    Label("Изначальная    высота    [м]"),
 heightField);
    mModelSettings.put(new   Label("Максимальная   высота   [м]"),
 maxHeightField);

    bidirectionalBinding(speedText,  trLine.initVelocityProperty(),
 false);
    bidirectionalBinding(durationField,
 trLine.durationProperty());
    bidirectionalBinding(distanceField,
 trLine.distanceProperty());
```

```
    bidirectionalBinding(heightField,
 trLine.initHeightProperty());
    bidirectionalBinding(maxHeightField,
 trLine.maxHeightProperty());
    speedText.focusedProperty().addListener((obs, oldV, newV)->{
        if (oldV){ //если фокус убран, меняем значение
            trLine.calculateTrajectory();
        }
    });
    speedText.setOnKeyPressed(e->{
        if (e.getCode().equals(KeyCode.ENTER)){
            trLine.calculateTrajectory();
        }
    });

    SpinnerValueFactory<Double> valueFactory =
            new    SpinnerValueFactory.DoubleSpinnerValueFactory(0,
 90, 0);
    angleSpinner.setValueFactory(valueFactory);
    angleSpinner.setEditable(true);
    mRotate.setOnTransformChanged(e-> {
        valueFactory.setValue(-mRotate.getAngle());
        trLine.calculateTrajectory();
    });
    valueFactory.valueProperty().addListener(e->
            mRotate.setAngle(-valueFactory.getValue()));
}

@Override
public void execute() {
    var execProp = mPropertiesMap.get("execButtonProperty");
    var expandProp = mPropertiesMap.get("expandButtonProperty");
    final var path = trLine.getPath();
    path.setVisible(false);
    mProjectile.setVisible(true);
    execProp.set(true);
    expandProp.set(true);
    mIsTransitionStarted = true;
    PathTransition  trans  =  new  PathTransition(Duration.seconds
 (trLine. durationProperty().get()), path, mProjectile);
    trans.setInterpolator(Interpolator.LINEAR);
    trans.setOnFinished(e->{
        mIsTransitionStarted = false;
        path.setVisible(true);
        execProp.set(false);
        expandProp.set(false);
        trLine.calculateTrajectory();
    });
    trans.play();
}
```

```
/** Получение объекта снаряда*/
private ImageView getProjectileObject() {
    final var input =
 getClass().getResourceAsStream("/root/img/cannon/
 projectile.png");
    var projectile = input != null? new ImageView(new
 Image(input)): null;
    if (projectile != null) {
        projectile.setFitWidth(50);
        projectile.setFitHeight(50);
        projectile.setVisible(false);
        borderPane.getChildren().add(projectile);
    }
    return projectile;
}

/** Класс, отвечающий за прорисовку кривой траектории*/
private class TrajectoryLine{
    /** Структура, хранящая передвижения кривой траектории */
    private final Path mPath = new Path();
    /** Кол-во пикселей на 1 метр */
    @SuppressWarnings("FieldCanBeLocal")
    private final int PIXELS_PER_METER = Constants.
 PIXELS_PER_UNIT;
    /** Частота обновления значений (чем меньше значение, тем
 более гладкая кривая)*/
    @SuppressWarnings("FieldCanBeLocal")
    private final double FREQUENCY = 0.005;
    /** Высота крепления ствола в метрах*/
    private final double HEIGHT = wheel.getFitHeight() / 2 /
 PIXELS_PER_METER;
    /** Изначальная скорость снаряда */
    private final IntegerProperty mInitVelocity;
    /** Разница между начальной позиции и текущей по оси Y (в
 пикселах)*/
    private double mInitHeight;
    /** Время полета снаряда в секундах*/
    private final DoubleProperty mTimeFlight;
    /** Дальность полета в метрах*/
    private final DoubleProperty mDistance;
    /** Итоговая высота с учетом наклона ствола в метрах*/
    private final DoubleProperty mTotalHeight;
    /** Максимальная высота в метрах*/
    private final DoubleProperty mMaxHeight;
    public TrajectoryLine(){
        mPath.setStroke(Color.RED);
        mPath.setStrokeWidth(3);
        borderPane.getChildren().add(mPath);
        mPath.getStrokeDashArray().addAll(10d, 10d);
```

```
        mInitVelocity     =     new     SimpleIntegerProperty(this,
"initVelocity", 6);
        mTimeFlight = new SimpleDoubleProperty(this, "duration");
        mDistance = new SimpleDoubleProperty(this, "distance");
        mTotalHeight = new SimpleDoubleProperty(this, "totalHeight
");
        mMaxHeight = new SimpleDoubleProperty(this, "maxHeight");
        mInitHeight                                                =
mPivotPoint.localToScene(mPivotPoint.getBoundsInLocal()).
getCenterY();
        Stage stage = (Stage) borderPane.getScene().getWindow();
    //  Т.к.  значения  GUI  обновляются  уже  после  максимизации
сцены, а не во время
        // нее, надо использовать runLater()
        stage.maximizedProperty().addListener((obs)              ->
Platform.runLater(()->{
            mProjectile.setVisible(false);
           //  эту  переменную  надо  обновлять  каждый  раз  при
изменении размера окна
            mInitHeight                                            =
mPivotPoint.localToScene(mPivotPoint.getBoundsInLocal()
).getCenterY();
            calculateTrajectory();
        }));
        calculateTrajectory();
    }
    /**  Метод  для  расчёта  координат  кривой  траектории  и  ее
дальнейшей прорисовки.
     *  Описание логики см. в конфиг. файле
     */
    public void calculateTrajectory() {
        final var pathElements = mPath.getElements();
        pathElements.clear();
        final double angle = -mRotate.getAngle();
        //Местонахождение точки начала траектории в пространстве
сцены
        final              Bounds              bounds              =
mTrackingPoint.localToScene(mTrackingPoint. getBoundsInLocal());
        //Разница по оси Y между точкой крепления ствола и концом
ствола
        final double dHeight = (mInitHeight - bounds.getCenterY())
/ PIXELS_PER_METER;
        mTotalHeight.set(HEIGHT+dHeight);
        //Вертикальная составляющая скорости
        final  double  V_y  =  (double)  mInitVelocity.get()  *
Math.sin(Math.toRadians(angle));
        //Горизонтальная составляющая скорости
        final  double  V_x  =  (double)  mInitVelocity.get()  *
Math.cos(Math.toRadians(angle));
```

```
            mTimeFlight.set((V_y + Math.sqrt(Math.pow(V_y, 2) +
        2*Constants.g*mTotalHeight.get()))/Constants.g);
            mDistance.set(V_x * mTimeFlight.get());
            mMaxHeight.set(mTotalHeight.get() + Math.pow(V_y,
        2)/(2*Constants.g));
            //Смещение точки начала полета по оси X
            final double deltaX = bounds.getCenterX();
            //Смещение точки начала полета по оси Y
            final double deltaY = bounds.getCenterY() -
        barrelPane.getHeight()/2 + dHeight*PIXELS_PER_METER;
            //Координата по оси Y, при которой траектория будет
        кончаться при соприкосновении с полом
           final double neededHeight = borderPane.getHeight() -
        floor.getFitHeight();
            //Множитель для значения X, чтобы не считать его в цикле
            final double xCoeff = mDistance.get() / mTimeFlight.get()
        * PIXELS_PER_METER;
            for (double time = 0, x, y = 0; y < neededHeight; time +=
        FREQUENCY){
                x = time * xCoeff + deltaX;
                y = -(mTotalHeight.get() + V_y * time -
        Constants.g*Math.pow(time, 2)/2) * PIXELS_PER_METER + deltaY;
                pathElements.add((time < FREQUENCY - 0.001)? new
        MoveTo(x, y): new LineTo(x, y));
            }
        }
        public IntegerProperty initVelocityProperty(){
            return mInitVelocity;
        }
        public DoubleProperty durationProperty(){
            return mTimeFlight;
        }
        public DoubleProperty distanceProperty(){
            return mDistance;
        }
        public Path getPath() {
            return mPath;
        }
        public DoubleProperty initHeightProperty(){
            return mTotalHeight;
        }
        public DoubleProperty maxHeightProperty(){
            return mMaxHeight;
        }
    }
}
```

ПРИЛОЖЕНИЕ Ё. Код класса WeightWithSpringController

```
//Есть два вида растяжения: растяжение от груза и растяжение от
     приложенной силы
//Второе можно применять только при перетягивании груза зажатой мышью
//Растяжение от груза задается через ввод настроек,
//ручное растяжение - при перетаскивании предмета
//А итоговое растяжение получается из суммы двух растяжений
//Ручное удлинение считается как разница между положением груза при
     перемещении
//и при расположении без перетягивания, деленная на количество
     пикселей на один метр
/** Модель, описывающая зависимость растяжения пружины от силы
     упругости */
public class WeightWithSpringController extends
     AbstractModelController {
    @FXML
    private Pane topPane;
    @FXML
    private Polygon trapezoid;
    @FXML
    private ImageView spring;
    @FXML
    private Pane hbox;

    /** Координаты зажатой ЛКМ*/
    private final Point mStartP = new Point();
    /**Начальная высота груза */
    private int mInitY;
    /** Величина в пикселах, на которую изображение груза
     * может накладываться на изображение пружины */
    private int mSpringLambda;
    /** Положение по оси Y только при удлинении весом груза*/
    private int mWeightY;
    /** Вес груза в килограммах */
    private DoubleProperty mCargoWeight;
    /** Жёсткость пружины */
    private IntegerProperty mStiffness;
    /** Прилагаемая сила в Ньютонах*/
    private DoubleProperty mManualPower;
    /** Удлинение пружины от груза в метрах */
    private DoubleProperty mWeightExtension;
    /** Удлинение пружины усилиями пользователя в метрах */
    private DoubleProperty mManualExtension;
    /** Итоговое удлинение пружины в метрах */
    private DoubleProperty mTotalSpringExtension;
    /** Количество пикселей на один метр (1 у.е. = 5 см = 0.05 м)*/
    private final int pixelsPerMeter =
     (int)(Constants.PIXELS_PER_UNIT/0.05);
```

```
@Override
protected void construct() {
    hbox.minWidthProperty().bind(hbox.getScene().widthProperty());

    mInitY = (int)trapezoid.getLayoutY();
    mSpringLambda = (int)(spring.getFitHeight() - mInitY);

    mCargoWeight = new SimpleDoubleProperty(this, "cargoWeight",
 1.52);
    mStiffness = new SimpleIntegerProperty(this, "stiffness",
 300);
    mManualPower = new SimpleDoubleProperty(this, "manualPower");
    mManualExtension = new SimpleDoubleProperty(this,
 "manualExtension");
    mWeightExtension = new SimpleDoubleProperty(this,
 "weightExtension");
    mTotalSpringExtension = new SimpleDoubleProperty(this,
 "springExtension");

    calculate();

    trapezoid.addEventHandler(MouseEvent.MOUSE_PRESSED,
 this::setMouse);
    trapezoid.addEventHandler(MouseEvent.MOUSE_DRAGGED,
 this::objectDragged);
    trapezoid.addEventHandler(MouseEvent.MOUSE_RELEASED,
 this::objectReleased);
}

//Ручное удлинение задается до вызова этого метода
/* Метод для расчета координат объектов модели и их параметров */
private void calculate() {
    final double weightExtension = mCargoWeight.get() *
 Constants.g / mStiffness.get();
    mWeightExtension.set(weightExtension);
    mWeightY = (int)(mInitY + mWeightExtension.get()*
 pixelsPerMeter);
    final double manualPower = mManualExtension.get() *
 mStiffness.get();
    mManualPower.set(manualPower);
    mTotalSpringExtension.set(mManualExtension.get() +
 mWeightExtension.get());
    final double totalExtensionInPX = mTotalSpringExtension.get()
 *pixelsPerMeter;
    trapezoid.setLayoutY(mInitY + totalExtensionInPX);
    spring.setFitHeight(mSpringLambda + trapezoid.getLayoutY());
}
@Override
protected void createSettings() {
    final var weightText = new TextField();
```

```
    final var springStiffnessText = new TextField();
    final var manualPowerText = new TextField();
    final var springExtensionText = new TextField();
    mModelSettings.put(new   Label("Жёсткость   пружины   [Н/м]"),
 springStiffnessText);
    mModelSettings.put(new Label("Масса груза [кг]"), weightText);
    mModelSettings.put(new    Label("Приложенная    сила    [Н]"),
 manualPowerText);
    mModelSettings.put(new    Label("Удлинение    пружины    [м]"),
 springExtensionText);
    bidirectionalBinding(springStiffnessText, mStiffness, false);
    bidirectionalBinding(weightText, mCargoWeight, false);
    bidirectionalBinding(manualPowerText, mManualPower);
    bidirectionalBinding(springExtensionText,
 mTotalSpringExtension);
    weightText.focusedProperty().addListener((obs, oldV, newV)->{
        if (oldV){ //если фокус убран, меняем значение
            mManualExtension.set(0);
            calculate();
        }
    });
    weightText.setOnKeyPressed(e->{
        if (e.getCode().equals(KeyCode.ENTER)){
            mManualExtension.set(0);
            calculate();
        }
    });
    springStiffnessText.focusedProperty().addListener((obs,   oldV,
 newV)->{
        if (oldV){ //если фокус убран, меняем значение
            mManualExtension.set(0);
            calculate();
        }
    });
    springStiffnessText.setOnKeyPressed(e->{
        if (e.getCode().equals(KeyCode.ENTER)){
            mManualExtension.set(0);
            calculate();
        }
    });
}
private void setMouse(MouseEvent event) {
    mStartP.setCoord(event);
}
/** Обработчик перетаскивания груза */
private void objectDragged(MouseEvent event) {
    final double endY = event.getSceneY();
    final double dy = trapezoid.getLayoutY() - mStartP.y;
    final double finalY = endY + dy;
    if (finalY > mInitY){
```

```
            mManualExtension.set((finalY - mWeightY)/ pixelsPerMeter);
            calculate();
        }
        setMouse(event);
    }
    /** Обработчик перемещения груза после разжатия мыши*/
    // Алгоритм следующий: после разжатия мыши груз начинает двигаться
     в
    //  противоположном направлении до рассчитаной величины, затем
     двигается в
    // противоположную сторону, и так до тех пор, пока
    // расстояние перемещения не становится слишком маленьким
    private void objectReleased(MouseEvent event) {
        final          double          manualExtensionInPX          =
     mManualExtension.get()*pixelsPerMeter*0.5;
        final double tempDeltaY = manualExtensionInPX > mWeightY-
     mInitY? mWeightY-mInitY: manualExtensionInPX;
        //Итоговое растяжение в пикселах за одну итерацию (которое
     будет меняться в другом потоке)
        final double pxPath = mManualExtension.get()*pixelsPerMeter +
     tempDeltaY;
        //Текущая скорость - 150 px/сек
        final double timeForT = 150; //В милисекундах
        Thread motionThread = new Thread(() -> {
            // Дистанция в пискелах, которую надо пройти
            double pixelPath = Math.abs(pxPath);
            // Если <0, груз движется вверх, иначе вниз
            int deltaP = pxPath < 0 ? 1 : -1;
            final int pixelLambda = 2;
            while (pixelPath > pixelLambda) {
                // Груз должен идти вверх в два раза быстрее, чем вниз
                int speedTime = (int) (deltaP < 0 ? timeForT / 2 :
     timeForT);
                // Время, на которое должен прерываться поток
                final int deltaT = (int)(speedTime / pixelPath);
            //  Координата Y, по которой груз перестает двигаться и
     начинает двигаться в
            // другую сторону
                final   double   cancelY   =   trapezoid.getLayoutY()   +
     pixelPath * deltaP;
                for  (int  i  =  0;  Math.abs(trapezoid.getLayoutY()  -
     cancelY) > 1; i++) {
                    // Координата Y на след. итерации
                    final  double  deltaY  =  trapezoid.getLayoutY()  +
     deltaP;
                    trapezoid.setLayoutY(deltaY);
                    spring.setFitHeight(deltaY + mSpringLambda);
                    try {
                        Thread.sleep(deltaT == 0 ? i % 2 :deltaT);
                    } catch (InterruptedException e) {
```

```
                        e.printStackTrace();
                    }
                }
                pixelPath    =    Math.abs(mWeightY    -    trapezoid.
     getLayoutY()) * 1.5;
                deltaP *= -1;
            }
            trapezoid.setLayoutY(mWeightY);
            spring.setFitHeight(mWeightY + mSpringLambda);
            mManualPower.set(0);
            mTotalSpringExtension.set(mWeightExtension.get());
        });
        motionThread.start();
    }
}
```

ПРИЛОЖЕНИЕ Ж. Код класса RefractionThroughLensController

```
/** Модель, позволяющая исследовать закономерность между расстоянием
     до
 * линзы и размером отображающегося изображения
 */
public class RefractionThroughLensController extends
     AbstractModelController {
    @FXML
    private Pane anchorPane;
    @FXML
    private Line OXline;
    @FXML
    private Line lensLine;

    /** Координаты зажатой ЛКМ */
    private final Point mStartP = new Point();
    /** Объект полигональной стрелки-объекта */
    private Polygon mPolygonArrow;
    /** Объект полигональной стрелки-изображения */
    private PolygonArrow mReflectionArrow;
    /** Линия от объекта до линзы */
    private Line mFromObjectToLensLine;
    /** Линия от линзы до фокуса линзы */
    private LineWithOdds mFromLensToFocusLine;
    /** Линия от объекта до оптического центра линзы */
    private LineWithOdds mFromObjectToOXYLine;

    // Изначальное значение - двойное фокусное расстояние
    /** Расстояние до линзы в метрах */
    private DoubleProperty mLensDistance;
    /** Кешируемое значение mLensDistance */
    private Number oldLensDistanceValue;
    /** Оптическая сила линзы в диоптриях*/
    private DoubleProperty mOpticalPower;
    /** Доля в процентах от размера предмета */
    private IntegerProperty mPartOfOriginalSize;

    @Override
    protected void construct() {
        mPolygonArrow = new PolygonArrow(Color.RED);
        anchorPane.getChildren().add(mPolygonArrow);
        mPolygonArrow.setLayoutX(3 * Constants.PIXELS_PER_UNIT);
        mPolygonArrow.setLayoutY(3 * Constants.PIXELS_PER_UNIT - 10);
        mReflectionArrow = new PolygonArrow(Color.rgb(255, 0, 0,
     0.25));
        mReflectionArrow.setMouseTransparent(true);
        anchorPane.getChildren().add(mReflectionArrow);
```

```
    OXline.endXProperty().bind(anchorPane.widthProperty()/*
  .divide(2) */);

  lensLine.endYProperty().bind(lensLine.getScene().heightProperty()
  );
    lensLine.setLayoutX(5 * Constants.PIXELS_PER_UNIT);

    mFromObjectToLensLine = new Line(mPolygonArrow.getLayoutX(),
  mPolygonArrow.getLayoutY(),                lensLine.getLayoutX(),
  mPolygonArrow.getLayoutY());
    mFromObjectToLensLine.setStroke(Color.rgb(130, 130, 130));
    mFromLensToFocusLine = new LineWithOdds(lensLine.getLayoutX(),
  mPolygonArrow.getLayoutY(),        lensLine.getLayoutX()        +
  Constants.PIXELS_PER_UNIT, OXline.getLayoutY());
    mFromLensToFocusLine.setStroke(Color.rgb(130, 130, 130));
    mFromObjectToOXYLine                    =                    new
  LineWithOdds(mPolygonArrow.getLayoutX(),
  mPolygonArrow.getLayoutY(),                lensLine.getLayoutX(),
  OXline.getLayoutY());
    mFromObjectToOXYLine.setStroke(Color.rgb(130, 130, 130));

    anchorPane.getChildren().add(mFromObjectToLensLine);
    anchorPane.getChildren().add(mFromLensToFocusLine);
    anchorPane.getChildren().add(mFromObjectToOXYLine);

    mLensDistance = new SimpleDoubleProperty(this, "lensDistance",
  0.2);
    mOpticalPower = new SimpleDoubleProperty(this, "opticalPower",
  10);
    mPartOfOriginalSize    =    new    SimpleIntegerProperty(this,
  "partOfOriginalSize", 100);

    calculate();

    mPolygonArrow.addEventHandler(MouseEvent.MOUSE_PRESSED,
  this::setMouse);
    mPolygonArrow.addEventHandler(MouseEvent.MOUSE_DRAGGED,
  this::arrowDragged);
}

@Override
protected void createSettings() {
    final var lensDistanceText = new TextField();
    final var opticalPowerText = new TextField();
    final var partOfOriginalSizeText = new TextField();

    mModelSettings.put(new  Label("Расстояние  от  линзы  [м]"),
  lensDistanceText);
    mModelSettings.put(new Label("Оптическая сила линзы [Дптр]"),
  opticalPowerText);
```

```
  mModelSettings.put(new Label("Доля от изначального размера
[%]"), partOfOriginalSizeText);

  bidirectionalBinding(lensDistanceText, mLensDistance, false);
  bidirectionalBinding(opticalPowerText, mOpticalPower, false);
  bidirectionalBinding(partOfOriginalSizeText,
mPartOfOriginalSize);

  HandleTextInput handler = () -> {
      final    double    dist    =    mLensDistance.get()    *
Constants.PIXELS_PER_UNIT * mOpticalPower.get();
      final double startX = mFromObjectToLensLine.getEndX() -
dist;
      final double focalDist = 4 * Constants.PIXELS_PER_UNIT;
      final int lambda = 5;
      if (Math.abs(startX - focalDist) < lambda) {
          mLensDistance.set((Double) oldLensDistanceValue);
          Logger.displayOnAlertWindow("Нельзя      устанавливать
объект на фокусном расстоянии");
      } else if (mLensDistance.get() <= 0) {
          mLensDistance.set((Double) oldLensDistanceValue);
          Logger.displayOnAlertWindow("Дистанция до линзы должна
быть больше 0");
      }
      else {
          mPolygonArrow.setLayoutX(startX);
          calculate();
      }
  };

  lensDistanceText.focusedProperty().addListener((obs,      oldV,
newV)->{
      if (oldV){ //если фокус убран, меняем значение
          handler.handleTextInput();
      }
      else{
          oldLensDistanceValue = mLensDistance.get();
      }
  });
  lensDistanceText.setOnKeyPressed(e->{
      if (e.getCode().equals(KeyCode.ENTER)){
          handler.handleTextInput();
      }
  });
  opticalPowerText.focusedProperty().addListener((obs,      oldV,
newV)->{
      if (oldV){ //если фокус убран, меняем значение
          calculate();
      }
  });
```

```
    opticalPowerText.setOnKeyPressed(e->{
        if (e.getCode().equals(KeyCode.ENTER)){
            calculate();
        }
    });
}

/*Инструмент для метода обратного вызова */
private interface HandleTextInput{
    public void handleTextInput();
}

private void setMouse(MouseEvent event) {
    mStartP.setCoord(event);
}

/** Обработчик перетаскивания предмета*/
private void arrowDragged(MouseEvent event) {
    final double endX = event.getSceneX();
    //Если dx > 0, объект движется справа налево, иначе слева
 направо
    final double dx = mPolygonArrow.getLayoutX() - mStartP.x;
    final double layoutX = endX + dx;
    if (layoutX < lensLine.getLayoutX() - 1) {
        final int lambda = 5;
        final double focalDist = 4*Constants.PIXELS_PER_UNIT;
        //Если объект близок к фокусному расстоянию, то
        //если dx>0, его надо сдвинуть левее на две лямбды, иначе
        //если dx<0, его надо сдвинуть правее на то же расстояние
        if (Math.abs(layoutX - focalDist) < lambda){
            if (dx > 0){
                mPolygonArrow.setLayoutX(layoutX - 2*lambda);
            }
            else if (dx < 0){
                mPolygonArrow.setLayoutX(layoutX + 2*lambda);
            }
        }
        else mPolygonArrow.setLayoutX(layoutX);
        calculate();
    }

    setMouse(event);
}

/* Метод для расчета координат объектов модели и их параметров */
private void calculate() {
    mFromObjectToLensLine.setStartX(mPolygonArrow.getLayoutX());
    mFromObjectToOXYLine.setStartX(mPolygonArrow.getLayoutX());
    mFromObjectToOXYLine.setEndX(lensLine.getLayoutX());
    mFromObjectToOXYLine.setEndY(OXline.getLayoutY());
```

```
            final Point p = LineWithOdds.getIntersectionPoint(
        mFromLensToFocusLine, mFromObjectToOXYLine);
            mFromLensToFocusLine.setEndX(p.x);
            mFromLensToFocusLine.setEndY(p.y);
            mFromObjectToOXYLine.setEndX(p.x);
            mFromObjectToOXYLine.setEndY(p.y);

            final double distance = (mFromObjectToLensLine.getEndX() -
        mFromObjectToLensLine.getStartX()) / mOpticalPower.get();
            mLensDistance.set(distance / Constants.PIXELS_PER_UNIT);
            final double originalHeight = OXline.getLayoutY() -
        mFromObjectToLensLine.getEndY();
            final double distortedHeight = p.y - OXline.getLayoutY();
            final double percentage = (distortedHeight / originalHeight);
            mPartOfOriginalSize.set((int)Math.abs(percentage*100));

            final int reflectionHeight = (int)Math.abs(originalHeight *
        percentage);
            mReflectionArrow.setHeight((int)reflectionHeight);
            mReflectionArrow.setLayoutX(p.x);
            //Если расстояние от линзы больше фокусного расстояния, тогда
        всё хорошо, иначе
            //задать поворот стрелки в 0 градусов и задать другие
        координаты отображения
            final double focalDist = 4*Constants.PIXELS_PER_UNIT;
            if (mPolygonArrow.getLayoutX() - focalDist < 0){
                mReflectionArrow.setRotate(180);
                mReflectionArrow.setLayoutY(OXline.getLayoutY());
            }
            else{
                mReflectionArrow.setRotate(0);
                mReflectionArrow.setLayoutY(p.y);
            }
        }
    }
```