**Федеральное государственное бюджетное образовательное учреждение высшего образования «Московский государственный университет технологий и управления имени К.Г. Разумовского**

**(Первый казачий университет)»**

|  |
| --- |
| Университетский колледж информационных технологий |

**Отчет**

**О прохождении практики**

**УП.02.01 Учебная практика**

**по профессиональному модулю**

**ПМ.02 Разработка модулей программного обеспечения для компьютерных систем**

*(вид, тип практики)*

|  |
| --- |
| **Обучающимся Поликановым Никитой Александровичем** |
| Направления подготовки/специальности  **09.02.07 Информационные системы и программирование** |
| Профиль подготовки (специализация/квалификация) программист |
| Форма обучения очная |
| Курс 3 |
| Группа 090207–9о–пр–22/2 |

практикант, ФГБОУ ВО «МГУТУ им. К.Г. Разумовского (ПКУ)» УКИТ

*(должность в которой проходил практику, наименование организации/предприятия)*

с «09» июня 2025 г. по «28» июня 2025 г.

Подпись обучающегося

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_Н.А. Поликанов\_\_\_\_\_

*(подпись) (Ф.И.О.)*

«⠀⠀⠀⠀» ⠀⠀⠀⠀⠀⠀⠀⠀⠀2025 г.

Подпись руководителя

практики от Университета

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ ⠀⠀ И.А. Озеркова⠀\_\_\_\_\_

*(подпись) (Ф.И.О.)*

«⠀⠀⠀⠀» ⠀⠀⠀⠀⠀⠀⠀⠀⠀2025 г.

Отчет принял

Заместитель директора по учебно–методической работе

Университетского колледжа информационных технологий

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ ⠀⠀⠀⠀⠀ Е.В. Вернер⠀⠀⠀⠀\_⠀⠀

*(подпись) (Ф.И.О.)*

«⠀⠀⠀⠀» ⠀⠀⠀⠀⠀⠀⠀⠀⠀2025 г.

**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 3](#_Toc201676358)

[ОСНОВНАЯ ЧАСТЬ 6](#_Toc201676359)

[1. Анализ предметной области 6](#_Toc201676360)

[1.1 Общая постановка задач 6](#_Toc201676361)

[1.2 Уточнение постановки задачи 6](#_Toc201676362)

[1.3 Математическая модель 8](#_Toc201676363)

[2. Обоснование выбора средств методов и средств 9](#_Toc201676364)

[2.1 Описание метода решения 9](#_Toc201676365)

[2.2 Анализ средств разработки 10](#_Toc201676366)

[3. Проектирование приложения 11](#_Toc201676367)

[3.1 UML–диаграммы 11](#_Toc201676368)

[3.2. Wireframe–эскизы приложения 15](#_Toc201676369)

[3.3 Описание выбранных форматов данных 18](#_Toc201676370)

[4. Разработка алгоритмов 19](#_Toc201676371)

[5. Разработка приложения 22](#_Toc201676372)

[6. Тестирование приложения 27](#_Toc201676373)

[7. Анализ качества моделирования 30](#_Toc201676374)

[ЗАКЛЮЧЕНИЕ 31](#_Toc201676375)

[СПИСОК ЛИТЕРАТУРЫ 33](#_Toc201676376)

[ПРИЛОЖЕНИЕ А Техническое задание 35](#_Toc201676377)

[Приложение Б код приложения 44](#_Toc201676378)

# ВВЕДЕНИЕ

Учебная практика УП02.01 проходила в лаборатории колледжа «Университетский колледж информационных технологий им. К.Г. Разумовского».

Цель практики: создание АС включающую в себя модули моделирования и анализа. Автоматизированная система ориентирована на визуализацию метода работы, демонстрацию процесса анализа, основанного на вычислении по математической формуле и методом Монте–Карло площадь большого сегмента, ограниченного окружностью радиуса R с центром в точке (x0, y0) и вертикальной или горизонтальной прямой, проходящей на заданном расстоянии C от соответствующей оси координат.

Задачи практики:

- определение требований для выполнения поставленной задачи;

- составление технического задания на основе поставленной задачи преподавателем;

- изучение математических аспектов вычисления площади большего сегмента фигуры;

- моделирование и создание графического интерфейса, комфортное для понимания и использования пользователем;

- разработка программного обеспечения с использованием метода Монте–Карло для вычисления площади большего сегмента;

- реализация форм в приложении: заставки, главного модуля с вводом количества точек, численного и графического моделирования, сохранения результатов, а также форм анализа моделирования и информации о программе;

- обеспечение разработанной программы встроенной справкой для помощи пользователям.

В ходе практики предусматривалось развитие следующих профессиональных компетенций:

ПК 2.1. Разрабатывать требования к программным модулям на основе анализа проектной и технической документации на предмет взаимодействия компонент.

ПК 2.2. Выполнять интеграцию модулей в программное обеспечение.

ПК 2.3. Выполнять отладку программного модуля с использованием специализированных программных средств.

ПК 2.4. Осуществлять разработку тестовых наборов и тестовых сценариев для программного обеспечения.

ПК 2.5. Производить инспектирование компонент программного обеспечения на предмет соответствия стандартам кодирования.

Также были продемонстрированы следующие общие компетенции:

ОК 01 Выбирать способы решения задач профессиональной деятельности, применительно к различным контекстам.

ОК 02 Осуществлять поиск, анализ и интерпретацию информации, необходимой для выполнения задач профессиональной деятельности

ОК 03 Планировать и реализовывать собственное профессиональное и личностное развитие

ОК 04 Работать в коллективе и команде, эффективно взаимодействовать с коллегами, руководством,

ОК 05 Осуществлять устную и письменную коммуникацию на государственном языке с учетом особенностей социального и культурного контекста

ОК 06 Проявлять гражданскопатриотическую позицию, демонстрировать осознанное поведение на основе традиционных общечеловеческих ценностей, применять стандарты антикоррупционного поведения

ОК 07 Содействовать сохранению окружающей среды, Умения: соблюдать нормы экологической безопасности; определять направления ресурсосбережения в рамках профессиональной деятельности по специальности ресурсосбережению, эффективно действовать в чрезвычайных ситуациях.

ОК 08 Использовать средства физической культуры для сохранения и укрепления здоровья в процессе профессиональной деятельности и поддержания необходимого уровня физической подготовленности.

ОК 09 Использовать информационные технологии в профессиональной деятельности

ОК 10 Пользоваться профессиональной документацией на государственном и иностранном языках

ОК 11 Использовать знания по финансовой грамотности, планировать предпринимательскую деятельность в профессиональной сфере.

# ОСНОВНАЯ ЧАСТЬ

## **Анализ предметной области**

## **Общая постановка задач**

Целью разработки является создание программного средства, позволяющего численно оценивать площадь большей части окружности, ограниченной горизонтальной прямой, методом Монте–Карло, а также с помощью аналитического (математического) расчёта.

Приложение должно содержать в себе следующие формы:

- заставка с указанием автора и варианта работы;

- основная форма с вводом данных, численным и графическим моделированием и сохранением данных моделирования;

- форма анализа моделирования, включающая в себя графический анализ всех выполненных попыток моделирования;

- форма «О программе»;

- встроенная справка по работе с программой.

Дальше речь пойдет об уточнение поставленной задачи.

## **Уточнение постановки задачи**

IDEF0 наглядно демонстрирует логические связи между операциями. С помощью двух диаграмм IDEF0 можно проследить общую логическую структуру всех объектов. Диаграмма IDEF0 показывает ключевые действия и этапы работы системы (см. Рисунок 1 – диаграмма IDEF0). В блок «Вычисления» поступают данные о количестве точек и результаты моделирования, а на выходе формируются итоги вычислений в соответствии с математической формулой. В качестве рабочей среды используется программа, а управление процессом обеспечивает математическая модель.

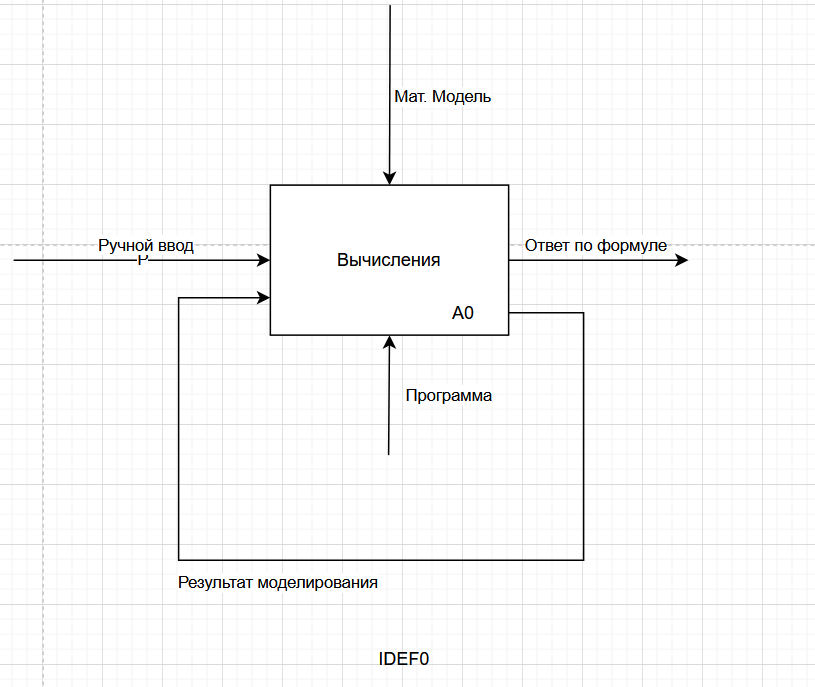


Рисунок 1 – диаграмма IDEF0

IDEF0–диаграмма с декомпозицией раскрывает внутреннюю структуру процесса, детализируя его элементы и уровни разбиения (см. Рисунок 2 – IDEF0–диаграмма с декомпозицией). В блоке «Моделирование» на вход поступают количество точек и предварительные результаты моделирования; на выходе формируется итоговый результат метода Монте–Карло и вычисления по заданной математической формуле. В качестве механизма здесь используется соответствующая программа. Блок «Анализ» принимает результаты Монте–Карло и математических вычислений, выполняет оценку и возвращает выводы анализа; в этом блоке также задействована программа в качестве механизма.

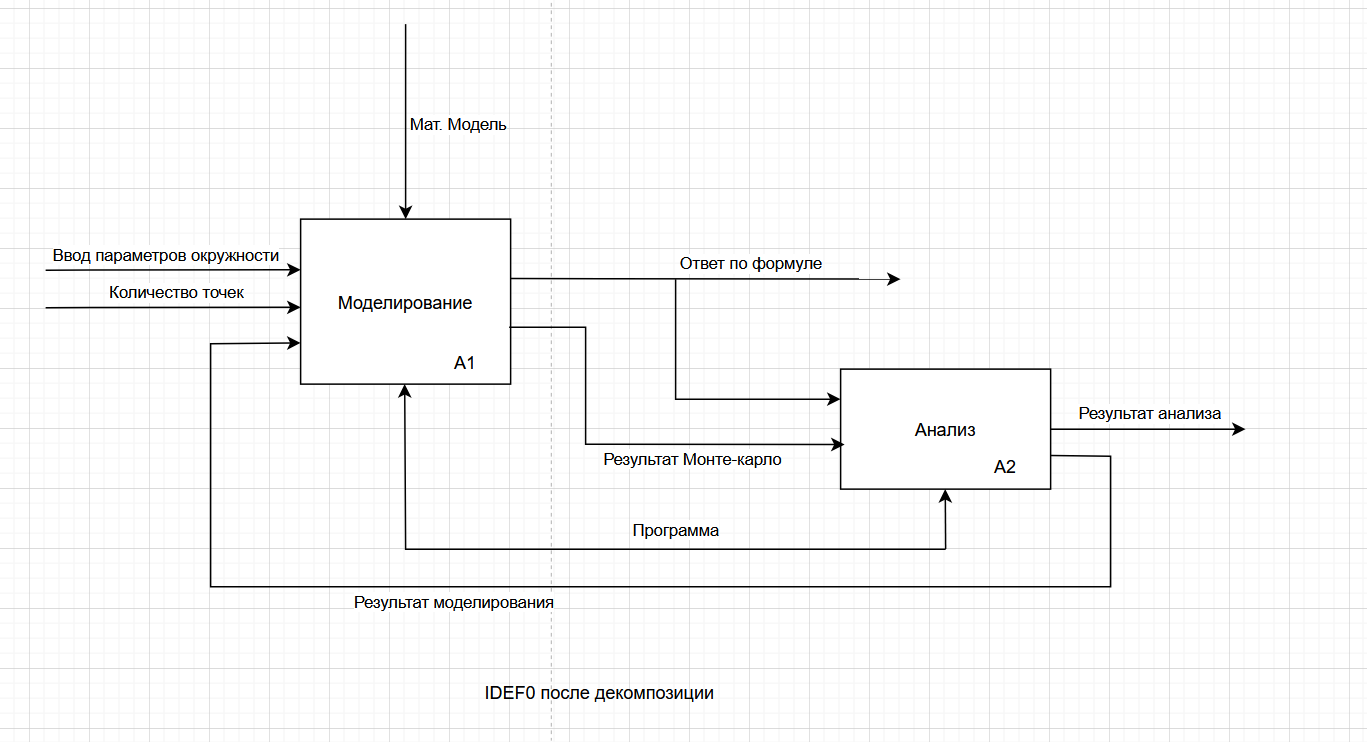


Рисунок 2 – IDEF0–диаграмма с декомпозицией

Ниже приводится описание математической модели.

## **Математическая модель**

Для определения площади большего сегмента окружности используется метод Монте-Карло, основанный на случайном генерации точек внутри заданной области и проверке, попадают ли они в нужную часть фигуры. Такой вероятностный подход позволяет оценить площадь сложной геометрической фигуры. На представленном чертеже вариант 13 с параметрами x0​=−2, y0​=1, радиусом R=2 и константой C=2 (см. Рисунок 3 – Чертеж) визуализирован сегмент, площадь которого требуется вычислить.

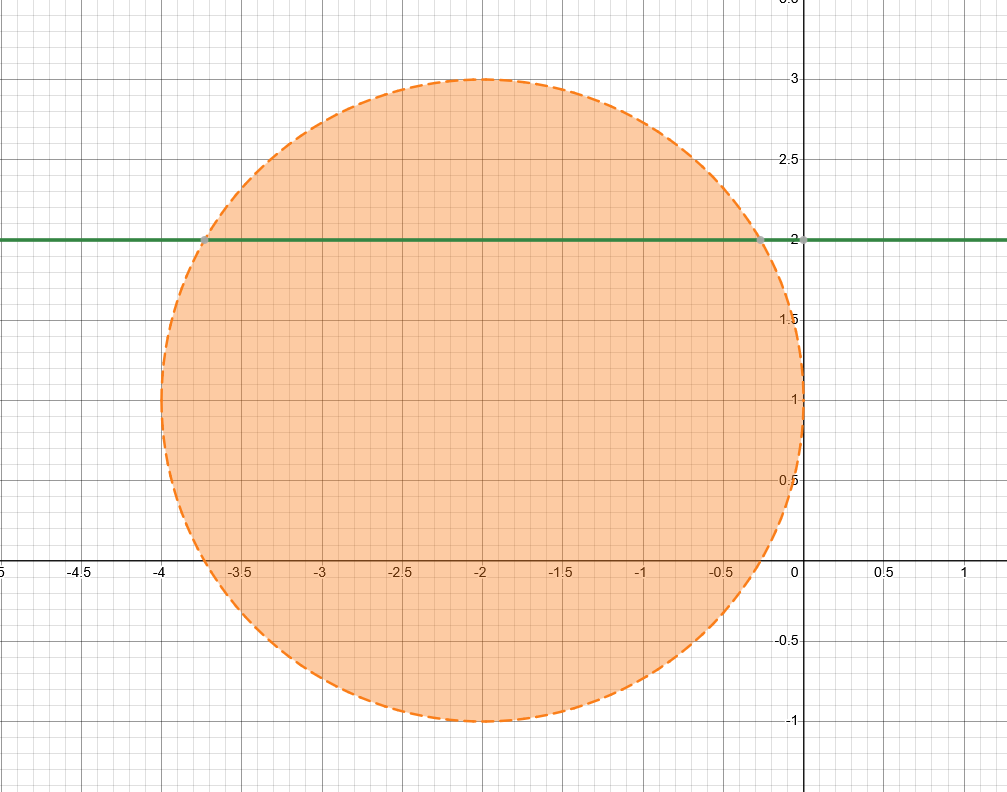


Рисунок 3 – Чертеж

Площадь сегмента фигуры может быть оценена как (1), где A – площадь области, охватывающей большой сегмент. A – описывается как – «» (2).

(3)

h–высота большего сегмента фигуры (исходя из варианта h=3).

Для нахождения площади по точкам: площадь прямоугольника \*(количество попавших точек в круг / на общее количество точек.

## **Обоснование выбора средств методов и средств**

## **Описание метода решения**

Метод Монте–Карло – это численный статистический метод, применяемый для моделирования процессов и оценки вероятностей различных исходов с помощью случайных чисел. Его основная идея заключается в использовании генерации случайных данных для исследования поведения сложных систем или объектов, а также для приближённого вычисления величин, которые трудно определить аналитически.

При оценке площади фигуры методом Монте–Карло задаётся геометрическая форма: выбирается крупный сегмент, ограниченный заданной кривой, внутри которого будет проводиться моделирование. Затем внутри прямоугольной области, охватывающей эту фигуру, равномерно генерируются случайные точки с координатами (x, y), каждая из которых проверяется на принадлежность к исходному сегменту. Точки, попавшие внутрь границ фигуры, учитываются при подсчёте, после чего отношение числа таких точек к общему числу сгенерированных умножается на площадь охватывающей области, что даёт приближённую оценку площади исследуемой формы.

Формула для оценки площади: где:

S – приближённая площадь фигуры,

A – площадь прямоугольной области, в которой генерировались точки,

M – число точек, попавших внутрь фигуры,

N – общее число сгенерированных точек.

Таким образом, метод Монте–Карло позволяет статистически оценить площадь сложной геометрической формы, опираясь на принцип вероятностного моделирования.

## **Анализ средств разработки**

При разрабатывании АС будет использован язык программирования – C#, среда разработки АС–Microsoft Visual Studio и способ хранения данных – Excel. Что бы убедиться, что выбранный язык программирования, среда разработки и способ хранения – подходящие, необходимо провести анализ с определением преимуществ и недостатков.

Язык программирования (ЯП) C#. Основным преимуществом C# является обладание популярностью и обширным сообществом разработчиков. C# предоставляет современные средства разработки и интеграции с платформой .NET. Язык подходит для разработки интерфейсов и интеграцией с разными источниками хранения данных. Недостатками являются ограничения в использовании на других платформах, за исключением платформы .NET.

Среда разработки Microsoft Visual Studio. Преимущества среды в том, что она предлагает инструменты для разработки, отладки и тестирования программного обеспечения. Предоставляет обширный набор функций, интегрированных визуальных дизайнеров и управление проектами. Недостатки недоступен для разработки на некоторых платформах, требует установки и занимает значительное место на жестком диске.

Способ хранения данных Excel. Преимущества данного вида хранения обусловлено тем, что он предлагает удобный формат для хранения структурированных данных. Excel поддерживается большинством средств разработки, в том числе C# и Visual Studio. Excel удобен для обмена данными между системами. Недостатки может занимать больше места по сравнению с бинарными форматами; Excel обработка потребляет больше ресурсов.

Так как АС не планируется использовать на других платформах, язык программирования C# подходит для нее. По такой же причине выбранная среда Microsoft Visual Studio.

Excel выбран в работе в качестве способа хранения данных так как, облегчает хранение структурированных данных в формате, понятном человеку.

## **Проектирование приложения**

## **3.1 UML–диаграммы**

UML помогает разработчикам и аналитикам наглядно отображать структуру, поведение и взаимодействие компонентов, упрощая понимание сложных систем.

**3.1.1 Диаграмма прецедентов**

UML–диаграмма прецедентов показывает, как пользователь взаимодействует с системой, и является частью концептуальной модели прецедентов. На диаграмме актором выступает Пользователь, получающий доступ к ключевым функциям: вводу данных, сохранению данных, расчёту, просмотру статистики, настройкам, справке и информации о программе. Взаимосвязи между прецедентами раскрываются через включения и расширения: так, прецедент «Расчёт» включает «Просмотр результатов», который может быть расширен за счёт численного и графического моделирования; «Просмотр статистики» и «Ввод данных» могут дополняться соответствующими графическими представлениями и вводом числовых значений; при сохранении данных возможно включение функции экспорта в Excel. В результате диаграмма наглядно демонстрирует функциональные возможности системы и логические связи между действиями пользователя (см. Рисунок 4 – UML–диаграмма прецедентов).

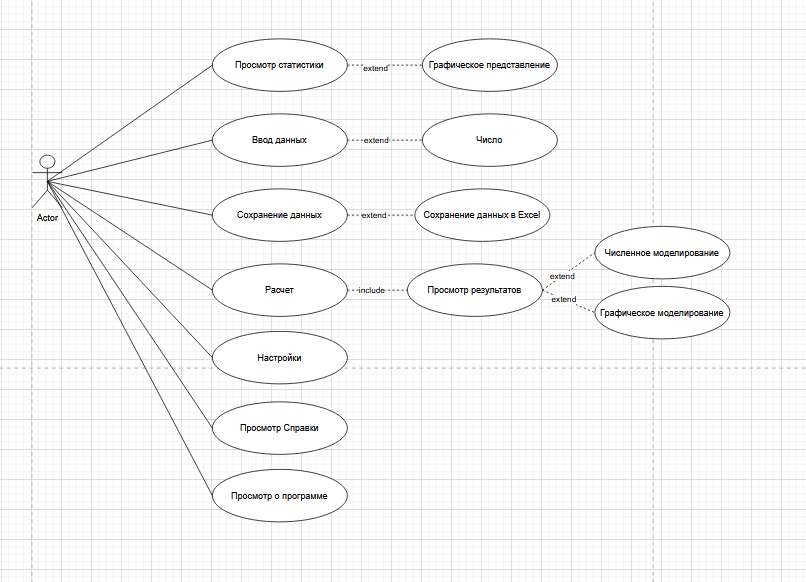


Рисунок 4 – UML–диаграмма прецедентов

Далее рассматривается диаграмма деятельности.

* + 1. **Диаграмма деятельности**

UML-диаграмма деятельности демонстрирует последовательность действий в системе, описывая её логику и конкретные шаги. На такой диаграмме представлены действия, включая «Заставка», «Справка», «Анализ» и «О программе», а также показаны их взаимосвязи и порядок выполнения (см. Рисунок 6 – UML-диаграммы деятельности).

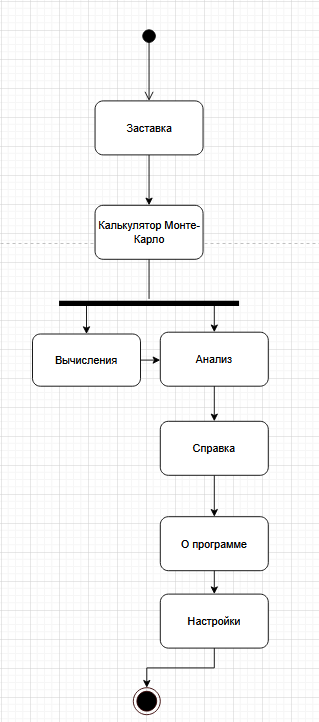


Рисунок 6 – UML-диаграммы деятельности

Далее рассматривается диаграмма последовательности.

* + 1. **Диаграмма последовательности**

UML-диаграмма последовательностей представляет собой диаграмму, отображающую взаимодействие объектов в системе на основе последовательности событий во времени. Является важной частью моделирования системы, позволяя представить последовательность действий и коммуникаций между объектами. В UML-диаграмме последовательностей, актер «Пользователь» взаимодействует с прецедентами «Заставка», «Справка», «Калькулятор Монте–Карло», «Анализ» и «О программе» в конкретной последовательности действий (см. Рисунок 5).

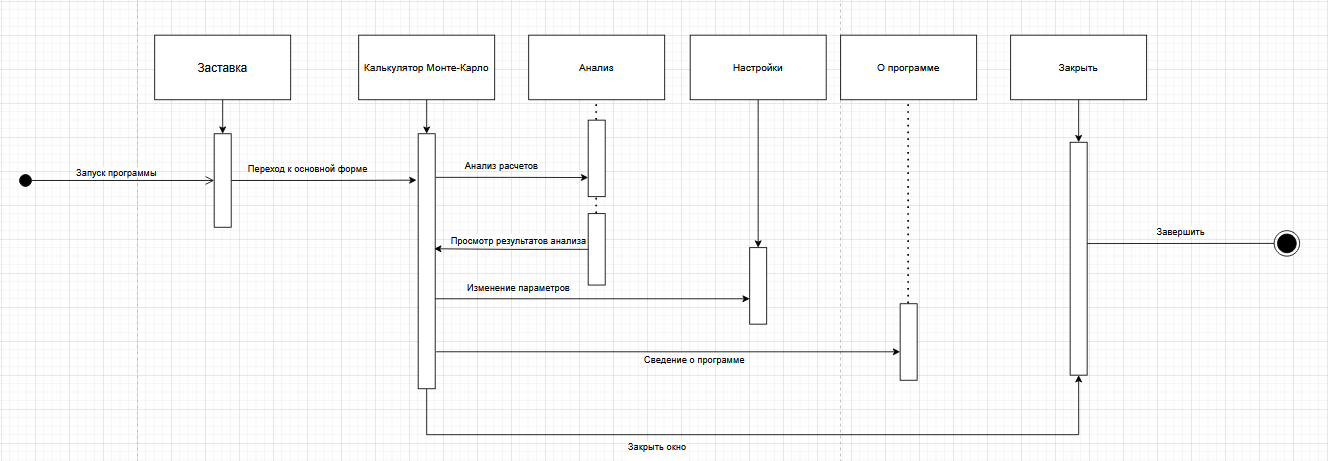


Рисунок 5 – UML-диаграмма последовательность

Эта диаграмма помогает понять, как объекты взаимодействуют во время процесса.

## **3.2. Wireframe-эскизы приложения**

Разработка программы подразумевает однооконное приложение с 3 страницами, такими как: «Расчет», «Анализ» и «Настройки». Каждый Wireframe-эскиз помогает визуализировать модули и страницы до начала разработка АС, что позволяет заранее понять, как будет выглядеть приложение и при разработке поможет упростить процесс. «Заставка» является первым что видит пользователь (Рисунок 7 – Заставка).



Рисунок 7 – «Заставка» выполнена в Figma

Страница «Расчет» отвечает за графическую часть метода Монте-Карло, она показывает заполнение фигуры точками. Пользователь может ввести количество точек. На странице есть кнопка: «Рассчитать» (Рисунок 8 – Расчет).

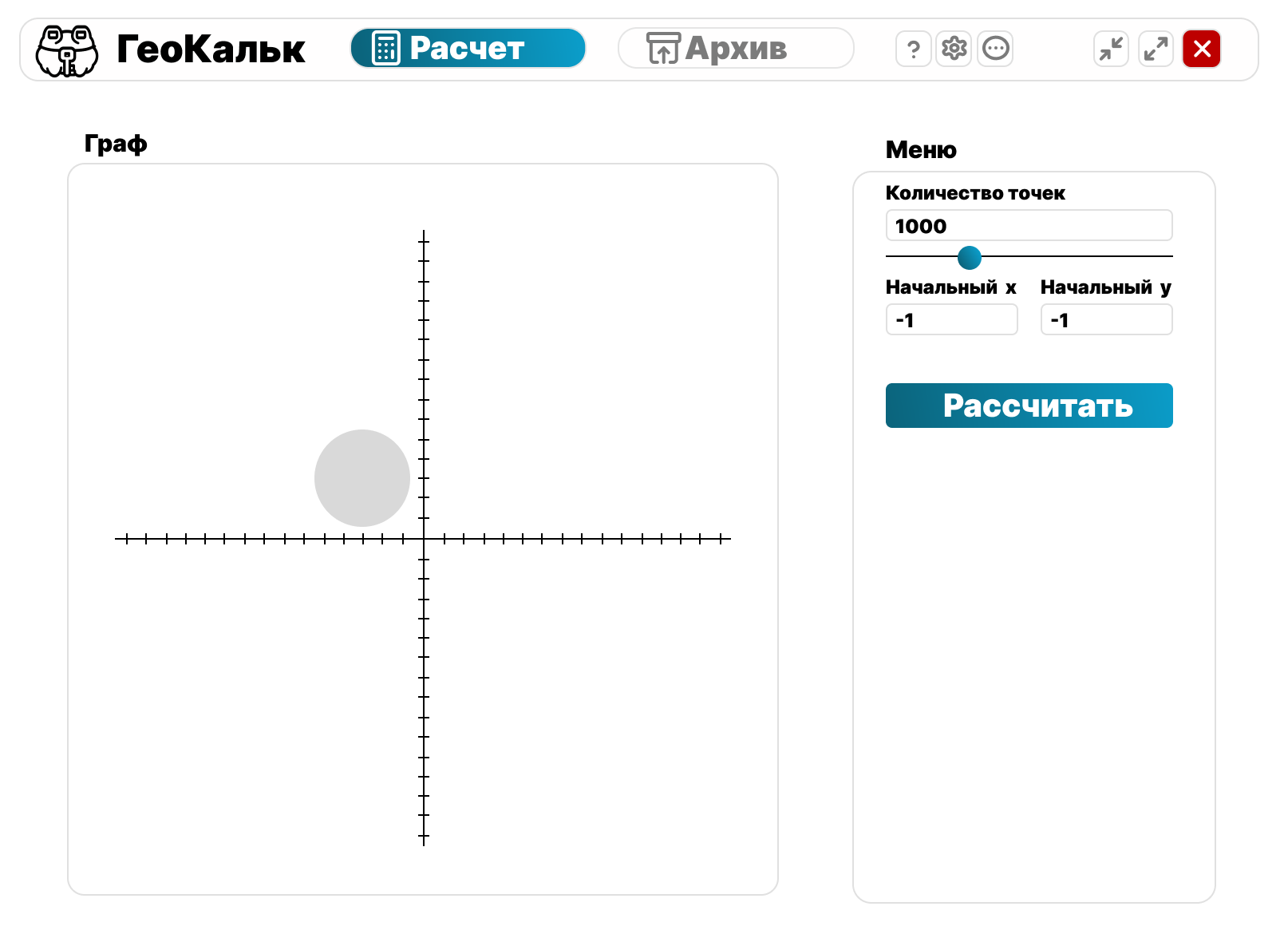


Рисунок 8 – «Расчет» выполнен в Figma

Страница «Анализ» показывает диаграмму результатов вычисления площади (Рисунок 9 – «Анализ» выполнен в Figma).

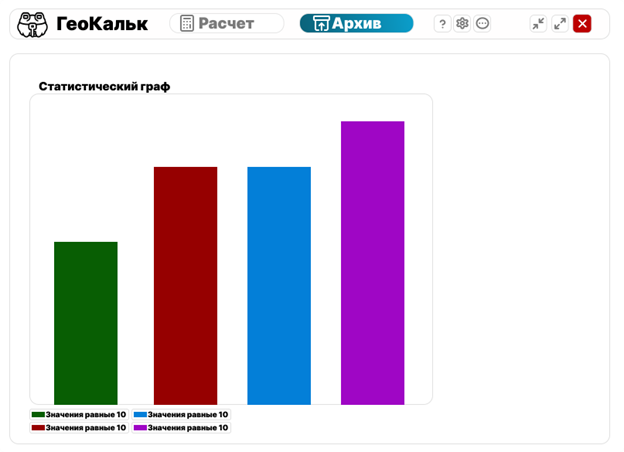


Рисунок 9 – «Анализ» выполнен в Figma

Кнопка «Настройки» выполненная в Figma открывает страницу с параметрами программы (Рисунок 10 – Настройки).

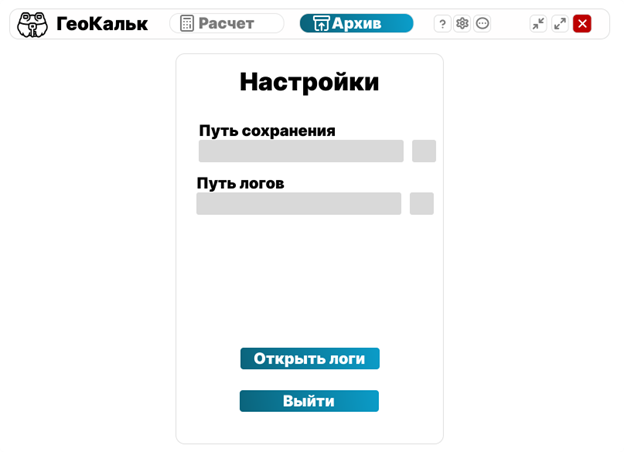


Рисунок 10 – «Настройки» выполнены в Figma

Кнопка «О программе» открывает окно с названием программы, автором и версией программы (Рисунок 11 – «О программе» выполнено в Figma).

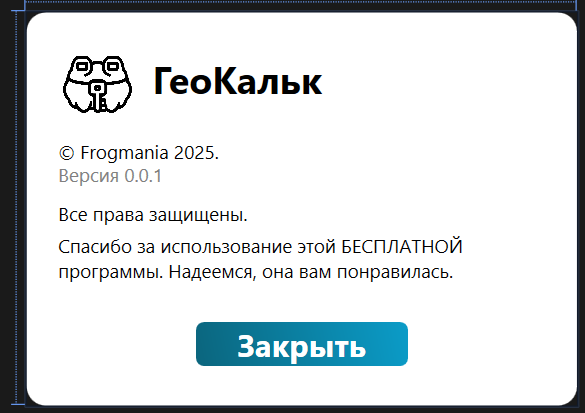


Рисунок 11 – «О программе» выполнено в Figma

Далее рассмотрим выбранный формат данных для сохранения.

## **3.3 Описание выбранных форматов данных**

XLSX – это формат файлов электронных таблиц, используемый в Microsoft Excel начиная с версии 2007. Файлы XLSX содержат данные в виде таблиц, включая ячейки с текстом, числами, формулами и датами. Использование файлов формата XLSX для хранения данных удобно благодаря широкой поддержке офисными приложениями и библиотеками, позволяющим работать с ними на разных платформах. Кроме того, XLSX хорошо подходит для экспорта отчётов, особенно если данные должны быть представлены в виде таблиц с формулами и стилями. Однако у формата есть ограничения: он не предназначен для хранения больших объёмов данных или работы в многопользовательском режиме, что может привести к проблемам с производительностью и согласованностью при параллельном доступе. Также изменение структуры файла вручную может привести к ошибкам, а отсутствие строгой схемы данных делает формат менее надёжным для программной обработки по сравнению с базами данных или форматом JSON/XML. В приложение XLSX будет использоваться для хранения данных о вычислениях (см. Рисунок 12 – пример XLSX файла).

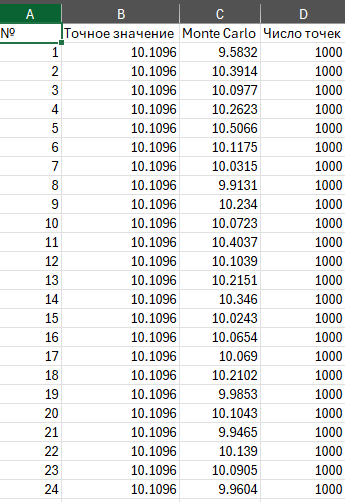


Рисунок 12 – пример XLSX файла

Табличная структура делает такие файлы наглядными для пользователя, а возможность форматирования и сортировки облегчает визуальный анализ данных.

## **Разработка алгоритмов**

Для разработки алгоритмов были использованы блок-схемы, отражающие основные этапы метода Монте-Карло, применяемого для нахождения большего сегмента фигуры, и формулы площади сегмента окружности, ограниченного горизонтальной прямой. Каждая блок схема отражает отдельные алгоритмы программы. Блок-схемы представлены в целях изложения методики и процесса алгоритма, использованного в модулях. Для демонстрации вычисления площади большего сегмента окружности методом Монте-Карло была создана блок–схема функции (Рисунок 13 – Блок-схема функции SimulateWithPointsWithAccuracyAsync). Блок-схема отражает работу функции в программе, при вызове функции передаются значения, после чего вычисляются границы прямоугольника для дальнейшего генерирования точек на графике. Следующим действием является вычисление площади и оценка площади сегмента

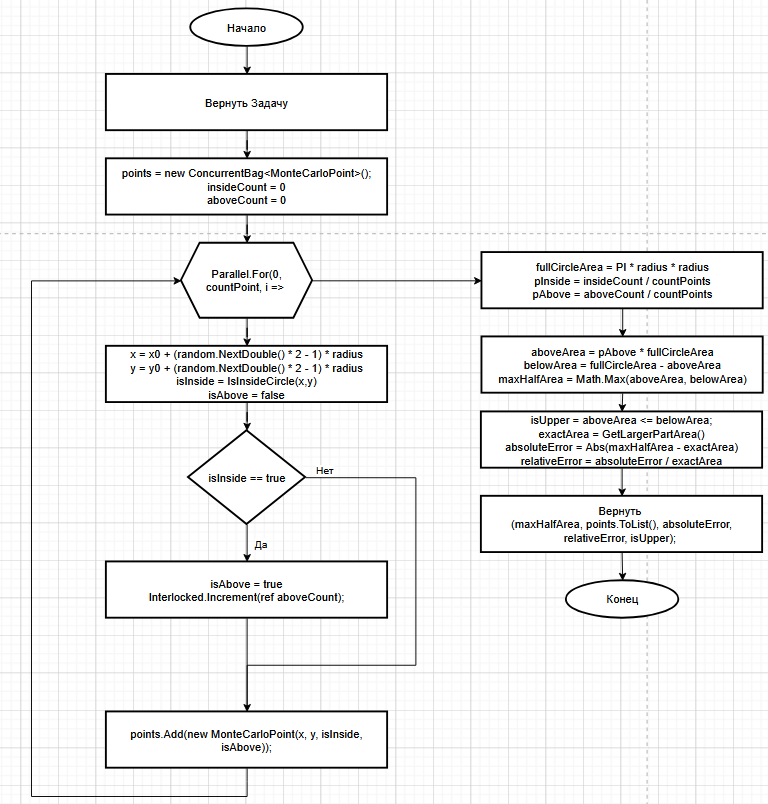


Рисунок 13 – Блок-схема функции SimulateWithPointsWithAccuracyAsync

Для проверки результата вычисления площади большего сегмента окружности методом Монте-Карло, используется формула площади сегмента окружности, ограниченного горизонтальной прямой, для наглядного изображения алгоритма используется блок–схема (Рисунок 14 – Блок-схема метода GetLargerPartArea).

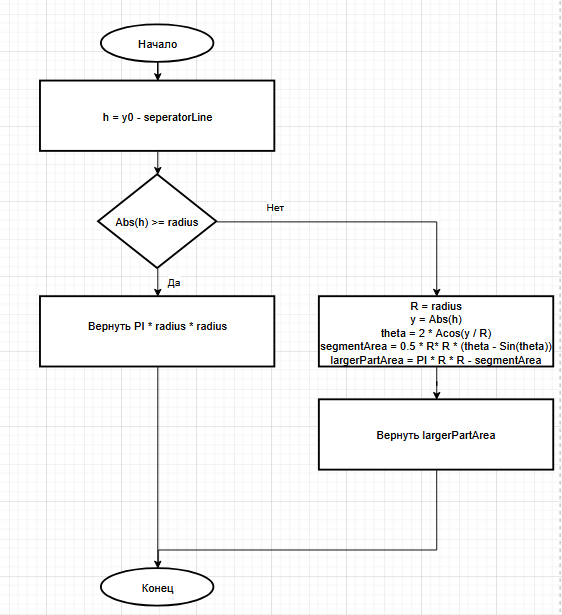


Рисунок 14 – Блок–схема метода GetLargerPartArea

Блок-схема отражает математический подход вычисления площади большего сегмента окружности с проверкой, что диаметр больше радиуса, чтобы не возникло отрицательного результата.

Ниже показана блок-схема, которая описывает работу метода OnCalculationButtonPressed отвечающего за запуск расчетов в приложении (см. Рисунок 15 – Блок-схема метода OnCalculationButtonPressed).

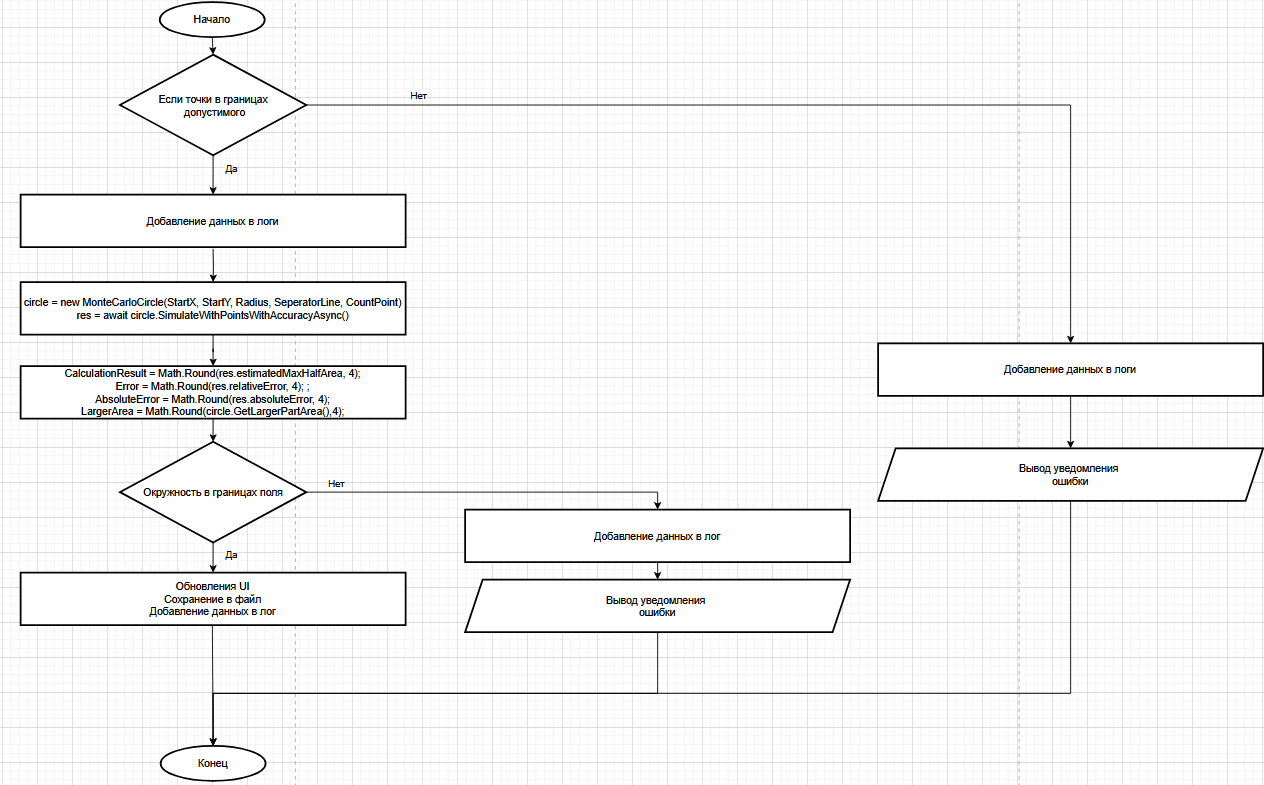


Рисунок 15 – Блок-схема метода OnCalculationButtonPressed

Блок-схема отражает логику работы метода, отвечающего за расчет и отображения результатов расчета.

## **Разработка приложения**

Разработка приложения подразумевает разработку одного окна с тремя страницами: Расчет, Анализ, Настройки.

Окно «Заставка» показывает имя разработчика и вариант (см. Рисунок 16 – Заставка).

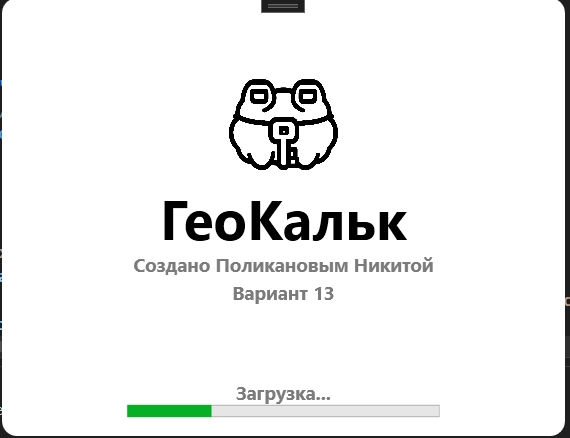


Рисунок 16 – Заставка

Страница «Расчет» отвечает за графическую часть метода Монте-Карло, она показывает заполнение фигуры точками. Пользователь может ввести количество точек, координаты центра окружности, радиус, удаление прямой от оси. На странице есть кнопка «Рассчитать». Также после нахождения площади можно получить результаты вычислений (см. Рисунок 17 – Расчет).

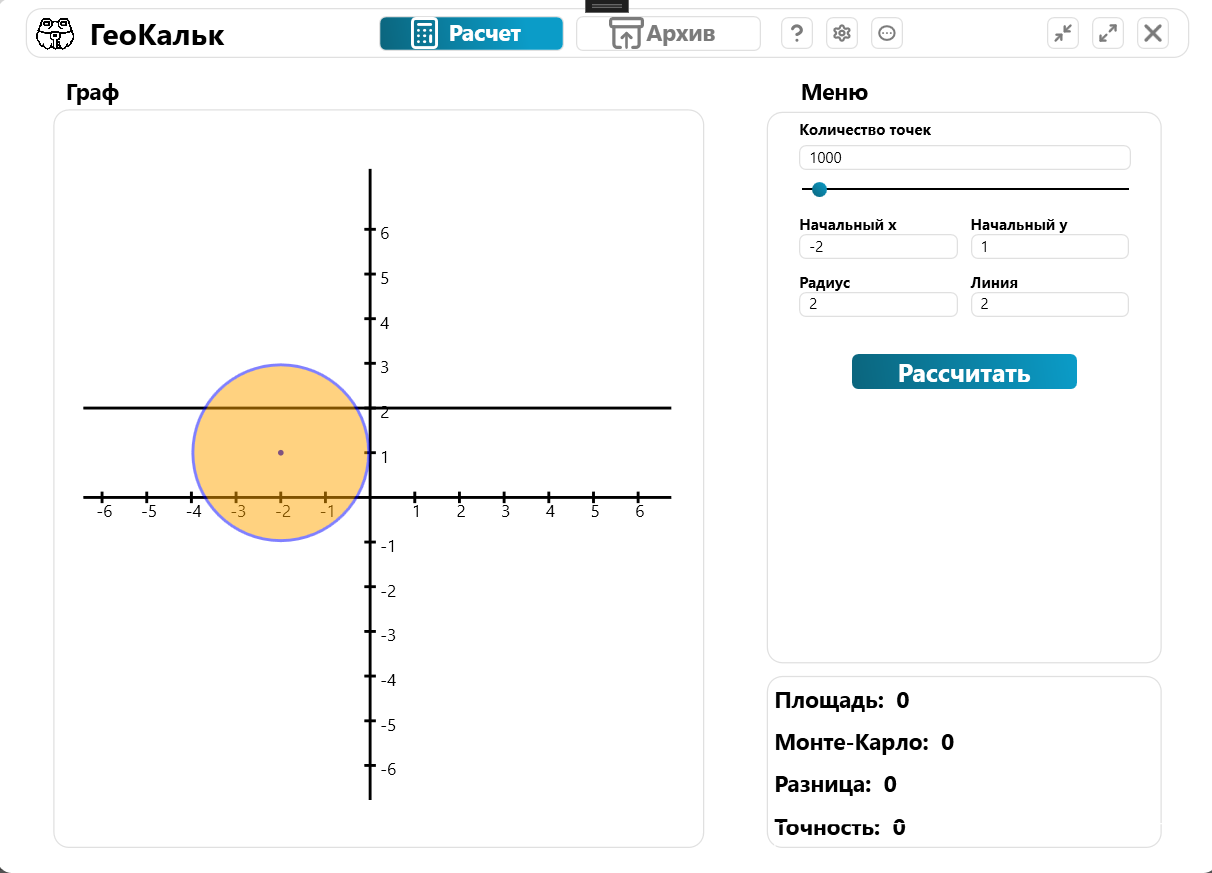


Рисунок 17 – Расчет

Страница «Анализ» показывает диаграмму результатов вычисления площади (Рисунок 17 – Анализ).

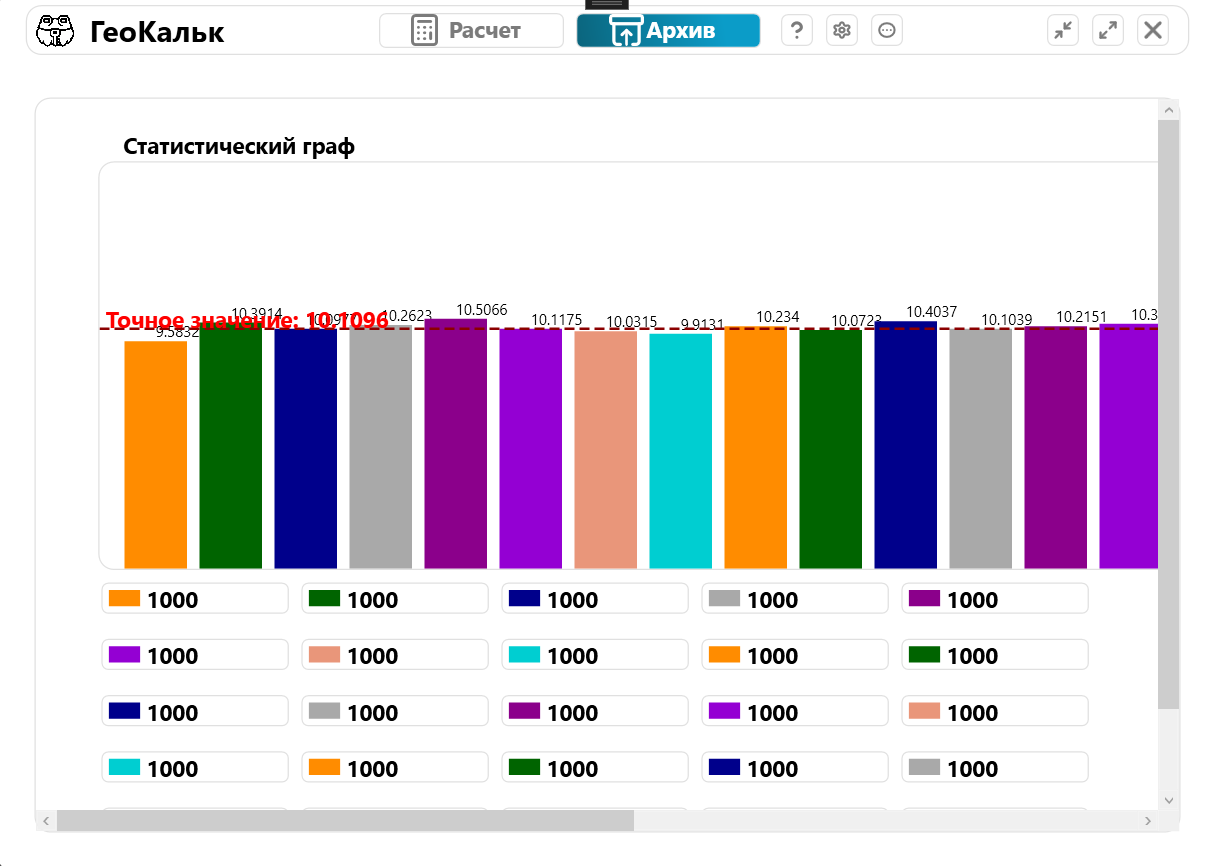


Рисунок 17 – Анализ

В верхний части приложения есть меню управления, в котором можно открыть «Справку», «Настройки», «О программе».

Окно «О программе» показывает название программы, ее разработчика и версию программы (Рисунок 18 – О программе).

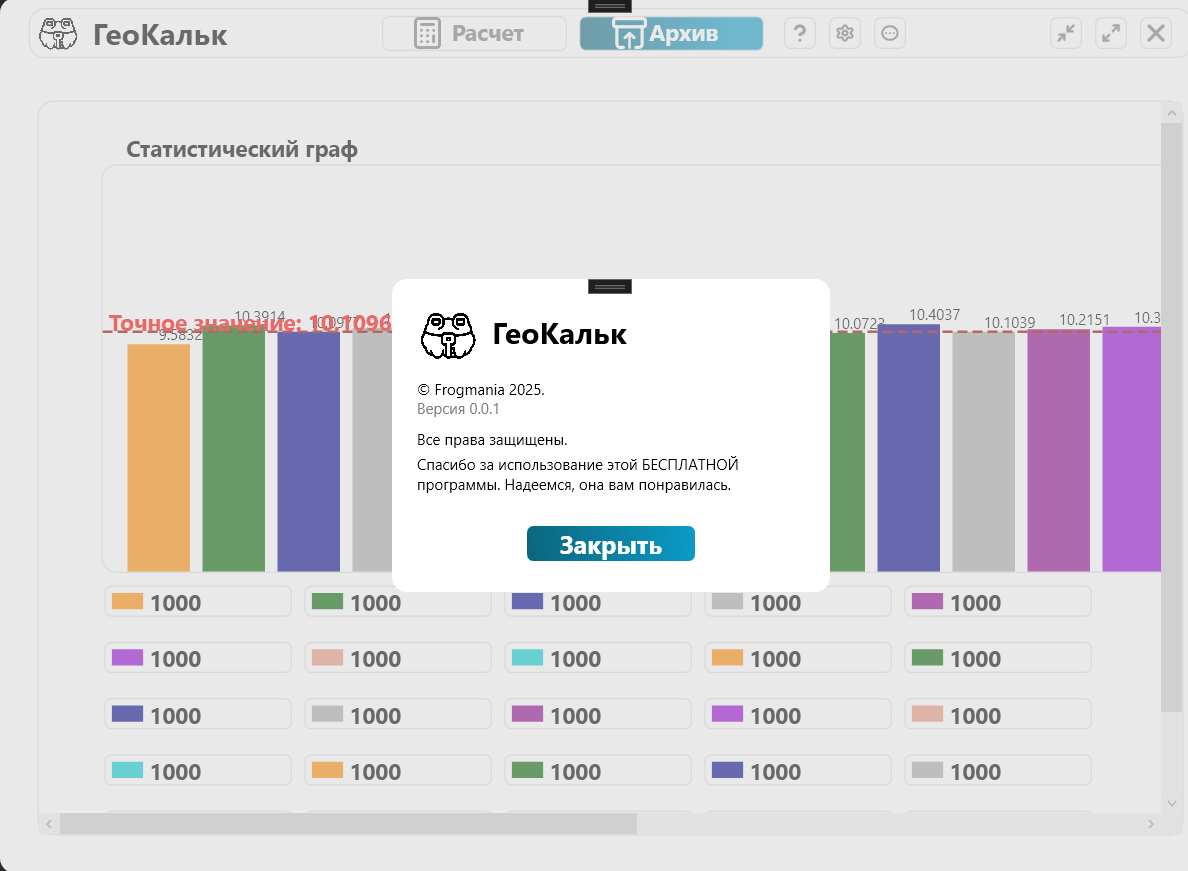


Рисунок 18 – О программе

Окно «Справки» отображает справку по эксплуатации программы (см. Рисунок 19 – Справка).

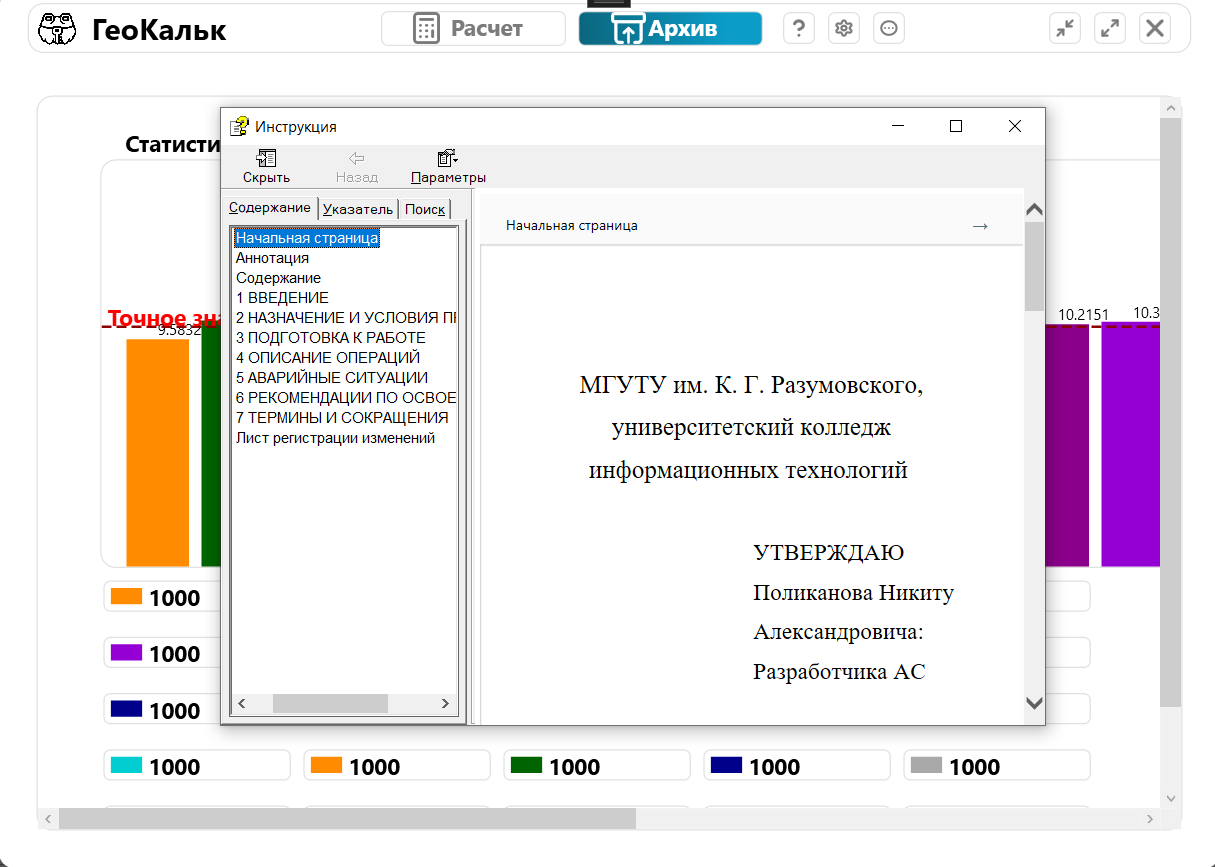


Рисунок 19 – Справка

Страница «Настройки» позволяет пользователю выбрать куда сохранять результаты вычислений и путь до легирования системы (см. Рисунок 20 – Настройки).

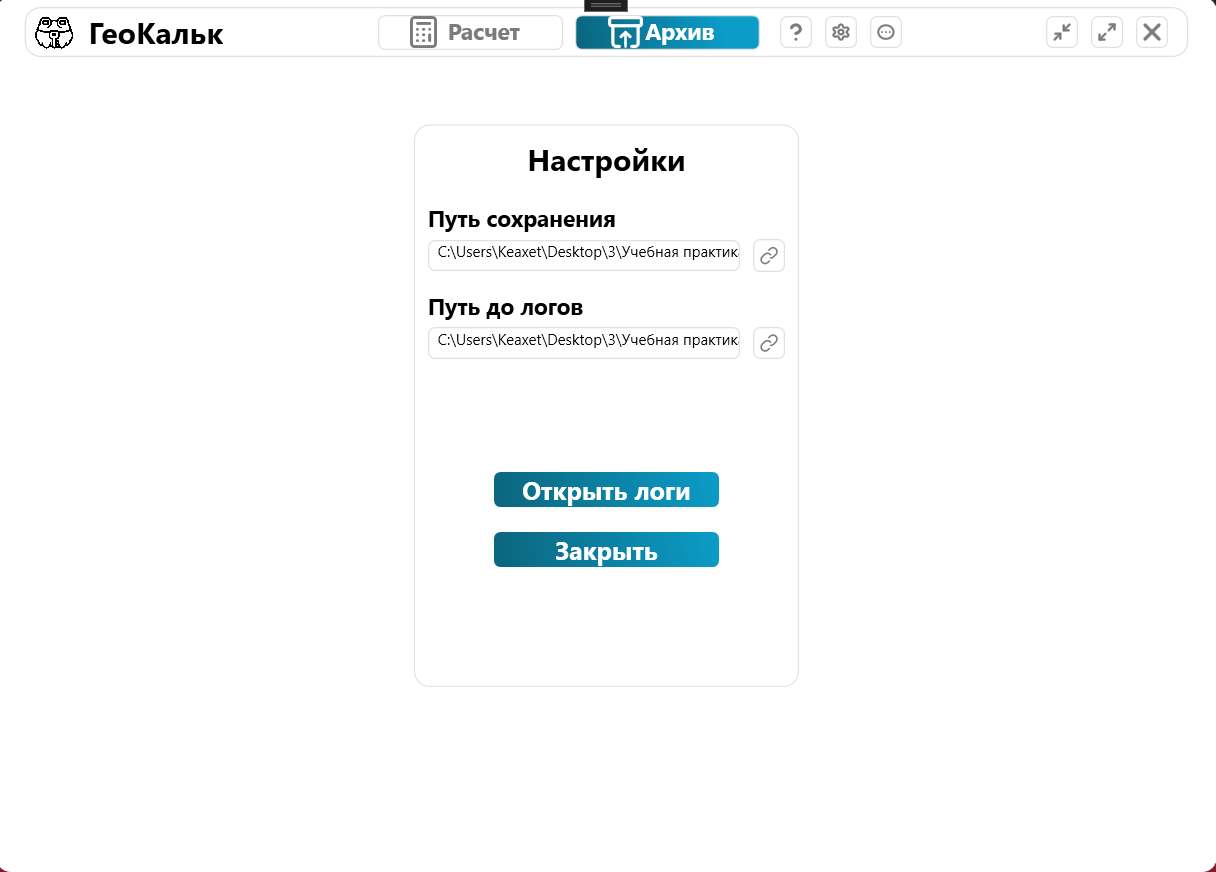


Рисунок 20 – Настройки

Далее рассматриваются результаты проведенного тестирования.

## **Тестирование приложения**

Для тестирования приложения был выбран метод «Белый ящик». Данный метод позволяет оценить работу с учетом ее внутреннего устройства, он полезен для корректной оценки эффективности и правильности алгоритмов. Причиной выбора метода тестирования является знание своего кода в программе. Тестирование производится на случаи заполнения поля количества точек моделирования и кнопок, таких как: «Рассчитать», «Открыть логи», «Анализ» (Таблица 1 – Тестирование).

Таблица 1 – Тестирование

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| №  Тестового случая | Входные данные | Выходные данные | Фактический результат | Скриншоты результата |
| 1 | Кол. Точек – 1000  X0 = –2,  Y0 = 1  C = 2,  R = 2 | Площадь – 10.1096  Монте–Карло – 10.1466  Разница – 0.0369  Точность – 0.0037 | Соответствует ожидаемому |  |
| 2 | Кол. Точек – 20 000  X0=–2,  Y0 = 1,  C = 2,  R = 2 | Площадь 10.1096  Монте–Карло – 10.0959  Точность 0.0014 | Соответствует ожидаемому |  |
| 3 | опакяпо | Запрет ввода | Запрет ввода |  |
| 4 | Проверка сохранения данных | Данные сохранены в правильном порядке | Соответствует ожидаемому |  |
| 5 | Сохранение в файл | Данные сохраняются в файл | Соответствует ожидаемому |  |

Также было проведено автоматическое тестирование методов на корректность расчетов результаты тестов приведено в таблице 2.

Таблица 2

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Номер теста | Название теста | Входные данные | Ожидаемый результат | Результат теста |
| 1 | Тестирование метода нахождения площади методом Монте–Карло | X0 = –2  Y0 = 1  R = 2  C = 2 | Точность не больше 0.5 |  |
| 2 | Тестирование метода нахождения мат площади большего сегмента | X0 = –2  Y0 = 1  R = 2  C = 2  Кол. Точек = 1000 | 10.1096 |  |
| 3 | Тестирование метода нахождения площади большего сегмента | X0 = –1  Y0 = 2  R = 3  C = 4  Кол. Точек = 1000 | 25.1769 |  |

Ниже приведен анализ качества моделирования.

## **Анализ качества моделирования**

Результатом разработки является полноценная автоматизированная система (АС), состоящая из нескольких модулей, предназначенных для моделирования и анализа. Основная цель системы – визуализация метода работы и демонстрация процесса анализа, основанного на вычислении площади большого сегмента, ограниченного окружностью с радиусом R и центром в точке (x0, y0), а также горизонтальной прямой, проходящей на фиксированном расстоянии C от соответствующей оси координат. Расчёт осуществляется с использованием математической формулы и метода Монте-Карло. При выполнении основной задачи система допускает незначительную погрешность, обусловленную случайным характером генерации точек внутри области. Точность результата зависит от количества сгенерированных точек: при увеличении их числа погрешность снижается и варьируется в пределах от 0,5 до 0, 0134

# ЗАКЛЮЧЕНИЕ

В ходе прохождения учебной практики по профессиональному модулю ПМ.02 Осуществление интеграции программных модулей в период с 09.06.2025 по 28.06.2025 в Университетском колледже информационных технологий ФГБОУ ВО «МГУТУ им. К.Г. Разумовского (ПКУ)» в Лаборатории программного обеспечения и сопровождения компьютерных систем, освоены необходимые практические умения и закреплены навыки работы по данному модулю.

Целью практики являлось развитие практических навыков, умений и опыта, а также формирование компетенций в процессе выполнения определенных практических задач, связанных с будущей профессиональной деятельностью. Выполнены задачи учебной практики: получен опыт выполнения работ по интеграции модулей программного обеспечения для компьютерных систем, освоены общие компетенции и профессиональные компетенции.

В ходе работы было выполнено:

- изучение математических основ, необходимых для расчёта площади большого сегмента фигуры;

- разработка программного модуля, реализующего метод Монте–Карло для численного вычисления площади;

- моделирование и реализация графического интерфейса;

- создание интуитивно понятного интерфейса для взаимодействия с пользователем;

- разработка экранных форм: заставки, основной формы для ввода параметров, модуля численного и графического моделирования, отображения результатов, анализа и справочной информации;

- составление технического задания на основе задания, полученного от преподавателя;

- определение требований, необходимых для реализации поставленной задачи;

- добавление встроенной справочной системы для поддержки пользователей при работе с программой.

Таким образом, были закреплены практические навыки и выполнены все поставленные цели и задачи.

# СПИСОК ЛИТЕРАТУРЫ

1. Интегрирование методом Монте–Карло // Stack Overflow на русском. – Режим доступа: [https://ru.stackoverflow.com/questions/1053960/Интегрирование–методом–Монте–Карло](https://ru.stackoverflow.com/questions/1053960/Интегрирование-методом-Монте-Карло) (Дата обращения: 15.06.2025).
2. Метод Монте–Карло для построения модели прохождения лазерного излучения через запыленную среду // ТУСУР, 2018. – Режим доступа: https://storage.tusur.ru/files/115518/2018\_3.pdf (Дата обращения: 09.06.2025).
3. Современные математические методы и компьютерные технологии в проектировании... // Гомель, 2022. – Режим доступа: <https://elib.gsu.by/bitstream/.../Новые_математические_методы_2022.pdf> (Дата обращения: 10.06.2025).
4. Современные технологии в теории и практике программирования // СПбГПУ, 2024. – Режим доступа: <https://hsse.spbstu.ru/userfiles/files/1941_sovremennie_tehnologii_s_oblozhkoy.pdf> (Дата обращения: 09.06.2025).
5. Методические указания по организации самостоятельной работы студентов // Инфоурок, 2021. – Режим доступа: [https://infourok.ru/metodicheskie–ukazaniya–po–organizatsii–samostoyatelnoj–raboty–studentov–5421119.html](https://infourok.ru/metodicheskie-ukazaniya-po-organizatsii-samostoyatelnoj-raboty-studentov-5421119.html) (Дата обращения: 14.06.2025).
6. Новые информационные технологии в научных исследованиях // БГУИР, 2021. – Режим доступа: <https://www.bsuir.by/m/12_100229_1_158717.pdf> (Дата обращения: 15.06.2025).
7. Троелсен Э. Язык программирования C# 5.0 и платформа .NET 4.5 // Вильямс, 2015 (Дата обращения: 15.06.2025).
8. Абрамян М. Visual C# на примерах // БХВ‑Петербург, 2016 (Дата обращения: 16.06.2025).
9. Доклад из конференции ИТ‑систем 2022 // БГУИР, 2022. – Режим доступа: <https://its.bsuir.by/m/12_130111_1_168455.pdf> (Дата обращения: 16.06.2025).
10. Джентльменский набор для создания WPF‑приложений // Хабр. – Режим доступа: <https://habr.com/ru/articles/647259/> (Дата обращения: 17.06.2025).

# ПРИЛОЖЕНИЕ А

# Техническое задание

**Общие сведения**

Наименование: ГеоКальк.

Условное обозначение: ГК.

Наименование организации: Университетский колледж информационных технологий им. Разумовского.

Разработчик: Поликанов Никита Александрович

Перечь документов, на основании которых создается АС утверждено руководителем практики 09.06.2025.

Плановые сроки начала и окончания работ по созданию АС

Начало работ: 09.06.2025

Окончание работ: 29.06.2025

Работа выполняется в рамках учебной практики. Финансирование не предусмотрено.

**Цели создание АС:**

- разработать программный продукт для численного моделирования методом Монте–Карло;

- автоматизировать вычисления площади сегмента окружности;

- визуализировать процесс моделирования;

- освоить практический навык программирования.

Критерии оценки достижения целей создания АС

- корректность численных расчетов (погрешность < 5% при количестве точек > 10 000);

- отображать графики, соответствующих результатам моделирования;

- удобство пользовательского интерфейса.

**Назначение АС**

Программа предназначена для поддержки учебного процесса, демонстрации метода Монте-Карло и автоматизации расчетов в учебных и исследовательских целях.

**Характеристика объектов автоматизации**

Объект автоматизации – процесс численного моделирования площади сегментов окружности.

**Характеристика окружающей среды:**

- эксплуатация в стандартных офисных и учебных помещениях.

**4. Требования к автоматизированной системе**

4.1 Требования к структуре АС в целом

Структура программы:

- заставка;

- основная форма;

- форма анализа моделирования;

- форма «О программе»;

- встроенная справка.

4.1.1 Компоненты взаимодействуют через общий модуль данных.

4.1.2 Требования к характеристикам взаимосвязей создаваемой АС со смежными АС не предусмотрено.

4.1.3 Требование к режимам функционирования АС:

- интерактивный режим с визуализацией;

- автоматический режим с сохранением результатов.

4.1.4. Требования по диагностированию АС

Обработка исключительных ситуаций:

- все исключения должны быть перехвачены. Отображение стеков трассировки пользователю недопустимо – следует показывать дружелюбные сообщения;

- каждое критическое событие должно фиксироваться в отдельном журнале («error.log») с указанием причины и времени возникновения.

4.1.5 Перспективы развития, модернизации АС:

- добавление поддержки других геометрических фигур;

- возможность задания произвольной прямой;

- многоязычная поддержка интерфейса.

**4.2 Требования к функциям (задачам), выполняемым АС**

4.2.1 Перечень функций:

- прием и проверка исходных данных;

- численное моделирование методом Монте–Карло;

- графическое отображение процесса моделирования;

- аналитический расчет;

- сохранение результатов;

- проведение серии моделирований для анализа погрешности.

4.2.2 Результаты выполнения функций:

- вывод значения площади сегмента;

- визуализация результатов;

- сохраненные файлы результатов.

**4.3 Требования к видам обеспечения АС**

4.3.1 Требования к математическому обеспечению АС

Использование метода Монте-Карло и математические формулы для оценки площади геометрического сегмента.

4.3.2 Требования к информационному обеспечению:

- ввод исходных данных пользователем;

- хранение и сохранение результатов в формате Excel таблицы;

- отображение промежуточных и итоговых результатов.

4.3.3 Требования к лингвистическому обеспечению

Интерфейс – русский язык.

4.3.4 Требования к программному обеспечению

Язык программирования – C#

Библиотеки:

- Math;

- EPPlus.

4.3.5 Требования к техническому обеспечению:

- ПК с процессором не ниже Intel Core i3;

- ОЗУ не менее 512 МБ;

- Клавиатура;

- Мышь;

- Монитор 13 дюймов с разрешением не менее 1920x1080;

- Место на диске 4 ГБ.

4.3.6 Требования к метрологическому обеспечению

Оценка погрешности < 5%.

4.3.7 Требования к организационному обеспечению не требуются.

4.3.8 Использовать методические материалы по практике дисциплины:

- ГОСТ 34.602–2020;

- ГОСТ 19.701–90;

- ГОСТ Р 7.0.80–2023;

- ГОСТ 7.32–2017;

- ГОСТ Р 7.05.–2008.

**4.4 Общие технические требования АС**

4.4.1 Требования к численности и квалификации персонала и пользователя АС

Пользователь – студент или преподаватель. Навыки: базовые знания C# и математической теории вероятности.

4.4.2 Требования к показателям назначения:

- автоматизация расчета

- демонстрация метода Монте–Карло.

4.4.3 Требования к надежности:

- устойчивость к ошибкам;

- обработка недопустимых значений.

4.4.4 Требования по безопасности не требуют специальных мер.

4.4.5 Требования к эргономике и технической эстетике

- простой и понятный интерфейс;

- предупреждение для эпилептиков если есть моргающие элементы;

- графическая визуализация результатов.

4.4.6 Требования к транспортабельности для подвижных АСне применимо.

4.4.7 Требования к эксплуатации, техническому обслуживанию, ремонту и хранению компонентов АС не требуются.

4.4.9 Требования по сохранности информации при авариях:

- каждые 10 минут происходит автосохранение в файл.

4.4.10 Требования к защите от влияния внешних воздействий не требуются.

4.4.11 Требования к патентной чистоте и патентоспособности не требуются.

4.4.12 Требования по стандартизации и унификации

Использование стандартных и сторонних библиотек C#.

4.4.13 Дополнительные требования не предусмотрены.

5. **Состав и содержание работ по созданию автоматизированной системы**

**-** анализ требований к разработке ТЗ (09.06.2025);

- разработка ТЗ к разработке (10.06.2025);

- разработка прототипа метода Монте–Карло (11.06.2025 – 12.06.2025);

- разработка пользовательского интерфейса (13.06.2025 – 14.06.2025);

- интеграция графического анализа (15.06.2025);

- подготовка автоматических тестов (16.06.2025);

- подготовка ручных тестов (17.06.2025);

- проведение тестирования (18.06.2025 – 20.06.2025);

- подготовка пояснительной записки (21.06.2025 – 26.06.2025);

- подготовка к приему разработанной АС (27.06.2025).

6. **Порядок разработки автоматизированной системы**

Разработка АС осуществляется по следующим этапам с учётом выбранной программно–технической платформы:

6.1 Формулировка цели и задач разработки

- определяются цели автоматизации, состав и назначение задач, решаемых системой;

- анализируется предметная область;

- оценивается эффективность внедрения.

6.2. Выбор среды и средств разработки

В качестве основной платформы выбрана:

- ОС: Windows 10;

- язык программирования: C# (.NET 8);

- фреймворк: WPF (Windows Presentation Foundation);

- математическая библиотека: Math.NET;

- работа с Excel: EPPlus;

- среда разработки: Visual Studio 2022 или новее.

6.3. Разработка математического и алгоритмического обеспечения

Построение математических моделей процессов;

Использование библиотеки Math.NET для реализации алгоритмов, включая:

- статистические расчёты;

- метод Монте–Карло;

- линейную алгебру;

- интерполяцию.

- математическая модель описывается и реализуется в отдельных модулях проекта.

6.4. Разработка программного обеспечения и пользовательского интерфейса

- реализация графического интерфейса с использованием WPF;

- реализация функций экспорта/импорта Excel–файлов через EPPlus;

- использование .NET 8 для оптимизации производительности с будущими версиями Windows;

6.5. Проведение испытаний программного обеспечения

- модульное тестирование классов и алгоритмов с использованием NUnit;

- интеграционные тесты взаимодействия между модулями;

- сценарии ручного тестирования интерфейса;

- проведение нагрузочных и граничных тестов алгоритмов Монте–Карло;

- испытания соответствуют этапам: предварительные, приёмочные, опытная эксплуатация.

6.6. Оформление документации и подготовка к вводу в эксплуатацию:

- подготовка комплекта документации;

- создание установщика;

- подготовка системы к сдаче: финальное тестирование, резервное копирование, выдача документации заказчику.

**7. Порядок контроля и приемки автоматизированной системы**

**-** проверка ручным тестированием пользовательский интерфейс;

- автоматизированное тестирование математических расчетов;

- проверка корректности реагирование программы на ошибки;

- проверка графического отображения результатов;

- демонстрация работы программы на нескольких наборах исходных данных.

**8. Требования к составу и содержанию работ по подготовке объекта автоматизации к вводу автоматизированной системы в действие**

- OC Windows 10/11;

- установка .NET v8;

- установка приложения;

- проверка корректной работы программы на целевом оборудовании.

**9. Требования к документированию**

- инструкция пользователя;

- встроенная справка по работе с программой;

- форма «О программе» с указанием автора и версии;

- код программы.

**10. Источники разработки**

- задание на учебную практику;

- ГОСТ 34.602–2020;

# Приложение Б код приложения

Менеджер конфигурации

using System.Collections.Generic;

using System.IO;

namespace GeoCalc.Resources.Utilities

{

    /// <summary>

    /// Менеджер конфигурации

    /// </summary>

    public static class ConfigManager

    {

        private const string \_configPath = "./config.conf";

        private static string \_minPoint;

        private static string \_maxPoint;

        /// <summary>

        /// Получает путь к файлу сохранения

        /// </summary>

        public static string GetSavePath

        {

            get

            {

                return LoadConfig()["save\_path"];

            }

        }

        /// <summary>

        /// Получает путь к файлу лога

        /// </summary>

        public static string GetLogPath

        {

            get

            {

                return LoadConfig()["log\_path"];

            }

        }

        /// <summary>

        /// Получает максимальное количество точек

        /// </summary>

        public static int GetMaxPoint

        {

            get

            {

                return int.Parse(LoadConfig()["max\_point"]);

            }

        }

        /// <summary>

        /// Получает минимальное количество точек

        /// </summary>

        public static int GetMinPoint

        {

            get

            {

                return int.Parse(LoadConfig()["min\_point"]);

            }

        }

        /// <summary>

        /// Сохраняет конфигурацию

        /// </summary>

        /// <param name="savePath">Путь к файлу сохранения</param>

        /// <param name="logPath">Путь к файлу лога</param>

        public static void SaveConfig(string savePath, string logPath)

        {

            File.WriteAllLines(\_configPath, new string[] { $"save\_path={savePath}", $"log\_path={logPath}", $"max\_point={\_maxPoint}", $"min\_point={\_minPoint}" });

        }

        /// <summary>

        /// Загружает конфигурацию

        /// </summary>

        /// <returns>Словарь конфигурации</returns>

        private static Dictionary<string, string> LoadConfig()

        {

            if (!Directory.Exists(Path.Combine(AppContext.BaseDirectory, "Saves")))

            {

                Directory.CreateDirectory(Path.Combine(AppContext.BaseDirectory, "Saves"));

            }

            var config = new Dictionary<string, string>() {

                { "save\_path", Path.Combine(AppContext.BaseDirectory, "Saves/save.xlsx")},

                {"log\_path", Path.Combine(AppContext.BaseDirectory, "log.txt")},

                { "max\_point", "20000"},

                {"min\_point", "300" }

            };

            if (File.Exists(\_configPath))

            {

                string[] lines = File.ReadAllLines(\_configPath);

                foreach (var line in lines)

                {

                    var key = line.Split('=')[0];

                    var value = line.Split("=")[1];

                    config[key] = value;

                }

            }

            \_minPoint = config["min\_point"];

            \_maxPoint = config["max\_point"];

            return config;

        }

    }

}

Класс для сохранения результатов вычислений

using OfficeOpenXml;

using System.IO;

using System.Windows;

namespace GeoCalc.Resources.Utilities

{

    /// <summary>

    /// Класс для сохранения результатов вычислений

    /// </summary>

    public class DataSaver

    {

        /// <summary>

        /// Список результатов вычислений

        /// </summary>

        private static List<(double exactArea, double monteCarlo, double countPoints)> \_saveData = new List<(double exactArea, double monteCarlo, double countPoints)>();

        public DataSaver()

        {

            // регистрация лицензии для использования библиотеки EPPlus

            ExcelPackage.License.SetNonCommercialOrganization("COLLECTIVE");

        }

        /// <summary>

        /// Метод для сохранения результата вычисления

        /// </summary>

        /// <param name="exactArea"> Exact value of the area </param>

        /// <param name="monteCarlo"> Monte Carlo value of the area </param>

        /// <param name="countPoints"> Number of points used for Monte Carlo </param>

        public void SaveData(double exactArea, double monteCarlo, double countPoints)

        {

            \_saveData.Add((exactArea, monteCarlo, countPoints));

            CreateDoc();

        }

        /// <summary>

        /// Метод для чтения результатов вычислений

        /// </summary>

        /// <returns> List of results </returns>

        public List<(double exactArea, double monteCarlo, double countPoints)> ReadData()

        {

            var result = new List<(double exactArea, double monteCarlo, double countPoints)>();

            FileInfo fileInfo = new FileInfo(ConfigManager.GetSavePath);

            if (!fileInfo.Exists)

                return result;

            using (var package = new ExcelPackage(fileInfo))

            {

                var worksheet = package.Workbook.Worksheets.FirstOrDefault();

                if (worksheet == null || worksheet.Dimension == null)

                    return result;

                int rows = worksheet.Dimension.End.Row;

                for (int row = 2; row <= rows; row++)

                {

                    if (double.TryParse(worksheet.Cells[row, 2].Text, out double exactArea) &&

                        double.TryParse(worksheet.Cells[row, 3].Text, out double monteCarlo) &&

                        double.TryParse(worksheet.Cells[row, 4].Text, out double countPoints))

                    {

                        result.Add((exactArea, monteCarlo, countPoints));

                    }

                }

            }

            return result;

        }

        /// <summary>

        /// Метод для создания excel файла и записи туда результатов вычислений

        /// </summary>

        private void CreateDoc()

        {

            try

            {

                // создаем файл Excel

                FileInfo fileInfo = new FileInfo(ConfigManager.GetSavePath);

                using (var package = new ExcelPackage())

                {

                    var worksheet = package.Workbook.Worksheets.FirstOrDefault()

                             ?? package.Workbook.Worksheets.Add("Результаты");

                    // если файл пустой, то создаем заголовки

                    if (worksheet.Dimension == null)

                    {

                        worksheet.Cells[1, 1].Value = "№";

                        worksheet.Cells[1, 2].Value = "Точное значение";

                        worksheet.Cells[1, 3].Value = "Monte Carlo";

                        worksheet.Cells[1, 4].Value = "Число точек";

                    }

                    // записываем результаты в файл

                    for (int i = 0; i < \_saveData.Count; i++)

                    {

                        worksheet.Cells[i + 2, 1].Value = i + 1;

                        worksheet.Cells[i + 2, 2].Value = \_saveData[i].exactArea;

                        worksheet.Cells[i + 2, 3].Value = \_saveData[i].monteCarlo;

                        worksheet.Cells[i + 2, 4].Value = \_saveData[i].countPoints;

                    }

                    // сохраняем файл

                    package.SaveAs(fileInfo);

                }

            }

            catch (Exception ex)

            {

                // если произошла ошибка, то выводим сообщение об ошибке

                MessageBox.Show("Произошла ошибка сохранения результатов (см. Логи)", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);

                Logger.Log(ex);

            }

        }

    }

}

Класс для логирования

using System.IO;

using System.Windows;

namespace GeoCalc.Resources.Utilities

{

    /// <summary>

    /// Класс для логгирования

    /// </summary>

    internal static class Logger

    {

        /// <summary>

        /// Путь до файла логов

        /// </summary>

        private static readonly string \_logFilePath = ConfigManager.GetLogPath;

        /// <summary>

        /// Объект для синхронизации доступа к файлу логов

        /// </summary>

        private static readonly object \_lock = new object();

        /// <summary>

        /// Уровни логгирования

        /// </summary>

        public enum LogLevel { Info, Warning, Error }

        static Logger()

        {

            try

            {

                lock (\_lock)

                {

                    // Записываем начало работы приложения

                    File.WriteAllText(\_logFilePath, $"[{DateTime.Now:dd-MM-yyyy HH:mm:ss}] Приложение запущенно\n");

                }

            }

            catch (Exception)

            {

                // Если не получилось записать в файл, то выводим ошибку

                MessageBox.Show("Неправильно указан путь до логов", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);

            }

        }

        /// <summary>

        /// Записывает сообщение в файл логов

        /// </summary>

        /// <param name="message">Сообщение</param>

        /// <param name="level">Уровень логгирования</param>

        /// <returns>0 , -1 - ошибка</returns>

        public static int Log(string message, LogLevel level = LogLevel.Info)

        {

            try

            {

                lock (\_lock)

                {

                    // Записываем сообщение в файл

                    File.AppendAllText(\_logFilePath, $"[{DateTime.Now:dd-MM-yyyy HH:mm:ss}] [{level}] {message}\n");

                    return 0;

                }

            }

            catch (Exception ex)

            {

                // Если не получилось записать в файл, то выводим ошибку

                MessageBox.Show($"Ошибка логгирования: {ex.Message}", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);

                return -1;

            }

        }

        /// <summary>

        /// Записывает объект в файл логов

        /// </summary>

        /// <param name="obj">Объект</param>

        public static void Log(object obj)

        {

            if (obj == null)

            {

                Log("NULL", LogLevel.Warning);

                return;

            }

            Log(obj.ToString());

            if (obj is Exception ex)

            {

                Log($"Exception stack: {ex.Message}", LogLevel.Error);

            }

        }

    }

}

Класс для подсчета площади

using System.Collections.Concurrent;

namespace GeoCalc.Resources.Utilities

{

    /// <summary>

    /// Класс для подсчета площади большей половины окружности

    /// </summary>

    public class MonteCarloCircle

    {

        private readonly Random \_random;

        private readonly int \_countPoints;

        private readonly double \_x0, \_y0, \_radius, \_seperatorLine;

        /// <summary>

        /// Событие, которое происходит при изменении статуса расчетов

        /// </summary>

        public event Action<string>? Notification;

        /// <summary>

        /// Конструктор

        /// </summary>

        /// <param name="x0">x-координата центра окружности</param>

        /// <param name="y0">y-координата центра окружности</param>

        /// <param name="radius">радиус окружности</param>

        /// <param name="seperatorLine">y-координата линии, разделяющей окружность на две половины</param>

        /// <param name="countPoints">количество точек, используемых в методе Монте-Карло</param>

        public MonteCarloCircle(double x0, double y0, double radius, double seperatorLine, int countPoints)

        {

            \_x0 = x0;

            \_y0 = y0;

            \_radius = radius;

            \_seperatorLine = seperatorLine;

            \_countPoints = countPoints;

            \_random = new Random();

        }

        /// <summary>

        /// Подсчет площади большей половины окружности

        /// </summary>

        /// <returns>площадь большей половины окружности</returns>

        public double GetLargerPartArea()

        {

            double h = \_y0 - \_seperatorLine;

            if (Math.Abs(h) >= \_radius)

            {

                return Math.PI \* \_radius \* \_radius;

            }

            double R = \_radius;

            double y = Math.Abs(h);

            double theta = 2 \* Math.Acos(y / R);

            double segmentArea = 0.5 \* R \* R \* (theta - Math.Sin(theta));

            double largerPartArea = Math.PI \* R \* R - segmentArea;

            return largerPartArea;

        }

        /// <summary>

        /// Подсчет площади большей половины окружности с точностью

        /// </summary>

        /// <returns>площадь большей половины окружности, абсолютная ошибка, относительная ошибка, флаг, указывающий, является ли большей половиной верхняя</returns>

        public async Task<(double estimatedMaxHalfArea, List<MonteCarloPoint> points, double absoluteError, double relativeError, bool isUpper)> SimulateWithPointsWithAccuracyAsync()

        {

            return await Task.Run(() =>

            {

                Notification?.Invoke("Расчеты начались...");

                var points = new ConcurrentBag<MonteCarloPoint>();

                int insideCount = 0;

                int aboveCount = 0;

                double minX\_mc = \_x0 - \_radius;

                double maxX\_mc = \_x0 + \_radius;

                double maxY\_mc = \_y0 + \_radius;

                double minY\_mc = \_y0 - \_radius;

                Parallel.For(0, \_countPoints, i =>

                {

                    double x = \_random.NextDouble() \* (maxX\_mc - minX\_mc) + minX\_mc;

                    double y = \_random.NextDouble() \* (maxY\_mc - minY\_mc) + minY\_mc;

                    bool isInside = IsInsideCircle(x, y);

                    bool isAbove = false;

                    if (isInside)

                    {

                        Interlocked.Increment(ref insideCount);

                        if (y < \_seperatorLine)

                        {

                            isAbove = true;

                            Interlocked.Increment(ref aboveCount);

                        }

                    }

                    points.Add(new MonteCarloPoint(x, y, isInside, isAbove));

                });

                double fullCircleArea = Math.PI \* \_radius \* \_radius;

                double pAbove = (double)aboveCount / insideCount;

                double aboveArea = pAbove \* fullCircleArea;

                double belowArea = fullCircleArea - aboveArea;

                double maxHalfArea = Math.Max(aboveArea, belowArea);

                bool isUpper = aboveArea <= belowArea;

                double exactArea = GetLargerPartArea();

                double absoluteError = Math.Abs(maxHalfArea - exactArea);

                double relativeError = absoluteError / exactArea;

                Notification?.Invoke("Расчеты завершены.");

                return (maxHalfArea, points.ToList(), absoluteError, relativeError, isUpper);

            });

        }

        /// <summary>

        /// Проверка, находится ли точка внутри окружности

        /// </summary>

        /// <param name="x">x-координата точки</param>

        /// <param name="y">y-координата точки</param>

        /// <returns>true, если точка находится внутри окружности, false иначе</returns>

        private bool IsInsideCircle(double x, double y)

        {

            double dx = x - \_x0;

            double dy = y - \_y0;

            return dx \* dx + dy \* dy <= \_radius \* \_radius;

        }

    }

}

Класс для хранения информации о точке, сгенерированной методом Монте–Карло

namespace GeoCalc.Resources.Utilities

{

    /// <summary>

    /// Класс для хранения информации о точке, сгенерированной методом Монте-Карло

    /// </summary>

    public class MonteCarloPoint

    {

        /// <summary>

        /// Координата X точки

        /// </summary>

        public double X { get; }

        /// <summary>

        /// Координата Y точки

        /// </summary>

        public double Y { get; }

        /// <summary>

        /// Признак того, что точка находится внутри окружности

        /// </summary>

        public bool IsInsideCircle { get; }

        /// <summary>

        /// Признак того, что точка находится выше разделительной прямой

        /// </summary>

        public bool IsAboveSeperator { get; }

        /// <summary>

        /// Конструктор для создания объекта MonteCarloPoint

        /// </summary>

        /// <param name="x">Координата X точки</param>

        /// <param name="y">Координата Y точки</param>

        /// <param name="isInsideCircle">Признак того, что точка находится внутри окружности</param>

        /// <param name="isAboveSeperator">Признак того, что точка находится выше разделительной прямой</param>

        public MonteCarloPoint(double x, double y, bool isInsideCircle, bool isAboveSeperator)

        {

            X = x;

            Y = y;

            IsInsideCircle = isInsideCircle;

            IsAboveSeperator = isAboveSeperator;

        }

    }

}

Страница расчета

using GeoCalc.Resources.Model;

using GeoCalc.Resources.Utilities;

using System.Windows;

using System.Windows.Input;

using System.Windows.Media;

namespace GeoCalc.Resources.ViewModel

{

    /// <summary>

    /// ViewModel для страницы расчета

    /// </summary>

    public class CalcVM: ViewModelBase

    {

        private readonly PageModel \_pageModel;

        /// <summary>

        /// Минимальное количество точек

        /// </summary>

        public int GetMinCountPoint { get; } = ConfigManager.GetMinPoint;

        /// <summary>

        /// Максимальное количество точек

        /// </summary>

        public int GetMaxCountPoint { get; } = ConfigManager.GetMaxPoint;

        /// <summary>

        /// Количество точек

        /// </summary>

        public int CountPoint { get => \_pageModel.CountPoint; set { \_pageModel.CountPoint = value; OnPropertyChanged(); }}

        /// <summary>

        /// Радиус окружности

        /// </summary>

        public int Radius { get => \_pageModel.Radius; set { \_pageModel.Radius = value; OnPropertyChanged(); ResizeCircleDrawRequest(); } }

        /// <summary>

        /// Координата X начала окружности

        /// </summary>

        public int StartX { get => \_pageModel.StartX; set { \_pageModel.StartX = value; OnPropertyChanged(); ResizeCircleDrawRequest(); } }

        /// <summary>

        /// Координата Y начала окружности

        /// </summary>

        public int StartY { get => \_pageModel.StartY; set { \_pageModel.StartY = value; OnPropertyChanged(); ResizeCircleDrawRequest(); } }

        /// <summary>

        /// Координата линии разграничения

        /// </summary>

        public int SeperatorLine { get => \_pageModel.SeperatorLine; set { \_pageModel.SeperatorLine = value; OnPropertyChanged(); ResizeCircleDrawRequest(); } }

        /// <summary>

        /// Доступность кнопки сохранения

        /// </summary>

        public bool SaveIsEnabled { get => \_pageModel.SaveIsEndabled; set { \_pageModel.SaveIsEndabled = value; OnPropertyChanged(); } }

        /// <summary>

        /// Результат расчета

        /// </summary>

        public double CalculationResult { get => \_pageModel.CalculationResult; set { \_pageModel.CalculationResult = value; OnPropertyChanged(); } }

        /// <summary>

        /// Ошибка расчета

        /// </summary>

        public double Error { get => \_pageModel.Error; set { \_pageModel.Error = value; OnPropertyChanged(); } }

        /// <summary>

        /// Площадь большей части окружности

        /// </summary>

        public double LargerArea { get => \_pageModel.LargerArea; set { \_pageModel.LargerArea = value; OnPropertyChanged(); } }

        /// <summary>

        /// Абсолютная ошибка

        /// </summary>

        public double AbsoluteError { get => \_pageModel.AbsoluteError; set { \_pageModel.AbsoluteError = value; OnPropertyChanged(); } }

        /// <summary>

        /// Команда на расчет

        /// </summary>

        public ICommand Calculating {  get; set; }

        /// <summary>

        /// Команда на сохранение

        /// </summary>

        //public ICommand Saving { get; set; }

        /// <summary>

        /// Событие на отрисовку окружности

        /// </summary>

        public event Action<int, int, int, int> RequestDrawCircle;

        /// <summary>

        /// Событие на отрисовку точек

        /// </summary>

        public event Action<int,int,int,int, List<MonteCarloPoint>, bool> RequestDrawPoints;

        /// <summary>

        /// Action на отображение уведомления

        /// </summary>

        public Action<string, Brush> ShowNotification { get; set; }

        /// <summary>

        /// Обработчик кнопки расчета

        /// </summary>

        public async void OnCalculationButtonPressed(object sender)

        {

            if (CountPoint >= GetMinCountPoint && CountPoint <= GetMaxCountPoint)

            {

                Logger.Log("Выполняются расчеты");

                Logger.Log($"Данные окружности: X={StartX}; Y={StartY}; R={Radius}; C={SeperatorLine}");

                SaveIsEnabled = true;

                MonteCarloCircle circle = new MonteCarloCircle(StartX, StartY, Radius, SeperatorLine, CountPoint);

                circle.Notification += NotificationHandle;

                var res = await circle.SimulateWithPointsWithAccuracyAsync();

                CalculationResult = Math.Round(res.estimatedMaxHalfArea, 4);

                Error = Math.Round(res.relativeError, 4); ;

                AbsoluteError = Math.Round(res.absoluteError, 4);

                LargerArea = Math.Round(circle.GetLargerPartArea(),4);

                if (CircleInPlane())

                {

                    RequestDrawPoints?.Invoke(StartX, StartY, Radius, SeperatorLine, res.points, res.isUpper);

                    var dataSaver = new DataSaver();

                    dataSaver.SaveData(LargerArea, CalculationResult, CountPoint);

                    Logger.Log("Расчеты выполнены");

                    Logger.Log($"Результаты вычисления: Площадь={LargerArea}; Монте-Карло={CalculationResult}; Точность={Error}");

                }

                else

                {

                    ShowNotification?.Invoke("Координаты вне координатной плоскасти", Brushes.Orange);

                    Logger.Log("Координаты вне координатной плоскасти", Logger.LogLevel.Warning);

                }

            }

            else

            {

                ShowNotification?.Invoke("Количество точек вне диапозона", Brushes.Orange);

                Logger.Log("Количество точек вне диапозона", Logger.LogLevel.Warning);

            }

        }

        /// <summary>

        /// Проверка нахождения окружности в координатной плоскости

        /// </summary>

        private bool CircleInPlane()

        {

            return Math.Abs(StartX) <= 6 && Math.Abs(StartY) <= 6;

        }

        /// <summary>

        /// Обработчик события на отрисовку окружности

        /// </summary>

        private void ResizeCircleDrawRequest()

        {

            if (Math.Abs(StartX) <= 6 && Math.Abs(StartY) <= 6 )

            {

                RequestDrawCircle?.Invoke(StartX, StartY, Radius, SeperatorLine);

            }

        }

        /// <summary>

        /// Обработчик уведомления

        /// </summary>

        public void NotificationHandle(string message)

        {

            Application.Current.Dispatcher.Invoke(() =>

            {

                ShowNotification?.Invoke(message, Brushes.Green);

            });

        }

        /// <summary>

        /// Конструктор

        /// </summary>

        public CalcVM()

        {

            \_pageModel = new PageModel();

            CountPoint = 1000;

            StartX = -2;

            StartY = 1;

            Radius = 2;

            SeperatorLine = 2;

            SaveIsEnabled = false;

            Calculating = new RelayCommand(OnCalculationButtonPressed);

            //Saving = new RelayCommand(OnSaveButtonPressed);

        }

    }

}

Страница настроек

using GeoCalc.Resources.Model;

using GeoCalc.Resources.Utilities;

using Microsoft.Win32;

using System.Diagnostics;

using System.Windows;

using System.Windows.Input;

namespace GeoCalc.Resources.ViewModel

{

/// <summary>

/// ViewModel для окна настроек

/// </summary>

public class SettingVM : ViewModelBase

{

private NavigationVM \_navigationVM;

private string \_savePath;

private string \_logPath;

/// <summary>

/// Путь к файлу сохранения

/// </summary>

public string SavePath { get => \_savePath; set { \_savePath = value; OnPropertyChanged(); } }

/// <summary>

/// Путь к файлу лога

/// </summary>

public string LogPath { get => \_logPath; set { \_logPath = value; OnPropertyChanged(); } }

/// <summary>

/// Команда закрытия окна

/// </summary>

public ICommand ExitCommand { get; }

/// <summary>

/// Команда изменения пути к файлу сохранения

/// </summary>

public ICommand ChangeSavePathCommand { get; set; }

/// <summary>

/// Команда изменения пути к файлу лога

/// </summary>

public ICommand ChangeLogPathCommand { get; set; }

/// <summary>

/// Команда открытия файла лога

/// </summary>

public ICommand OpenLogCommand { get; set; }

private void OnChangeSavePath(object obj)

{

var dialog = new SaveFileDialog()

{

Filter = "Excel файл(\*.xlsx)|\*.xlsx"

};

if (dialog.ShowDialog() == true)

{

SavePath = dialog.FileName;

}

}

private void OnChangeLogPath(object obj)

{

var dialog = new SaveFileDialog()

{

Filter = "Текстовый файл(\*.txt)|\*.txt"

};

if (dialog.ShowDialog() == true)

{

LogPath = dialog.FileName;

}

}

private void OnExitCommand(object obj)

{

if (!string.IsNullOrEmpty(SavePath) || !string.IsNullOrEmpty(LogPath))

{

\_navigationVM.CalcCommand.Execute(this);

ConfigManager.SaveConfig(SavePath, LogPath);

}

else

{

MessageBox.Show("Поля не должы быть пустые", "Предупреждение", MessageBoxButton.OK, MessageBoxImage.Warning);

}

}

private void OnOpenLog(object obj)

{

Process.Start(new ProcessStartInfo

{

FileName = "explorer.exe",

Arguments = LogPath,

UseShellExecute = true

});

}

/// <summary>

/// Конструктор

/// </summary>

/// <param name="navigationVM">ViewModel для навигации</param>

public SettingVM(NavigationVM navigationVM)

{

\_navigationVM = navigationVM;

SavePath = ConfigManager.GetSavePath;

LogPath = ConfigManager.GetLogPath;

ExitCommand = new RelayCommand(OnExitCommand);

ChangeSavePathCommand = new RelayCommand(OnChangeSavePath);

ChangeLogPathCommand = new RelayCommand(OnChangeLogPath);

OpenLogCommand = new RelayCommand(OnOpenLog);

}

}

}