

СОДЕРЖАНИЕ

Введение.....	3
1 Анализ предметной области	4
1.1. Описание предметной области	4
1.2. Обзор аналогов.....	4
1.3. Требования к разрабатываемой ИС	7
1.3.1. Функциональные требования	7
1.3.2. Требования к интерфейсу	7
1.4. Обоснование выбора стека технологий.....	8
2 Проектирование.....	11
2.1. Проектирование системы.....	11
2.1.1. Определение группы пользователей.....	11
2.1.2. Функциональное моделирование	12
2.2. Проектирование интерфейсов	16
3 Реализация.....	18
3.1. Реализация основных функций	18
3.2. Реализация интерфейсов	31
3.3. Тестирование.....	36
4 Руководство пользователя	41
4.1. Описание установки	41
4.2. Описание запуска.....	42
4.3. Инструкции по работе	42
4.3.1. Ввод данных	42
4.3.2. Управление полем вывода	44
4.3.3. Использование заглавного меню.....	45
4.3.4. Работа с окном настроек	46
4.3.5. Формат математических функций	47
4.4. Сообщения пользователю	56
4.4.1. Ошибки ввода функций	56
4.4.2. Прочие сообщения.....	60
5 Мероприятия по информационной безопасности	63
5.1. Уязвимости программы.....	63
5.2. Меры предотвращения угроз.....	63
Заключение	65
Список источников	67

Введение

Компьютерные программы для построения графиков функций являются важными инструментами в образовании, науке и искусстве.

Такие приложения помогают ученикам освоить математические функции, производные, интегралы, некоторые законы физики и прочее. Программы визуализируют их поведение и помогают легко его изменять. Кроме этого, они реализуют наложение графиков функций для изучения их отличий. Эти программы позволяют лучше освоить разные системы координат [32] и параметрические функции [7].

Графическая визуализация графиков также упрощает расчёты, к примеру, экономические проблемы графическим методом решения задачи линейного программирования [15].

Одной из областей изобразительного искусства является создание изображений с помощью визуализации математических функций [54]. Такие изображения можно отнести к самостоятельным произведениям искусства и использовать их в дизайне. Компьютерные средства незаменимы при построении сложных и красивых графиков. Они ускоряют и упрощают этот процесс.

Таким образом, компьютерные программы для построения графиков функций – незаменимые инструменты в образовании, расчётных работах и искусстве.

Цель данного проекта – разработать простую в использовании программу для построения графиков функций «GraThing» (далее GraThing), позволяющую визуализировать двумерные математические функции, в том числе параметрические, в прямоугольной и полярной системах координат.

Задачи проекта:

- определить требования к приложению;
- определить платформу и язык программирования;
- разработать основную структуру модулей проекта и интерфейс;
- реализовать построение графиков;
- реализовать интерпретацию математических функций;
- добавить работу с пользовательскими настройками;
- добавить экспорт и импорт данных;
- создать интерфейс приложения;
- выполнить тестирование программы;
- разработать руководство пользователя;
- обеспечить информационную безопасность приложения.

1 Анализ предметной области

1.1. Описание предметной области

GraThing предназначена для построения графиков функций в прямоугольной и полярной системах координат. В программе доступна параметрическая запись функций и отрисовка в полярных координатах от радиуса.

В ней выделено несколько основных составляющих:

Функции

Пользователь вводит от 1 до 10 математических функций в строковом формате при помощи определённого синтаксиса. Программа их обрабатывает и сохраняет в память в формате функции-делегата. Есть возможность загрузить список функций из файла или сохранить их в новый.

Система координат

Может быть прямоугольной, полярной от радиуса и полярной от градуса. Графики строятся в соответствии с выбранной системой координат.

Диапазон вычисления функции

Пользователь может выбрать диапазон вычисления функции: минимальное и максимальное значение. Он не может заходить за границы максимального диапазона, который зависит от масштаба координатной сетки.

Графики

Программа строит графики сохранённых в формате функции-делегата функций на поле для рисования. При построении графиков используются настройки пользователя.

Поле для рисования

На поле рисуются координатные оси, сетка и графики в соответствии с настройками пользователя. По желанию пользователь может скрыть оси и сетку. Можно изменять размеры и соотношение сторон поля, двигать центр координат относительно поля, изменять масштаб рисования сетки и графиков. Изображение поля можно сохранить в память ПК.

Настройки пользователя

Время ожидания отрисовки, цвета графиков (10), ширина линии графика, цвет осей координат, цвет координатной сетки, цвет символов на координатной сетке. При первом запуске программы создаётся экземпляр стандартных настроек. Пользователь может изменить их по своему желанию. После внесения изменений они сохраняются в память устройства. При повторных запусках программа загружает сохранённые настройки.

1.2. Обзор аналогов

Существует множество программных решений, которые позволяют визуализировать функции, анализировать их свойства и проводить необходимые расчёты. Далее будут рассмотрены популярные программные средства для построения графиков функций, их преимущества и недостатки.

Desmos [43]

Онлайн-калькулятор для построения графиков, ориентированный на образовательные нужды.

Преимущества: Простой интерфейс, интерактивная работа с графиками.

Недостатки: Зависимость от Интернет-соединения.

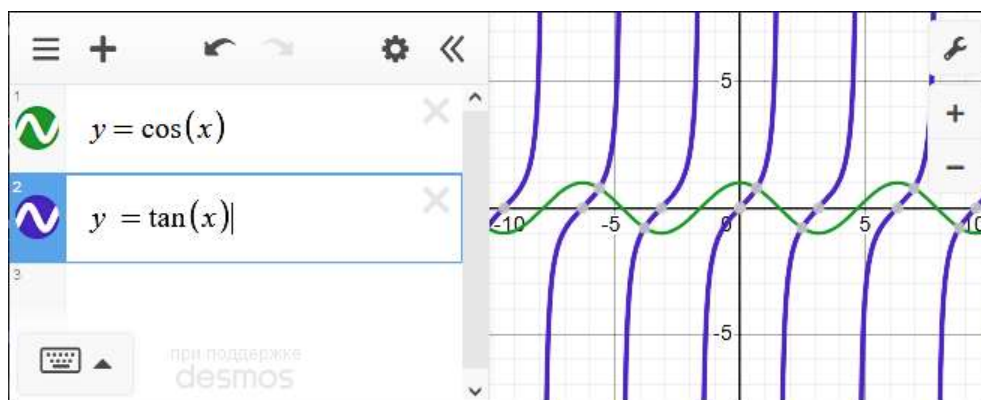


Рисунок 1 – Графики косинуса и синуса в Desmos

MATLAB [49]

Мощная среда для численных вычислений.

Преимущества: Высокая производительность, широкие возможности для научных расчётов и визуализации, множество встроенных функций.

Недостатки: Высокая стоимость лицензии, необходимость глубоких знаний для полной реализации возможностей.

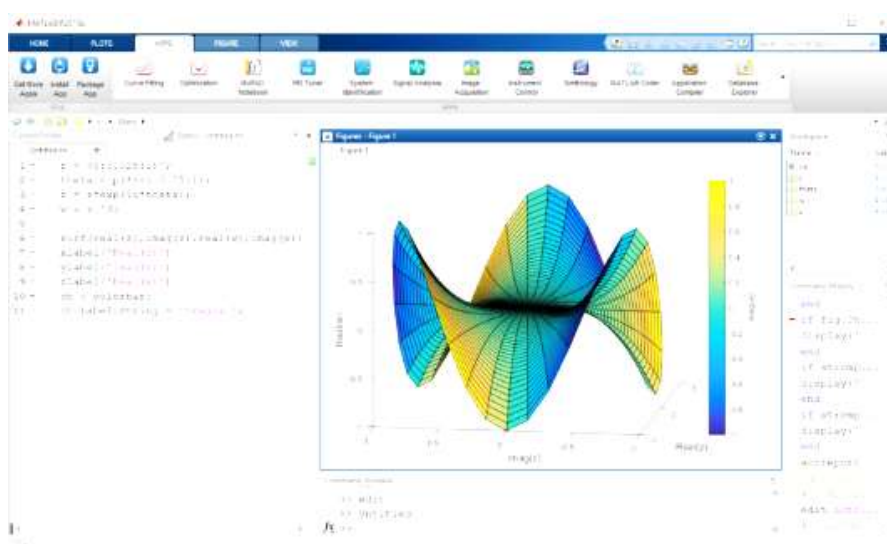


Рисунок 2 – Визуализация трёхмерной функции в MATLAB

Excel [51]

Программа для работы с табличными данными, включающая инструменты для построения графиков.

Преимущества: Широкая распространённость, минимальный порог входа, возможность работы с данными в табличном формате.

Недостатки: Ограниченные возможности построения сложных графиков, средняя стоимость.

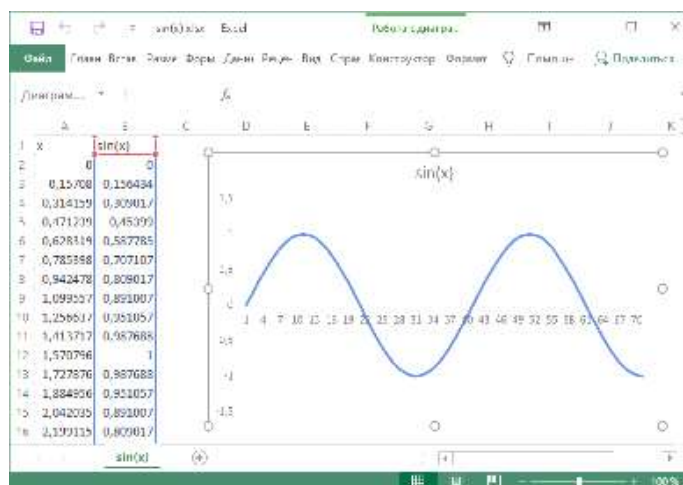


Рисунок 3 – График синуса в Excel

WolframAlpha [59]

Средство для сложных вычислений.

Преимущества: Широкий спектр возможностей, глубокая аналитика функций, возможность применения во множестве областей науки.

Недостатки: Зависимость от интернет-соединения, наличие платных функций, сложность освоения.

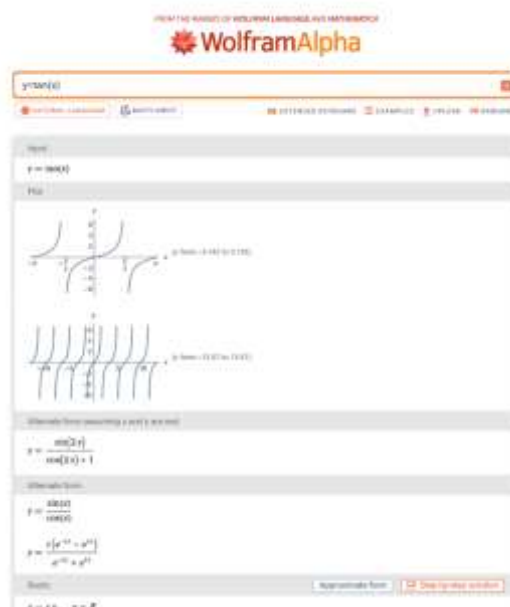


Рисунок 4 – Информация о функции тангенса в WolframAlpha

MathWay [50]

Инструмент для решения многих математических задач.

Преимущества: Простой интерфейс, широкий диапазон задач.

Недостатки: Низкая графическая функциональность, наличие платных функций.

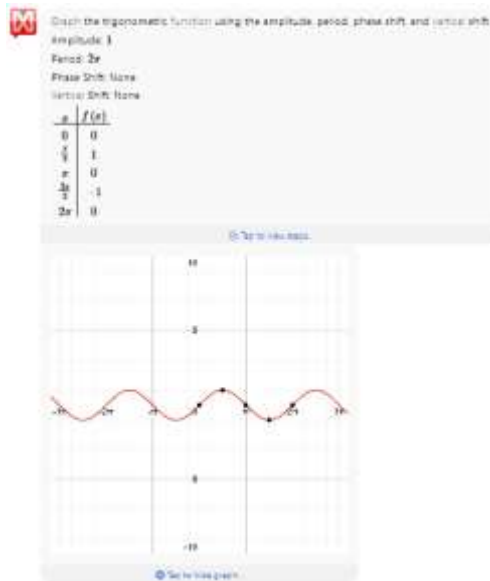


Рисунок 5 – График синуса в MathWay

Каждая из этих программ по-своему хороша, но либо она платная, либо имеет ограниченный графический функционал. В GraThing ставится упор на простоту интерфейса, автономность работы, что может быть полезно в учебных заведениях и местах с плохим интернет-соединением, и удобство визуальных настроек.

1.3. Требования к разрабатываемой ИС

1.3.1. Функциональные требования

К программе GraThing представляются следующие требования:

- доступна на ПК с windows 10 и выше;
- корректное отображение графиков заданных пользователем функций;
- одновременное рисование от 1 до 10 графиков функций;
- масштабирование координатной сетки;
- перемещение центра координат;
- работа с прямоугольной и полярной системами координат;
- работа с параметрической записью функций;
- работа с пользовательскими настройками;
- сохранение строковой записи функций в файл и загрузка их из файла;
- сохранение выводимого изображения.

1.3.2. Требования к интерфейсу

К интерфейсу GraThing представляются следующие требования:

- оконный;
- адаптивный дизайн в зависимости от разрешения и размеров окна;
- навигация при помощи меню;
- основные компоненты (ввод функций и отображение графиков) находятся на главном окне, есть возможность регулирования каждого компонента по размеру;
- отдельное окно настроек пользователя;
- изменение цветов в настройках при помощи колеса выбора цвета с квадратом яркости и насыщенности, в том числе можно использовать слайдеры;

- сглаженная отрисовка графиков;
- рядом с каждой строкой ввода функции должен стоять её номер от F0 до F9, он должен быть того же цвета, что и график функции;
- доступен на русском и английском языках в зависимости от настроек системы.

1.4. Обоснование выбора стека технологий

Для разработки можно использовать множество языков программирования, фреймворков, систем и сред. Далее будут рассмотрены некоторые из них.

Языки программирования

Популярными языками для написания оконных приложений под Windows являются C++ и C# [21].

C++ предоставляет средства низкоуровневого программирования, что необходимо для написания приложений, требующих очень высокого быстродействия. Но вести разработку на нём сложно, и он не имеет готовых средств разработки интерфейса.

C# разработан компанией Microsoft и широко используется для создания приложений под Windows. Он прост в использовании, но имеет широкие возможности. Так как он интегрирован с экосистемой Microsoft, он идеально подходит для разработки GraThing для Windows, ведь в ней не требуется быстрая обработка больших объёмов данных, и для неё нужен простой графический интерфейс.

Системы разработки для C#

Основные системы разработки Windows приложений, поддерживающие C# – Windows Forms (WinForms) и Windows Presentation Framework (WPF) [60].

WinForms – система разработки клиентских приложений Windows. Предоставляет визуальный редактор стандартных элементов управления Windows.

WPF – тоже система разработки клиентских приложений Windows. Однако, в ней есть собственные стандартные элементы, которые можно изменять как в визуальном редакторе, так при помощи языка разметки XAML. Благодаря этому она предоставляет куда большую гибкость в создании интерфейсов, нежели WinForms.

Несмотря на сложность WPF в сравнении с WinForms, WPF больше подходит для GraThing, ведь имеет больше возможностей для создания современного интерфейса, соответствующего требованиям.

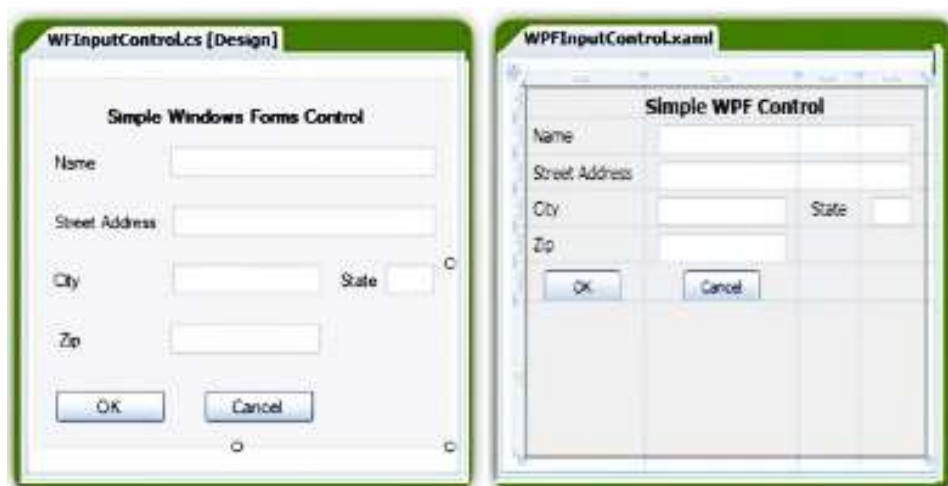


Рисунок 6 – Пример идентичных интерфейсов, созданных при помощи WinForms и WPF [52]

Среды разработки для WPF

JetBrains Rider – одна из сред разработки, поддерживающих WPF [61]. Он имеет умные функции редактирования кода, что полезно при создании приложений. Но в нём нет визуального редактора, и в целом у него низкая поддержка данной системы.

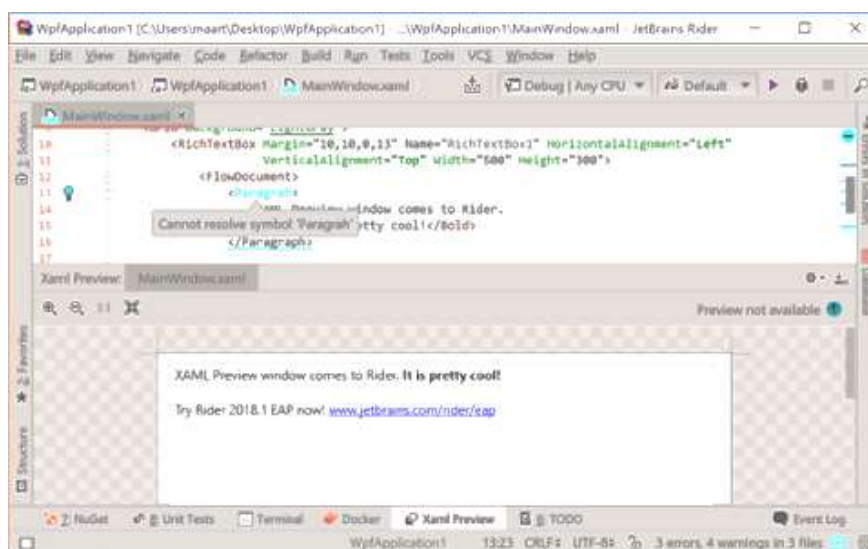


Рисунок 7 – Предпросмотр разметки XAML в JetBrains Rider

Visual Studio – ещё одна среда разработки для WPF [25]. Она предоставляет гораздо большие возможности разработки, в том числе визуальный редактор и множество специализированных средств. Она является предпочтительной для выбранной системы.

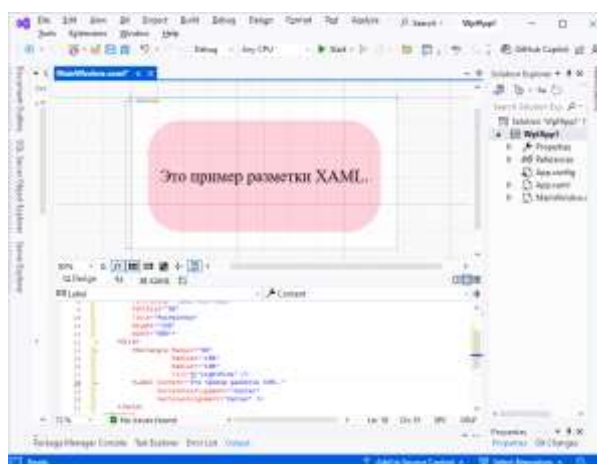


Рисунок 8 – Визуальный редактор с XAML в Visual Studio

Средства модульного тестирования для C# с Visual Studio

Модульное (unit) тестирование позволяет автоматически тестировать работоспособность отдельных модулей или частей кода программы. Оно позволяет избежать ошибок или быстро исправить их при обновлении или дополнении ПО новыми компонентами, не тратя время на проверку программного обеспечения целиком [56].

MSTest – официальный, интегрированный в Visual Studio инструмент для unit-тестирования [53]. Он превосходно интегрирован в выбранную среду разработки, но имеет несколько ограниченный функционал по сравнению со своими аналогами.

NUnit даёт большие возможности по сравнению с MSTest. Это зрелая экосистема с большим количеством разработчиков, но она слишком тяжеловесная для простых тестов.

XUnit – быстрый и более современный инструмент. Он позволяет писать простые, но мощные тесты с высокой скоростью выполнения. Несмотря на недостаток средств для углублённого тестирования, он является оптимальным вариантом для разработки GraThing.

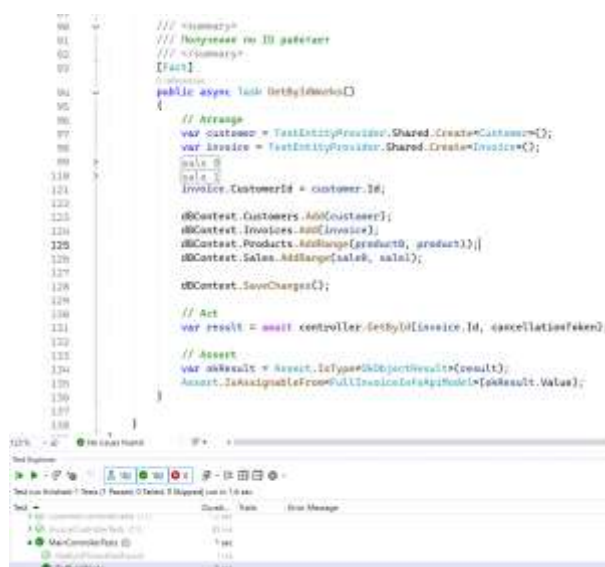


Рисунок 9 – Пример модульного теста с использованием xUnit и выводом результатов в Visual Studio

Форматы хранения данных

Для сохранения данных в файл необходимо выбрать формат.

XML (Extensible Markup Language) – открытый стандарт для хранения и обмена данными. Для записи данных используют теги: `<tag>data</tag>`. Он позволяет хранить архитектуру данных любой сложности [48]. Однако для записи XML необходимо использовать очень много места.

JSON (JavaScript Object Notation) – более легковесный формат хранения данных. Для записи данных используются пары ключ-значение: `{"tag": "data"}`. В GraThing выбран именно этот формат.

Разметка справки

В справке программы необходима разметка с заголовками, форматированием и списками. Это можно реализовать при помощи XAML, используемого для большей части приложения, но именно для справки был выбран HTML – язык гипертекстовой разметки, который вместе с CSS, каскадными таблицами стилей, обычно используется для создания веб-страниц [42, 45]. HTML в стандартном элементе управления WPF WebBrowser [58] позволяет создавать простую разметку, текст которой можно скопировать в буфер обмена, а также масштабировать.

В итоге, для разработки GraThing были выбраны язык программирования C# с применением системы разработки WPF (и, соответственно, .NET фреймворк), инструмент модульного тестирования xUnit в среде Visual Studio; хранение данных в формате JSON; разметка справки на HTML с CSS.

2 Проектирование

2.1. Проектирование системы

Проектирование приложения – это процесс разработки концепции, структуры и дизайна программы. Он включает в себя определение функциональных требований, архитектурных компонентов, интерфейса пользователя, базы данных и интеграции с другими системами [31].

2.1.1. Определение группы пользователей

Пользователи системы:

– оператор

Варианты использования системы описаны в таблицах с 1 по 6.

Таблица 1 – Вариант использования 1

Название	Построение графиков
Цель	Предоставить визуализацию введённых математических функций.
Актёры	Оператор
Описание	Оператор выбирает систему координат. Оператор вводит диапазон вычисления функций. Оператор вводит математические функции в строковом формате. Оператор запускает рисование. Выводятся построенные графики. Оператор изменяет масштаб координатной сетки и перемещает центр координат.
Тип варианта	Основной

Таблица 2 – Вариант использования 2

Название	Изменение пользовательских настроек
Цель	Изменить пользовательские настройки.
Актёры	Оператор
Описание	Оператор выбирает «Настройки». Оператор изменяет данные пользовательских настроек (см. п. 1.1.).
Тип варианта	Основной

Таблица 3 – Вариант использования 3

Название	Загрузка списка функций из файла
Цель	Загрузить список функций из файла на устройстве.
Актёры	Оператор
Описание	Оператор выбирает «Загрузить функции из файла». Оператор выбирает путь к файлу. Загружаются данные из файла.
Тип варианта	Второстепенный

Таблица 4 – Вариант использования 4

Название	Сохранение списка функций в файл
Цель	Сохранить список функций в файл на устройстве.
Актёры	Оператор
Описание	Оператор выбирает «Сохранить функции». Оператор выбирает путь к файлу. Строки функций сохраняются в файл.
Тип варианта	Второстепенный

Таблица 5 – Вариант использования 5

Название	Сохранение выводимого изображения в файл
Цель	Сохранить выведенное изображение координатной сетки, осей и функций в графический файл.
Актёры	Оператор
Описание	Оператор выбирает «Сохранить графики». Оператор выбирает путь к файлу. Изображение поля для рисования (см. п. 1.1.) сохраняется в файл.
Тип варианта	Второстепенный

Таблица 6 – Вариант использования 6

Название	Просмотр справки
Цель	Предоставить пользователю информацию о программе.
Актёры	Оператор
Описание	Оператор выбирает «Справка». Выводится руководство программы.
Тип варианта	Дополнительный

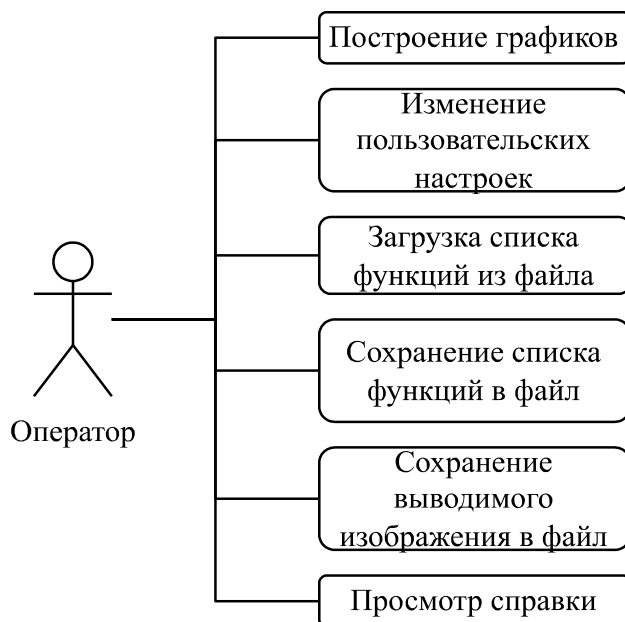


Рисунок 10 – Диаграмма вариантов использования

2.1.2. Функциональное моделирование

Модули программы GraThing: главный модуль, модуль интерфейсов, модуль интерпретатора математических функций и модуль визуализации графиков.

Главный модуль программы отвечает за связь и координацию всех остальных модулей системы. Он получает запросы от модуля интерфейсов, вызывает модули обработки, обрабатывает исключения и возвращает результат. Также он работает с файловой системой устройства.

Модуль интерфейсов обеспечивает работу пользователя с системой. В нём находятся все элементы управления, ввода и вывода. Он непосредственно связан только с главным модулем. Также он работает со стандартными окнами ОС.

Модуль интерпретатора функций получает на вход математические функции в строковом виде и выдаёт их в формате функции-делегата (см. п. 1.1.). При некорректных данных он выбрасывает специфичные исключения, которые обрабатывает главный модуль.

Модуль визуализации графиков получает на вход функции-делегаты и выполняет их построение. Также ему на вход могут прийти команды изменения масштаба координатной сетки или перемещения центра координат. Он возвращает главному модулю сгенерированное изображение.

Взаимодействие модулей показано на рис. 11 и 12.

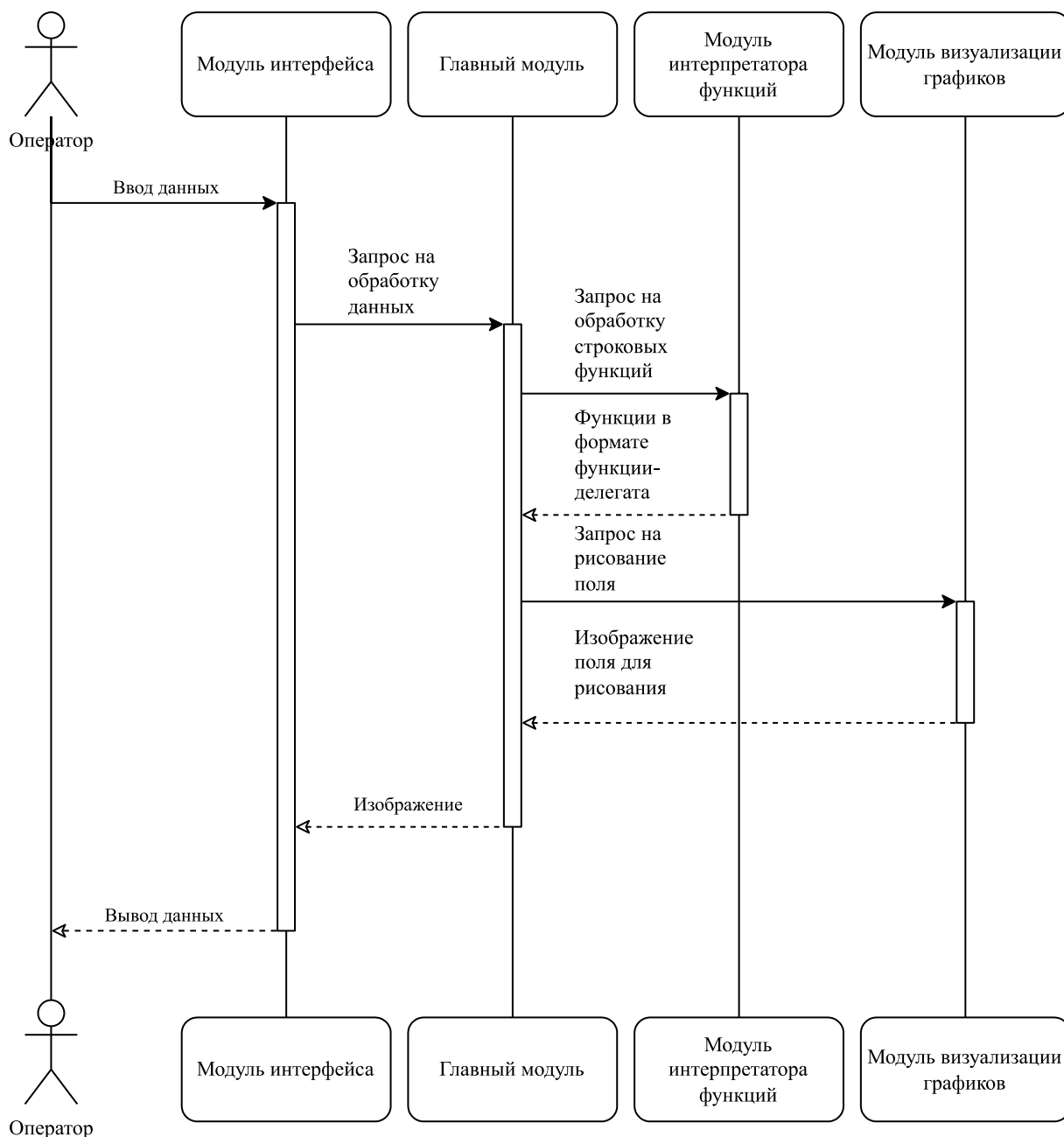


Рисунок 11 – Диаграмма последовательности

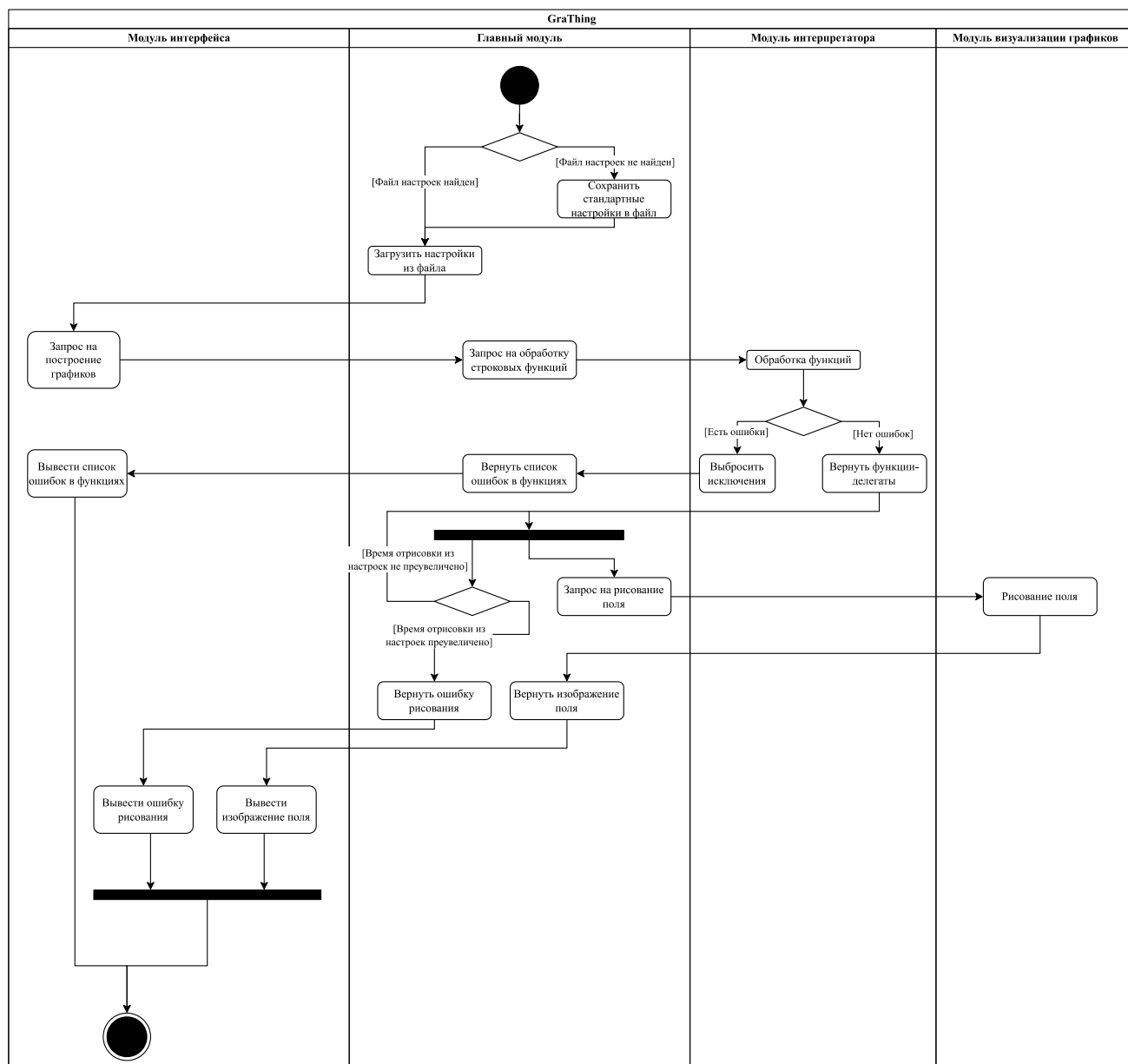


Рисунок 12 – Диаграмма деятельности

Основной алгоритм программы показан на рис. 13.

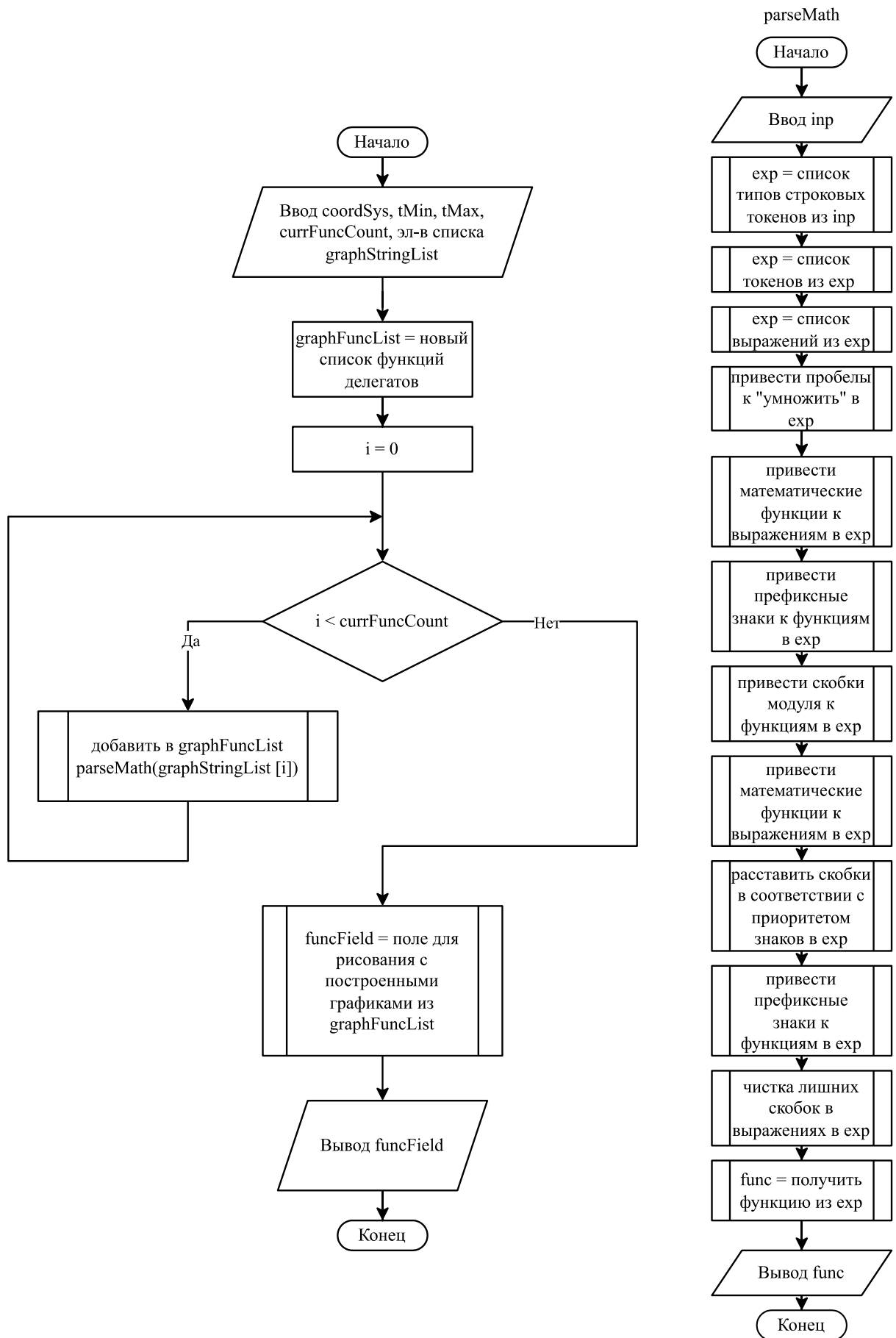


Рисунок 13 – Основной алгоритм программы

2.2. Проектирование интерфейсов

Для GraThing был выбран простой дизайн с округлыми элементами интерфейса. Основной фон – градиент от #4A4A4A до #373737, основной текст белый #FFFFFF, цвет целевых элементов управления белый, цвет текста на целевых элементах управления: чёрный. Основные цвета показаны на таб. 7; цвета графиков при стандартных настройках – на таб. 8; цвета координатных осей, сетки и текста на поле – на таб. 9.

Таблица 7 – Основные цвета интерфейса

#4A4A4A	#373737	Белый	Чёрный

Таблица 8 – Стандартные цвета графиков

#FF8E8E	#A5FB59	#5F9191	#F24D4D	#62CA04
#12ADE0	#FF00FF	#FFA500	#FFFF00	#BDA2FF

Таблица 9 – Стандартные цвета координатных осей, сетки и текста на поле

#D3D3D3	#808080

Шрифт основного интерфейса Bahnschrift (см. рис. 14);

Bahnschrift

Рисунок 14 – Шрифт Bahnschrift

Был создан логотип программы. Он представлен на рис. 15.



Рисунок 15 – Логотип программы

На рис. 16 и 17 представлены макеты интерфейса окон приложения.

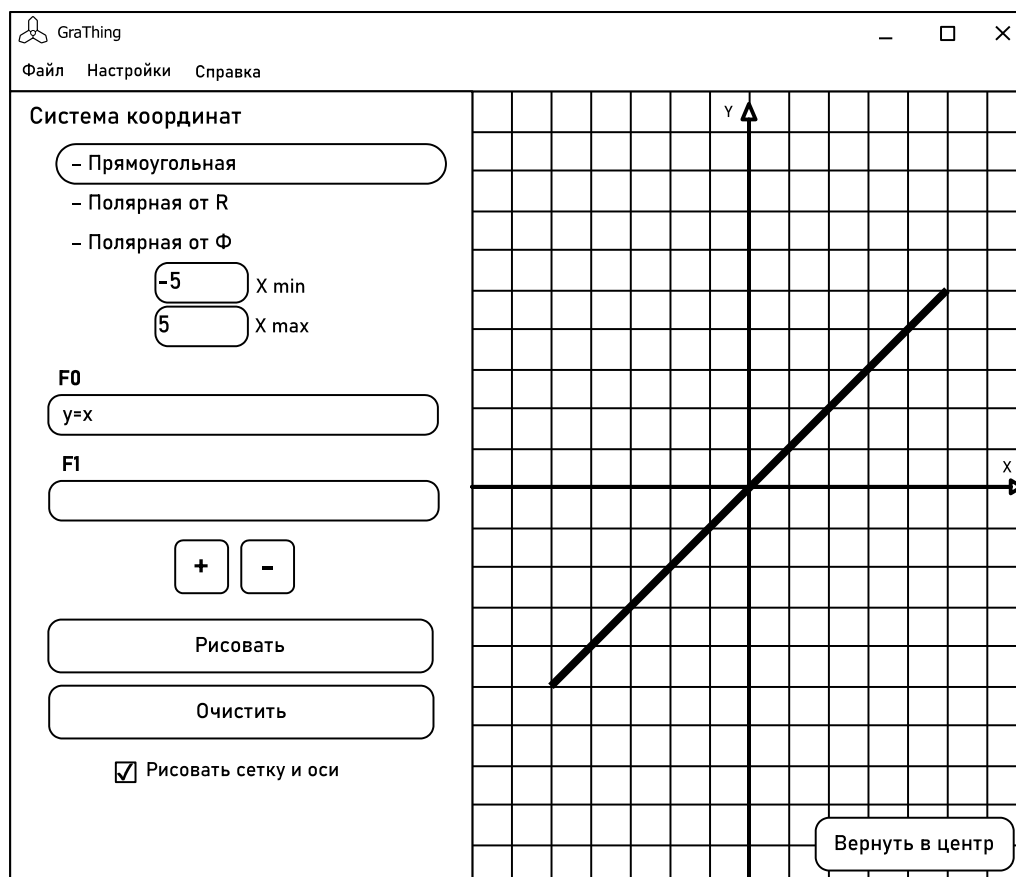


Рисунок 16 – Макет интерфейса главного окна

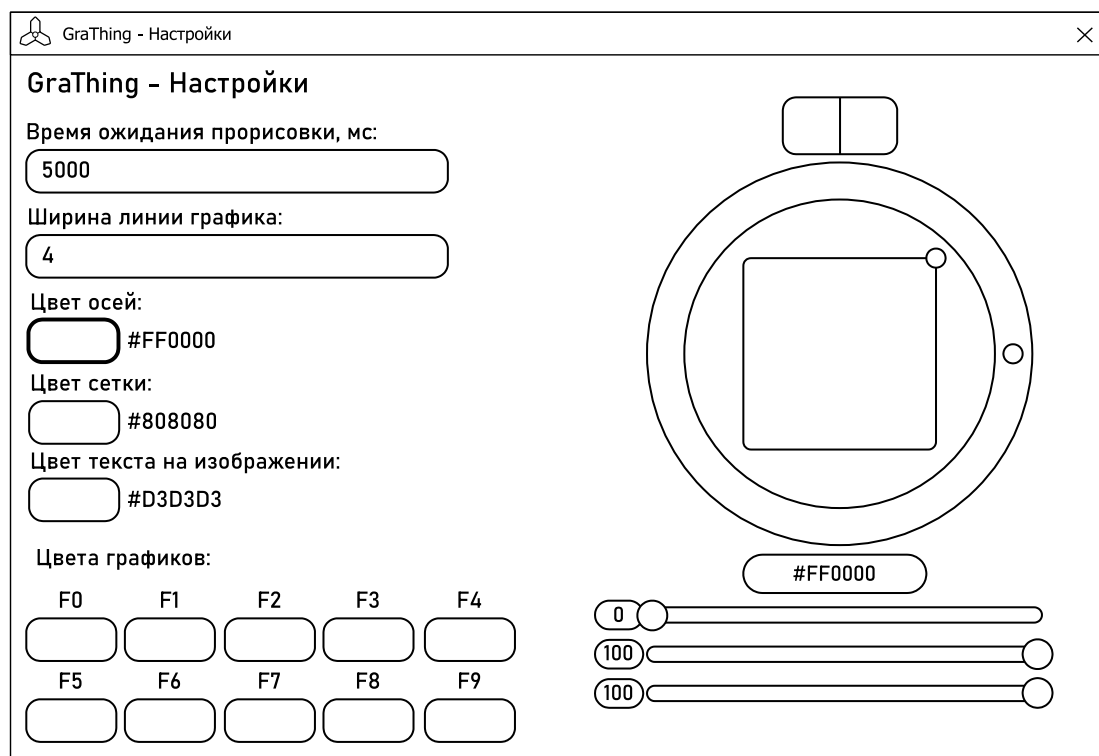


Рисунок 17 – Макет интерфейса окна настроек

3 Реализация

3.1. Реализация основных функций

Первый этап разработки – связка модуля интерфейсов с главным модулем. Она была реализована при помощи ICommand с привязками данных и вызовов главного модуля при оконных событиях. ICommand – контракт кода для команд, написанных для .NET Framework. Они предоставляют поведение для элементов интерфейса [46].

Привязка данных в WPF – способ представления и взаимодействия с данными. Элементы могут быть привязаны к разным источникам данных [26].

Для использования ICommand необходимо создать класс, имплементирующий данный интерфейс:

```
public class NoParamAction : ICommand
{
    private readonly Action execute;
    private readonly Predicate<object> canExecute;

    public NoParamAction(Action execute, Predicate<object> canExecute = null)
    {
        this.execute = execute ?? throw new
ArgumentNullException(nameof(execute));
        this.canExecute = canExecute;
    }

    bool ICommand.CanExecute(object parameter)
        => canExecute == null || canExecute(parameter);

    void ICommand.Execute(object parameter)
        => execute();

    event EventHandler ICommand.CanExecuteChanged
    {
        add { }
        remove { }
    }
}
```

Затем в классе данных окна создаётся поле с типом команды, которому в качестве значения передаётся функция-делегат. Далее в разметке объекту нужно указать привязку. Пример связи NoParamAction с кнопкой и комбинацией клавиш:

```
public NoParamAction UpdateFunctionsFieldCommand { get; private set; }

private async void UpdateFunctionsFieldAsync(...)
{ ... }
public MainAppModel(....)
{
    ...
    UpdateFunctionsFieldCommand =
new NoParamAction(() => UpdateFunctionsFieldAsync(...));
    ...
}
<Window.InputBindings>
    <KeyBinding Gesture="Ctrl+D"
```

```

        Command="{Binding UpdateFunctionsFieldCommand}" />
</Window.InputBindings>
<Button x:Name="ButtonDraw"
        Command="{Binding UpdateFunctionsFieldCommand}"
        ToolTip="Ctrl+D"/>

```

Для создания привязок в коде может использоваться подобный синтаксис:

```

var bindToListBox = new Binding(nameof(CoordSys))
{
    Source = this,
    Mode = BindingMode.TwoWay,
};
CoordSysListBox.SetBinding(ListBox.SelectedIndexProperty, bindToListBox);

```

Для корректной работы с привязками, в файле разметки указывается класс контекста данных, а в конструкторе класса – конкретный его экземпляр:

```

<Window ... d:DataContext="{d:DesignInstance Type=userpref:UserprefModel}"> ...

```

```

public UserprefSettingsWindow(UserprefModel userprefModel)
{
    InitializeComponent();
    DataContext = userprefModel;
}

```

Чаще всего привязки используются для изменения данных объектов через элементы интерфейса:

```

<TextBox Text="{Binding RenderWaitTime}" />

```

Привязки в C# с WPF работают только со свойствами зависимостей. Эти свойства позволяют задать значение по умолчанию и логику, вызываемую при изменении данных. У стандартных элементов управления такие поля уже есть, а в пользовательских классах необходимо реализовать их самостоятельно:

```

public readonly static DependencyProperty HueProperty =
    DependencyProperty.Register(nameof(Hue), typeof(double), typeof(HSBControl),
        new PropertyMetadata(DefaultHue, OnColorsChanged, CoerceHue));
public double Hue
{
    get => (double)GetValue(HueProperty);
    set => SetValue(HueProperty, value);
}
private static object CoerceHue(DependencyObject dependObj, object value)
{
    if (value is double val)
    {
        return CoerceHSB(val, Cnst.MaxHue);
    }
    return 0;
}
private static void OnColorsChanged(DependencyObject dependObj,
DependencyPropertyChangedEventArgs evArgs)
{
    if (dependObj is HSBControl con)
    {
        con.OnColorsChanged();
    }
}

```

Конвертеры типов необходимы для привязки нестандартных данных или нестандартным способом. Пример такого класса, приведение объекта кисти SolidColorBrush к текстовому представлению его цвета в шестнадцатеричном формате [39]:

```
public class SWMbrushHexConverter : IValueConverter
{
    public object Convert(object value, Type targetType, object parameter,
        CultureInfo culture)
    {
        if (value is string hex)
        {
            var color = hex.ToColor();
            var output = new SolidColorBrush(Color.FromRgb(color.R, color.G,
color.B));
            output.Freeze();
            return output;
        }
        return Colors.Black.ToBrush();
    }
    public object ConvertBack(object value, Type targetType, object parameter,
        CultureInfo culture)
    {
        if (value is SolidColorBrush brush)
        {
            var color = brush.Color;
            return System.Drawing.Color.FromArgb(color.A, color.R, color.G,
color.B).ToHex();
        }
        return string.Empty;
    }
}
```

Для изоляции логики отдельных модулей разрабатываются интерфейсы. Интерфейс – группа функциональностей, которые должен реализовать другой класс [19].

Интерфейсы позволяют организовать внедрение зависимостей – механизм, позволяющий сделать архитектуру проекта гибкой и слабосвязанной [13]. В отдельном модуле создаётся интерфейс, описывающий необходимые функции. В модуле 1 создаётся класс А, имплементирующий данный интерфейс. В модуле 2 создаётся класс Б, которому нужна функциональность А, но нет к ней доступа. В параметры конструктора Б добавляется объект с типом интерфейса. Теперь, при создании экземпляра Б в конструктор передаётся экземпляр А. Так реализуется внедрение зависимостей. Пример, связка класса приложения и класса данных главного окна посредством описанного выше механизма:

```
public interface IApplication
{
    bool IsCursorLoading { get; set; }
    void Close();
    void MouseCursorLoading();
    void MouseCursorReset();
}
internal class App : Application, IApplication
{
    internal App()
    {
        ...
    }
    bool IApplication.IsCursorLoading { get; set; }
    void IApplication.Close() => Shutdown();
}
```

```

void IApplication.MouseCursorLoading()
{
    (this as IApplication).IsCursorLoading = true;
    Mouse.OverrideCursor = Cursors.Wait;
}
void IApplication.MouseCursorReset()
{
    (this as IApplication).IsCursorLoading = false;
    Mouse.OverrideCursor = null;
}
}
public MainAppModel(IApplication currApp, ...)
{
    ...
    this.currApp = currApp;
    ...
}
private static void Main()
{
    var app = new App();
    ...
    var viewModel = new MainAppModel(app, ...);
    ...
}

```

В программе используется архитектурный паттерн программирования **Model-View-ViewModel** (MVVM) [9]. Он позволяет отделить логику приложения от визуальной части, что упрощает разработку, отладку и поддержку приложения. Данный паттерн первоначально нацелен именно на WPF приложения.

Составные части MVVM это:

- **Модель**, то есть данные и их обработка;
- **Модель Интерфейса** – логика, связывающая интерфейс с данными;
- **Интерфейс**, в том числе логика ввода и вывода.

В GraThing моделями являются модули построения графиков и обработки математических функций, модели интерфейса есть у главного окна и окна настроек, а интерфейс – сами окна. На рис. 18 показана схема паттерна MVVM с указанием модулей программы.

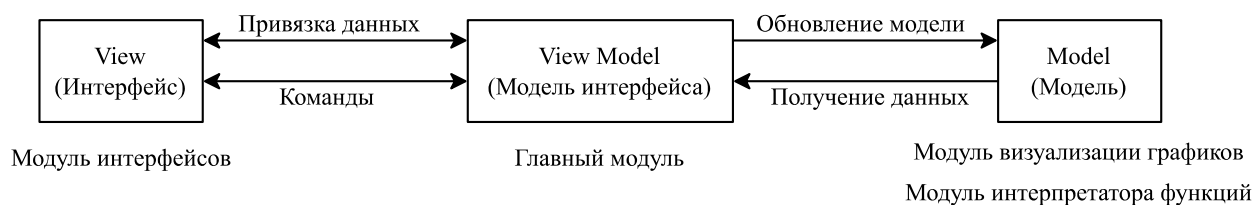


Рисунок 18 – Схема архитектурного паттерна MVVM

Далее будут подробнее рассмотрена реализация основных модулей по отдельности.

Модуль интерфейсов

Модуль интерфейсов включает в себя: классы окон, классы пользовательских элементов управления, файл стилей интерфейса, поставщик системных диалоговых окон, класс вызова окон, конвертеры типов, а также класс приложения и программы.

Данный модуль реализует основной ввод и вывод, а окно настроек изменяет данные пользовательских предпочтений.

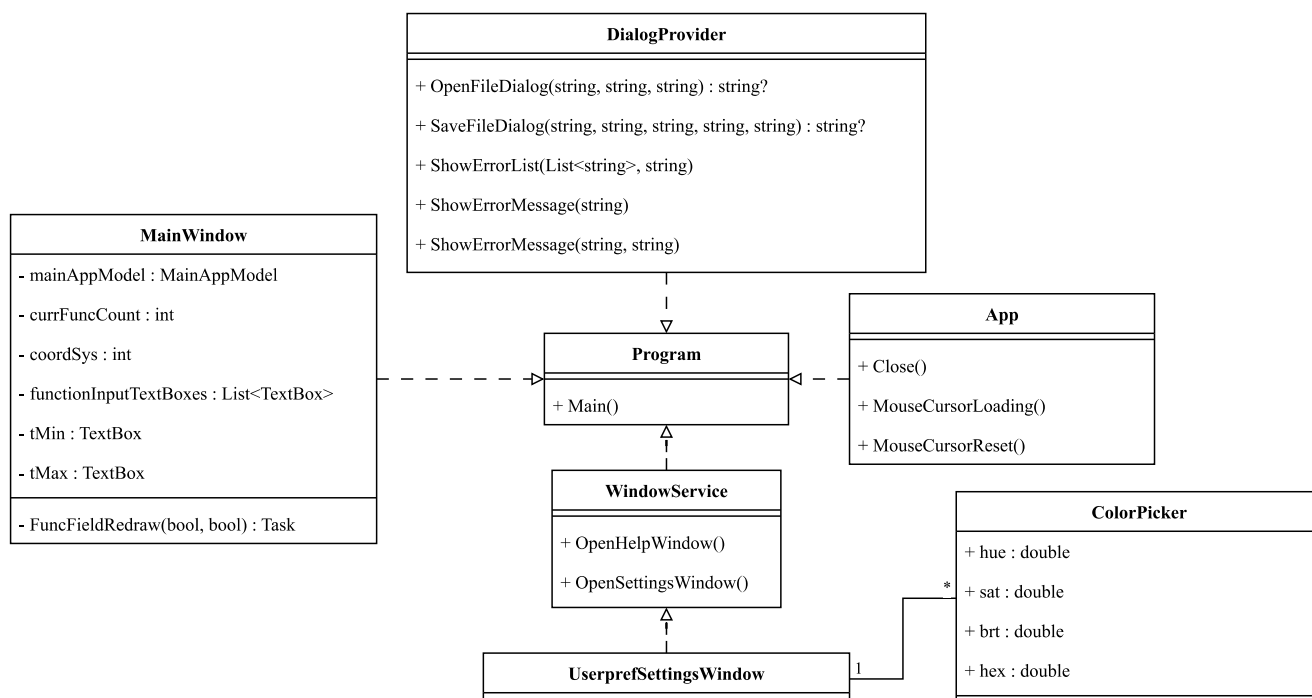


Рисунок 19 – Диаграмма классов модуля интерфейсов

Главный модуль

В главном модуле находится класс контекста главного окна, сервис работы с файлами, класс настроек и класс помощника для настроек. Для сохранения данных настроек в файл используется конвертация в формат JSON (см. п. 1.4).

Если есть необходимость записать нестандартный тип в JSON файл, нужно разработать класс конвертера:

```

public class StringListConverter : JsonConverter<StringList>
{
    public override StringList Read(ref Utf8JsonReader reader, Type typeToConvert,
    JsonSerializerOptions options)
    {
        var stringList = new StringList();
        if (reader.TokenType != JsonTokenType.StartArray)
        { throw new JsonException(); }
        while (reader.Read())
        {
            if (reader.TokenType == JsonTokenType.EndArray)
            { break; }
            if (reader.TokenType != JsonTokenType.String)
            { throw new JsonException(); }

            stringList.Add(reader.GetString());
        }
        return stringList;
    }
    public override void Write(Utf8JsonWriter writer, StringList value,
    JsonSerializerOptions options)
    {

```

```

        writer.WriteStartArray();
        foreach (var str in value.List)
        {
            writer.WriteStringValue(str);
        }
        writer.WriteEndArray();
    }
}

```

Реализация чтения и записи в JSON формат находится в классе помощника настроек:

```

public static string UserprefModelToJson(this UserprefModel settingsModel)
    => JsonSerializer.Serialize(settingsModel);
public static UserprefModel JsonToUserprefModel(this string settingsModelJson)
{
    var options = new JsonSerializerOptions();
    options.Converters.Add(new StringListConverter());
    return JsonSerializer.Deserialize<UserprefModel>(settingsModelJson, options);
}

```

Для работы с файлами есть отдельный класс `FileService`. Он организует чтение и запись файлов в систему устройства. Сервис файлов сохраняет строки функций, картинки поля и обновлённые настройки в файл, загружает строки и предыдущие настройки из файла. При работе с файлами очень вероятны ошибки, к примеру, при отсутствии файла. Поэтому необходимо ловить ошибки ввода-вывода, `IOException` [47].

Код сохранения настроек:

```

public void SaveUserprefModelToFile(UserprefModel userprefModel)
{
    var json = userprefModel.UserprefModelToJson();
    var fileName = Path.Combine(GetAppDataDir(), UserPrefFileName);
    if (fileName != null)
    {
        try
        {
            using (var outputFile = new StreamWriter(fileName))
            { outputFile.Write(json); }
        }
        catch (IOException ex)
        { dialogProvider.ShowErrorMessage(ex.Message); }
    }
}

```

Код загрузки настроек из файла:

```

public UserprefModel LoadUserprefModelFromFile()
{
    var output = UserprefModelHelper.DefaultUserprefModel;
    var fileName = Path.Combine(GetAppDataDir(), UserPrefFileName);
    if (fileName != null)
    {
        lastLocation = fileName;
        try
        {
            using (var fileStream = File.Open(fileName, FileMode.Open))
            {
                using (var reader = new StreamReader(fileStream))
                { output = reader.ReadToEnd().JsonToUserprefModel(); }
            }
        }
    }
}

```

```

        catch (FileNotFoundException)
        { dialogProvider.ShowErrorMessage(Resources.UserprefNotFound +
Environment.NewLine + Resources.TheDefaultUserprefWillBeLoaded); }
        catch (IOException ex)
        { dialogProvider.ShowErrorMessage(ex.Message + Environment.NewLine +
Resources.TheDefaultUserprefWillBeLoaded); }
        finally
        { SaveUserprefModelToFile(output); }
    }
    return output;
}

```

Сам объект настроек передаётся в модуль визуализации посредством внедрения зависимостей.

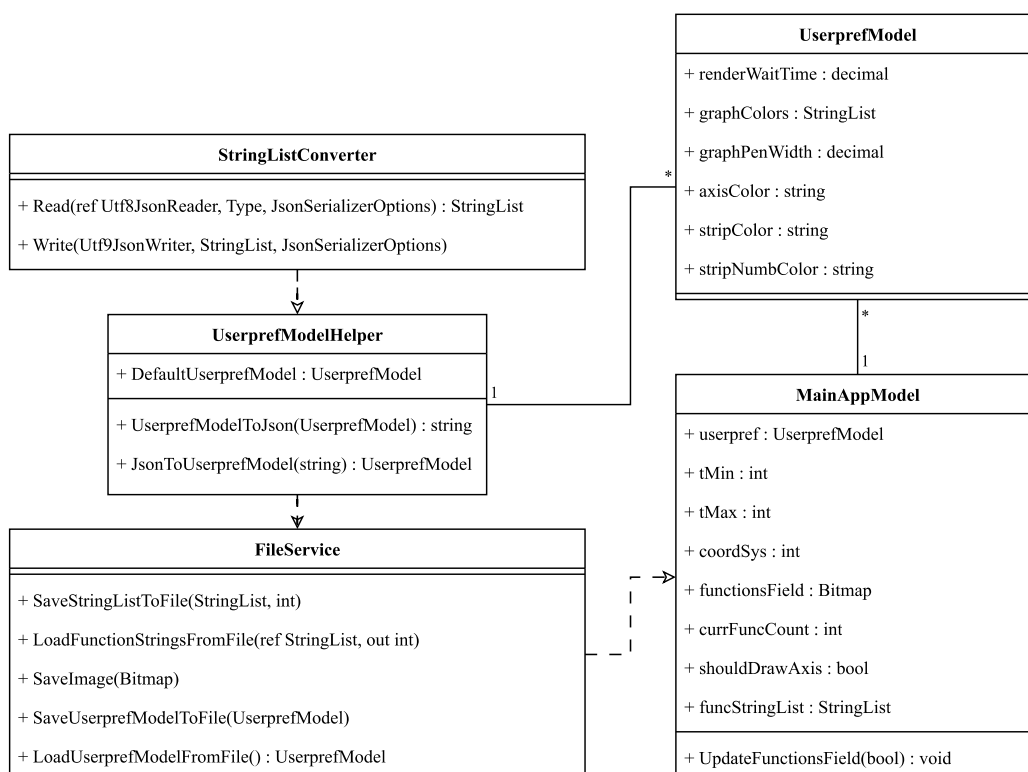


Рисунок 20 – Диаграмма классов главного модуля

Модуль визуализации графиков

Классы модуля: основной класс визуализации, вспомогательный класс визуализации, класс вычислений. Основной класс выполняет большую часть работы по генерации поля и имеет связь с главным модулем. Во вспомогательном классе находятся отдельные методы рисования графиков. В последнем классе – дополнительные функции вычислений, необходимые для построения.

При работе с этим модулем используются асинхронные операции – концепт в программировании, обеспечивающий выполнение отдельной функции параллельно основному ходу программы. Результат выполнения доступен спустя некоторое время. Это позволяет выполнять длительные операции, не прерывая выполнение основного кода [11].

В заголовке таких функций в C# используется ключевое слово `async`, и они обычно возвращают тип `Task`. Объекты с типом `Task` можно вызвать по ключевому слову `await`.

Чтобы код функции действительно выполнялся асинхронно, его нужно обернуть в `Task.Run(() => {}, token)`.

Для отмены выполнения асинхронной операции используется `CancellationTokenSource` [28]. Поле `Token` этого класса передаётся в асинхронную функцию, которая должна периодически проверять, была ли вызвана отмена, после чего должна прервать своё выполнение.

Пример:

```
private static async Task<....> GetGraphMathPointsAndDerivative(...
CancellationToken cancellation)
{
    ...
    await Task.Run(() =>
    {
        ...
        for (...)
        {
            if (!cancellation.IsCancellationRequested)
            { ... }
            else
            { return; }
        }
    }, cancellation);
    return (...);
}

public async Task DrawAll(bool onlyAxis, bool redrawingOnResize)
{
    ...
    var drawingTask = funcGraphicsRender.GetBitmap(onlyAxis, CancelSource.Token);
    var delayTask = Task.Delay((int)userprefModel.RenderWaitTime);
    Task completedTask = null;
    try
    { completedTask = await Task.WhenAny(drawingTask, delayTask); }
    catch (OperationCanceledException) { }
    if (completedTask != null)
    {
        if (completedTask == delayTask)
        {
            ...
            CancelSource.Cancel();
            ...
            CancelSource.Dispose();
        }
    }
    try
    {
        if (redrawingOnResize)
        {
            ...
            drawingTask = funcGraphicsRender.GetBitmap(onlyAxis,
CancelSource.Token);
        }
        (FunctionsField, MinTmin, MaxTmax) = await drawingTask;
        CancelSource.Dispose();
    }
}
```



```

    catch (ObjectDisposedException) { }
    catch (OperationCanceledException) { }
    ...
}

```

Для рисования графиков используется пространство имён System.Drawing. Оно предоставляет доступ к основным графическим функциям Windows [55]. Всё рисуется на объекте типа Graphics и методов по типу DrawPath или DrawLine. Объект Graphics зависит от Bitmap и синхронно изменяет его. После завершения рисования модуль выдаёт именно объект точечного рисунка.

Создание Bitmap и Graphics:

```

functionsFieldBitmap = new Bitmap(winWidth, winHeight);
canvas = Graphics.FromImage(functionsFieldBitmap);

```

Для определения разрывности функций вычисляется производная методом конечной разности [34] по формуле

$$(f(x+h) - f(x-h)) / 2h \quad (1)$$

Где f – данная функция; h – точность расчётов; x – абсцисса точки, для которой нужно вычислить значение производной.

Код функции:

```

public static double DerivativeAt
(this Func<double, (double x, double y)> func, double x, bool ofX)
=> ofX ? (func(x + Epsilon).x - func(x - Epsilon).x) / (2 * Epsilon)
: (func(x + Epsilon).y - func(x - Epsilon).y) / (2 * Epsilon);

```

Функция рисования графика строит список точек полученный и обработанный другими. Сначала генерируются точки и их производные численным методом. Далее обнуляются точки со слишком большой производной. Затем список точек приводится к экранным координатам и очищается от дубликатов. В конце концов, этот список рисуется.

Для того, чтобы нарисовать график в разных координатах, в функцию рисования всего лишь подставляется другой метод приведения к экранным координатам. При работе с полярными координатами, в метод приведения добавляется функция перевода полярных координат по формуле

$$x = r \cos \varphi; y = r \sin \varphi \quad (2)$$

Где x – абсцисса точки, y – ордината, r – радиус, φ – градус.

Смена функции приведения к экранным координатам реализована так:

```

private async Task DrawGraphs(Cancellation_token cancellation)
{
    switch (coordSys)
    {
        case CoordinateSystem.Cartesian:
        {
            mathToScreenCoord = MathToScreenCoord;
            break;
        }
        case CoordinateSystem.PolarFromR:
        {
            mathToScreenCoord = PolarFromRToScreenCoord;
            break;
        }
        case CoordinateSystem.PolarFromPhi:

```

```

        {
            mathToScreenCoord = PolarFromPhiToScreenCoord;
            break;
        }
        default:
        {
            mathToScreenCoord = MathToScreenCoord;
            break;
        }
    }
    for (var i = 0; i < graphFunctions.Count; i++)
    {
        await canvas.DrawGraph(graphFunctions[i], mathToScreenCoord,
        funcProp.GetGraphPen(i), tMin, tMax, step, cancellation);
    }
}

    Код функции рисования графика:
if (func != null)
{
    var penColor = pen.Color;
    if (penColor.A != byte.MaxValue)
    {
        pen = new Pen(new SolidBrush(Color.FromArgb(penColor.R, penColor.G,
        penColor.B)), pen.Width);
    }
    try
    {
        var funcAndDer = await func.GetGraphMathPointsAndDerivative(tMin, tMax,
        step, cancellation);
        var pointList = GetPointList(funcAndDer.CleanGraphMathPoints(),
        MathToScreenCoord);
        var paths = new List<GraphicsPath>();
        var newPath = new GraphicsPath();
        if (pointList.Count > 1)
        {
            for (var i = 0; i < pointList.Count - 1; i++)
            {
                cancellation.ThrowIfCancellationRequested();

                var point = pointList[i];
                var next = pointList[i + 1];
                if (point == Calc.UndefinedPoint || next == Calc.UndefinedPoint ||
                Calc.LineTooLong(point, next) || newPath.PointCount >= PathTooLong)
                {
                    paths.Add(newPath);
                    newPath = new GraphicsPath();
                }
                if (!Calc.LineTooLong(point, next))
                { newPath.AddLine(point, next); }
            }
            paths.Add(newPath);
            newPath.FillMode = FillMode.Alternate;
            foreach (var path in paths)
            {
                try
                {

```

```

        canvas.DrawPath(pen, path);
    }
    catch (OperationCanceledException) { }
}
else if (pointList.Count == 1)
{
    if (pointList[0] != Calc.UndefinedPoint)
    { canvas.DrawPoint(pointList[0], pen.Width, pen.Color); }
}
catch (OperationCanceledException) { }
}

```

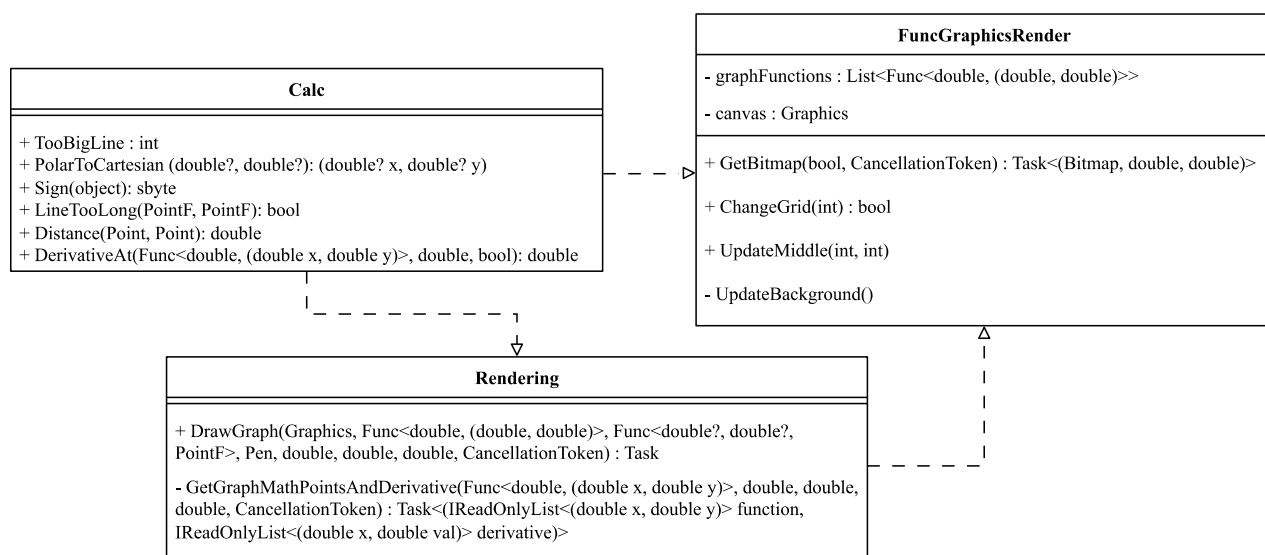


Рисунок 21 – Диаграмма классов модуля визуализации графиков

Модуль интерпретатора функций

Задача этого модуля – разбор математических функций. Модуль интерпретации включает в себя несколько подмодулей: основной подмодуль, подмодуль коллекций данных, подмодуль исключений, подмодуль токенов и подмодуль типов токенов.

Токен – отдельный блок математической функции, который для себя выделяет программа. Все токены наследуют интерфейс `IToken`. Выделены следующие токены: выражение, скобка модуля, открывающая скобка, закрывающая скобка, функция, значение, математический знак, пробел, слово. Вычисляемые токены, то есть выражение, слово (переменная) и значение, наследуют интерфейс `IOperand`.

Приведение списка токенов к формату функции делегата – задача вычисляемого токена `Expression`, выражения. Его данными является список токенов, который можно привести к виду функции.

Основной подмодуль содержит в себе логику интерпретации функций; этапы интерпретации см. в п. 2.1.2. рис. 13. Подмодуль типов токенов хранит несколько типов перечислений, в которые строка конвертируется при первом проходе интерпретатора, а также список типов математических знаков и функций. В подмодуле токенов находятся классы токенов, именно их обрабатывают последующие проходы.

Исключения, которые выбрасываются при некорректном вводе функции тоже находятся в отдельном подмодуле. Определены следующие исключения: открывающая скобка без пары, закрывающая скобка без пары, неверные скобки модуля, некорректная функция, неверный

числовой формат, неверный порядок операторов, некорректный токен, несколько переменных или некорректные токены, отсутствие аргументов у функции, неверное количество аргументов.

В подмодуле коллекций данных есть списки связей данных, такие как: тип токена функции и функция-делегат, строковое название функции и тип токена функции, строковое название константы и её значение, тип знака префиксной функции (минус и корень) и функция-делегат, символ знака и тип токена знака, тип токена знака и функция-делегат, тип токена знака и его приоритет.

Все функции, с которыми работает программа – параметрические, то есть от одного значения t вычисляются два: x и y . Но при работе с «обычными» функциями x всегда равен t .

Код главной функции модуля:

```
public static Func<double, (double x, double y)>? ParseMath(string input)
{
    #region parse
    var expression = input.StringToTokenTypes()
        .TokenTypesToBaseTokens()
        .GetExpressions()
        .SpaceToMultiply()
        .CountMathSigns()
        .WrapFunctions()
        .PrefixSignsToFunctions()
        .AbsToFunc()
        .WrapFunctions()
        .ResolveSignPriority()
        .PrefixSignsToFunctions()
        .CleanExpressions()
        ;
    #endregion

    var func = (expression as IOperand).GetFunc();
    Func<double, (double x, double y)>? output =
    func != null ? (arg => TupleFunc(arg, func) ?? (0, 0)) : null;
    output?.Invoke(0);
    return output;
}
```

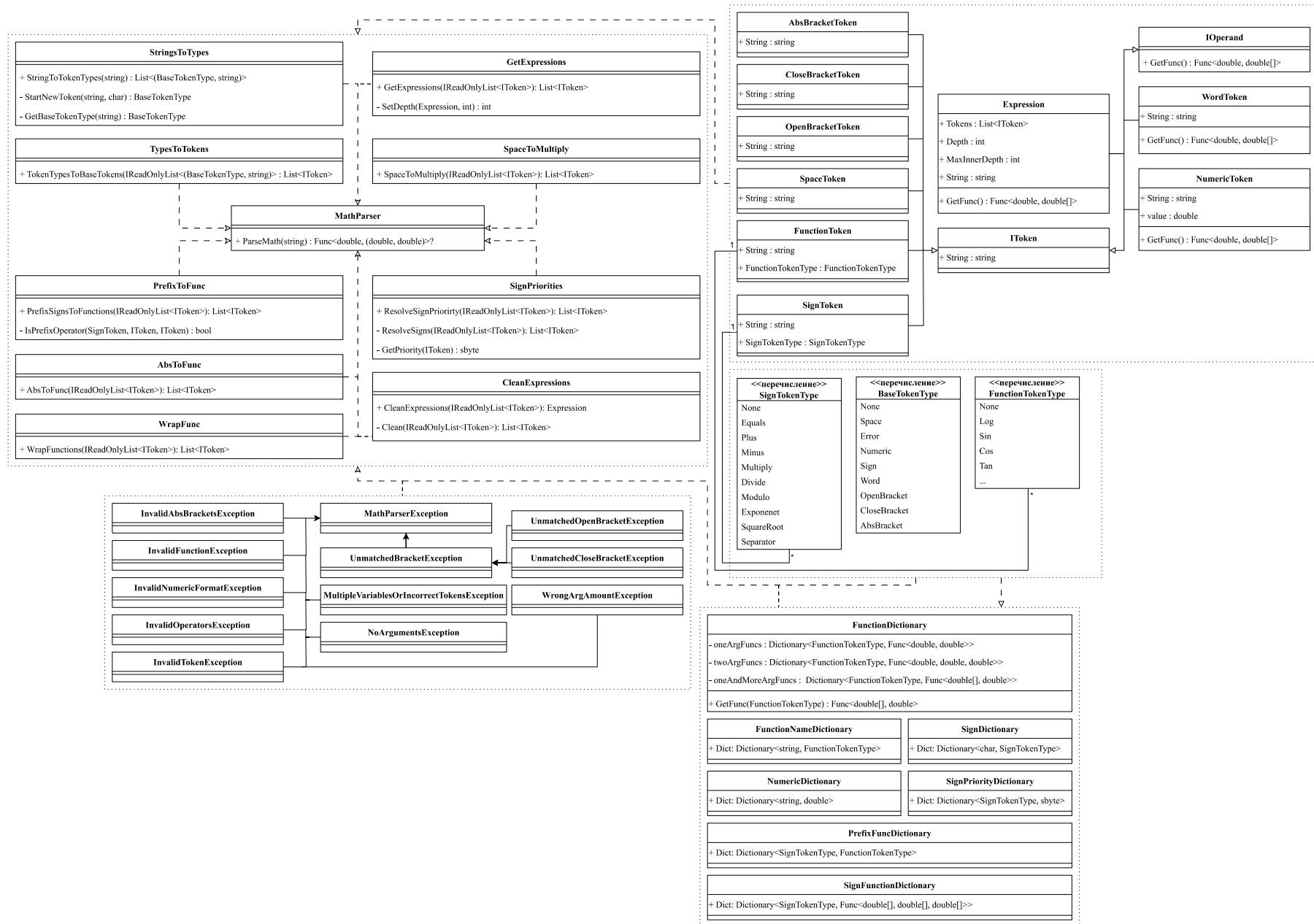


Рисунок 22 – Диаграмма классов модуля интерпретатора функций

Второстепенные модули

В проекте есть ещё несколько вспомогательных модулей: общего пользования и исключений. В модуле общего пользования определены интерфейсы для внедрения зависимостей и дополнительные классы, доступные во всей программе. У специфичных исключений программы так же есть свой модуль.

3.2. Реализация интерфейсов

Для создания интерфейсов в WPF используется язык разметки XAML. XAML – язык разметки, основанный на XML [8]. Он позволяет очень сильно разделить код логики времени выполнения от логики интерфейса.

Часть разметки списка полей ввода функций:

```
<StackPanel x:Name="FunctionInputContainer">
    <StackPanel>
        <Label Content="F0"/>
        <TextBox />
    </StackPanel>
    <StackPanel >
        <Label Content="F1"/>
        <TextBox />
    </StackPanel>
    <StackPanel>
        <Label Content="F2"/>
        <TextBox />
    </StackPanel>
</StackPanel>
```

Для реализации программы на нескольких языках используются файлы ресурсов, как на рис. 23. В них необходимый в программе текст прописывается на разных языках.

Name	Neutral Value	Neutral Comment	en	ru
AxisY	Y	Y as in Y-axis	Y	Y
CoordinateSystem	The coordinate system	The tag for a list of coord...	The coordinate system	Система координат
CoordSysCartesian	- Cartesian	Coordinate system - Cart...	- Cartesian	- Прямоугольная
CoordSysPolarFromPhi	- Polar from Φ	Coordinate system - Pola...	- Polar from Φ	- Полярная от Φ
CoordSysPolarFromR	- Polar from R	Coordinate system - Pola...	- Polar from R	- Полярная от R
Error	ERROR	Error message header	ERROR	ОШИБКА

Рисунок 23 – Файл ресурсов с разными языками

Использование ссылки на файл ресурсов в разметке:

```
<Window ...
    xmlns:comuse="clr-
namespace:GraThing.CommonUse.Properties;assembly=GraThing.CommonUse"
    ...>
    ...
    <Grid>
        <Label Content="{x:Static comuse:Resources.CoordinateSystem}"/>
    </Grid>
</Window>
```

На рис. 24 и 25 показан результат запуска с разным языком.

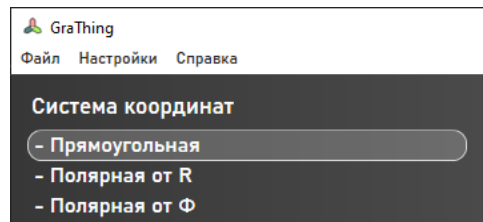


Рисунок 24 – Программа запущена на русском языке

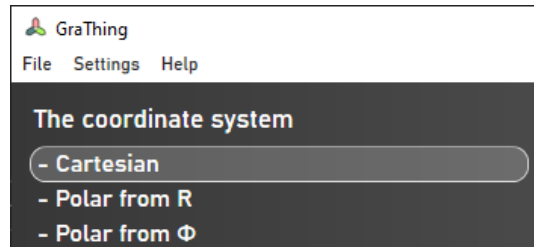


Рисунок 25 – Программа запущена на английском языке

Для того, чтобы элементы интерфейса одинаково выглядели во всех окнах, можно создать отдельный файл со стилями. Пример стиля объекта сетки с градиентным фоном:

```
<Style x:Key="GradientBG"
    TargetType="Grid">
    <Setter Property="Background">
        <Setter.Value>
            <LinearGradientBrush EndPoint="0.5,1"
                                StartPoint="0.5,0">
                <LinearGradientBrush.RelativeTransform>
                    <TransformGroup>
                        <ScaleTransform CenterY="0.5"
                                        CenterX="0.5" />
                        <SkewTransform CenterX="0.5"
                                    CenterY="0.5" />
                        <RotateTransform Angle="90"
                                       CenterX="0.5"
                                       CenterY="0.5" />
                        <TranslateTransform />
                    </TransformGroup>
                </LinearGradientBrush.RelativeTransform>
                <GradientStop Color="#FF4A4A4A" />
                <GradientStop Color="#FF373737"
                              Offset="1" />
            </LinearGradientBrush>
        </Setter.Value>
    </Setter>
</Style>
```

К файлам разметки окон он подключается следующим образом:

```
<Window.Resources>
    <ResourceDictionary>
        <ResourceDictionary.MergedDictionaries>
            <ResourceDictionary Source="/Styles.xaml" />
        </ResourceDictionary.MergedDictionaries>
    </ResourceDictionary>
</Window.Resources>
```

Также в рамках создания интерфейсов создаются пользовательские элементы управления. В данной программе это объект для выбора цвета: цветовое колесо с квадратом яркости и насыщенности или бегунков (см. п. 1.3.2.). Для его разметки также использовался язык XAML, а для реализации функций – привязки данных с конвертерами. Внутри основного блока находится несколько подблоков: колесо, квадрат, бегунки, и все они наследуют класс HSBCControl, который в свою очередь наследует класс UserControl. Использование наследования от UserControl – самый простой метод реализации пользовательских элементов управления в WPF [57].

Часть кода разметки блока выбора цвета:

```
<StackPanel HorizontalAlignment="Center">
    <StackPanel Orientation="Horizontal"
        HorizontalAlignment="Center">
        <inner:HSBShowColor x:Name="OldColor"
            CornerType="BowlOnLeft"/>
        <inner:HSBShowColor x:Name="ShowColor"
            CornerType="BowlOnRight"/>
    </StackPanel>
    <Grid Margin="3">
        <inner:HueWheel x:Name="HueWheel"/>
        <inner:SatBrtsSquare x:Name="SatBrtsSquare"/>
    </Grid>
    <TextBox Text="{Binding Hex, UpdateSourceTrigger=PropertyChanged}"/>
    <StackPanel Orientation="Horizontal">
        <TextBox Text="{Binding Hue, StringFormat={} {0:F1},
UpdateSourceTrigger=PropertyChanged}" />
        <inner:HueSlider x:Name="HueSlider"/>
    </StackPanel>
    <StackPanel Orientation="Horizontal">
        <TextBox Text="{Binding Sat, StringFormat={} {0:F1},
UpdateSourceTrigger=PropertyChanged}" />
        <inner:SatSlider x:Name="SatSlider"/>
    </StackPanel>
    <StackPanel Orientation="Horizontal">
        <TextBox Text="{Binding Brt, StringFormat={} {0:F1},
UpdateSourceTrigger=PropertyChanged}" />
        <inner:BrtSlider x:Name="BrtSlider"/>
    </StackPanel>
</StackPanel>
```

Часть кода разметки квадрата выбора яркости и насыщенности:

```
<Rectangle x:Name="HueBackground"
    Fill="{Binding Hue, Converter={StaticResource HueToSWMBConverter}}"/>
<Rectangle x:Name="ImgRect"
    Margin="-0.5"
    Stroke="DimGray">
    <Rectangle.Fill>
        <ImageBrush ImageSource="{StaticResource SatBrtsSquarePNG}" />
    </Rectangle.Fill>
</Rectangle>
```

Для выбора цвета используется цветовая модель HSB – Тон, Насыщенность, Яркость [27]. Она более понятна человеку, нежели система RGB – Красный, Зелёный, Синий, которая используется в мониторах, и соответственно, системе компьютера.

Разметка справки реализована при помощи HTML в стандартном элементе управления WebBrowser. В коде окна генерируется содержимое страницы и выводится в WebBrowser при помощи метода NavigateToString. Часть кода генерации HTML:

```
private void InitHtml()
{
    Wb1.NavigateToString(GetPart1());
    ...
}
private string GetPart1()
{
    var p = $"{@"
<h4>{Tab}{Hwr.Pl_01}
...
";
    return FormatHeadBody(p);
}
```

На рис. 26, 27 и 28 представлен итоговый внешний вид главного окна, окна настроек и окна справки.

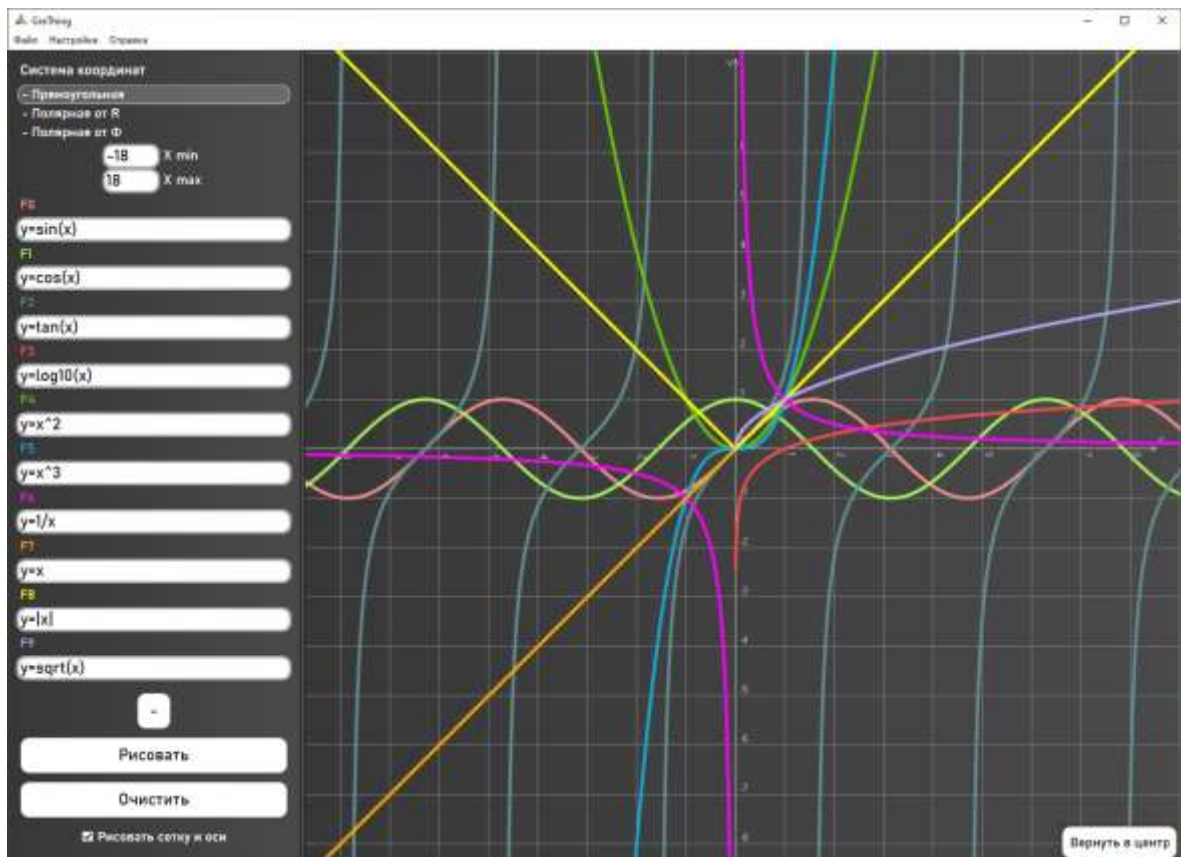


Рисунок 26 – Главное окно приложения

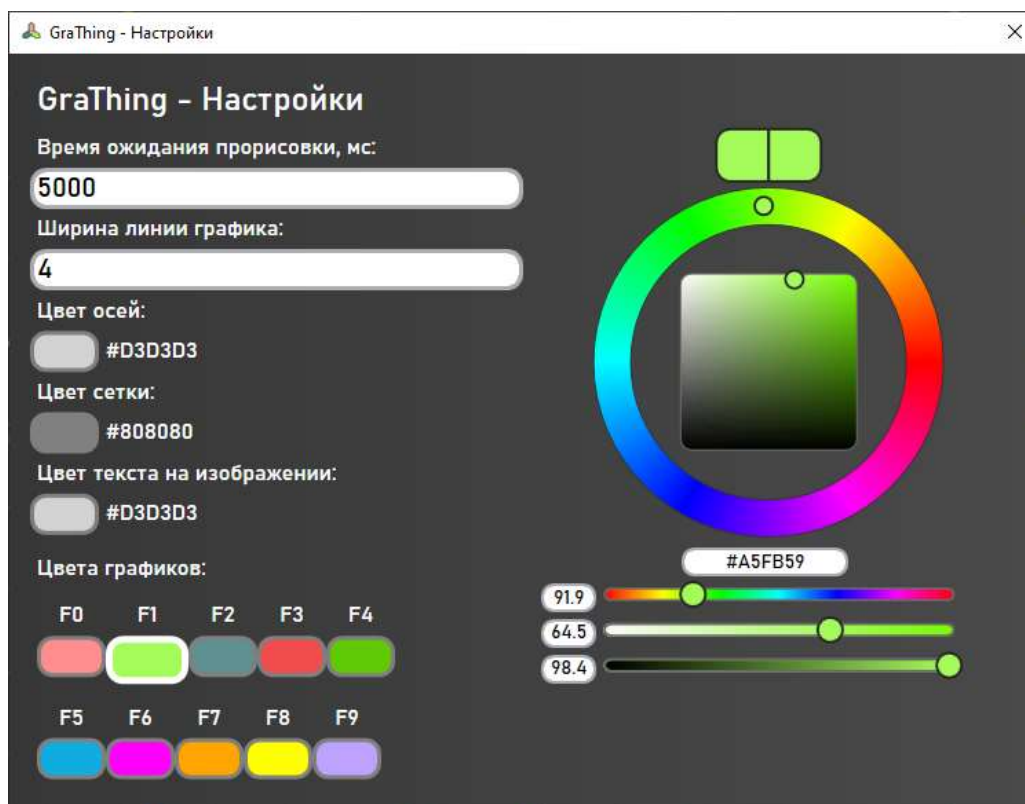


Рисунок 27 – Окно настроек

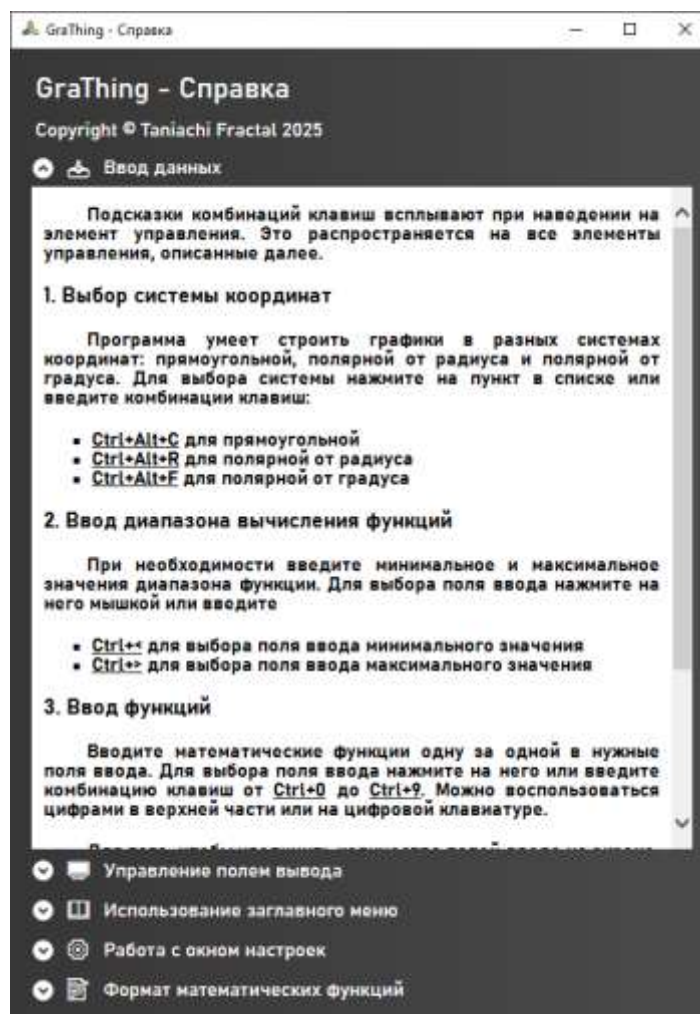


Рисунок 28 – Окно справки

3.3. Тестирование

Тестирование GraThing можно разделить на 2 категории: ручное-функциональное и автоматическое-модульное.

Формальные и чёткие данные и результат, необходимые для автоматического тестирования, есть только у модуля интерпретатора. Он получает на вход строку и возвращает функцию-делегат, которую можно вызвать и также протестировать.

Модуль визуализации возвращает изображения, а их корректность невозможно проверить формально. Модули интерфейсов и основной только получают и выдают данные, запрашивая обработку у других. Поэтому, далее будет описано модульное тестирование только для интерпретатора.

Тестирование интерпретатора

Для модульного тестирования необходимо создать отдельный класс, к которому подключено пространство имён XUnit.

Каждый отдельный модульный тест – метод, помеченный атрибутом [Theory]. У этого метода есть аргументы, данные для них берутся из атрибутов [InlineData()]:

```
[Theory]
[InlineData("sin(x)", 0)]
[InlineData("10E+4", 10E+4)]
[InlineData("qwerty=2^2", 4)]
public void ValidOnePartExpressionsWork_at0(string expression, double expected)
{
    // Act
    var result = MathParser.ParseMath(expression)?.Invoke(0);
    // Assert
    Assert.Equal(expected, result?.y);
}
```

Таким образом, можно проверить огромное количество разных возможных входных данных, написав лишь немного кода.

Были разработаны следующие тесты: на каждое исключение интерпретатора (см. п. 3.2.), тесты обычных правильных функций и тесты параметрических функций.

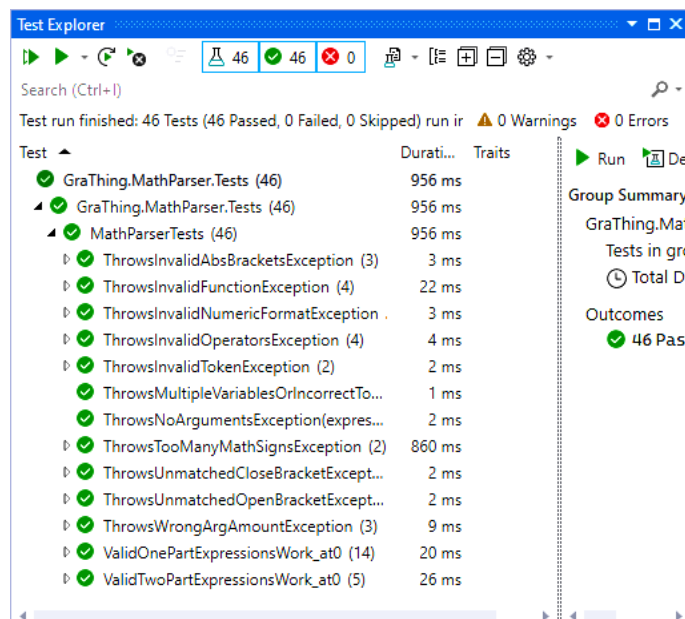


Рисунок 29 – Окно с успешными результатами модульных тестов

Ручное тестирование

Далее описаны протоколы тестирования вручную.

Таблица 10 – Тестовый пример №1

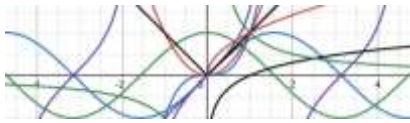
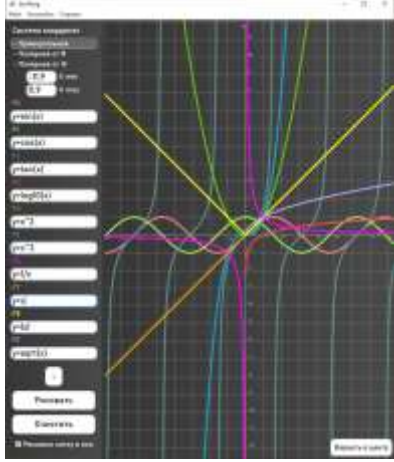
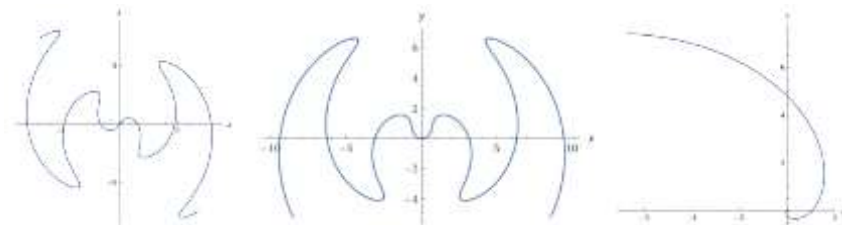
Приоритет тестирования	Высокий
Название	Функции из файла
Краткое изложение	Загрузить 10 функций из файла
Этапы теста	<ol style="list-style-type: none"> 1) Раскрыть меню «Файл»; 2) Выбрать «Загрузить функции из файла»; 3) Выбрать файл и открыть его; 4) Нажать «Рисовать».
Тестовые данные	<p>Текстовый .ТХТ файл, его содержимое:</p> <p>«$y=\sin(x)$ $y=\cos(x)$ $y=\tan(x)$ $y=\log_{10}(x)$ $y=x^2$ $y=x^3$ $y=1/x$ $y=x$ $y= x$ $y=\sqrt{x}$»</p>
Ожидаемый результат	<p>Раскрыты все 10 полей ввода, в них загружены корректные строки. Графики идентичны данному изображению из Desmos:</p> 
Фактический результат	
Статус	Зачёт
Предварительное условие	Открыто главное окно, раскрыто 1 поле ввода, выбрана прямоугольная система координат

Таблица 11 – Тестовый пример №2

Приоритет тестирования	Высокий
Название	Полярная система координат от градуса

Краткое изложение	Ввести 3 функции и выбрать полярную систему координат от градуса.
Этапы теста	1) Раскрыть 3 поля ввода; 2) Ввести 3 функции; 3) Выбрать полярную систему координат от градуса; 4) Нажать «Рисовать».
Тестовые данные	«sin x», «cos x», «tan x».
Ожидаемый результат	Построенные графики идентичны данному изображению из Desmos: 
Фактический результат	
Статус	Зачёт
Предварительное условие	Открыто главное окно, раскрыто 1 поле ввода

Таблица 12 – Тестовый пример №3

Приоритет тестирования	Высокий
Название	Полярная система координат от радиуса
Краткое изложение	Ввести 3 функции и выбрать полярную систему координат от радиуса.
Этапы теста	1) Раскрыть 3 поля ввода; 2) Ввести 3 функции; 3) Выбрать полярную систему координат от радиуса; 4) Нажать «Рисовать».
Тестовые данные	«sin x», «cos x», «ln x».
Ожидаемый результат	Построенные графики идентичны данным изображениям из WolframAlpha: 

Фактический результат	
Статус	Зачёт
Предварительное условие	Открыто главное окно, раскрыто 1 поле ввода

Таблица 13 – Тестовый пример №4

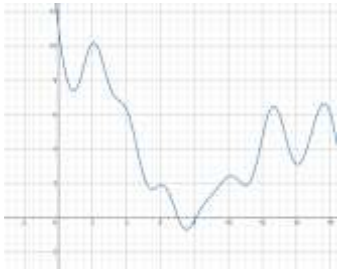

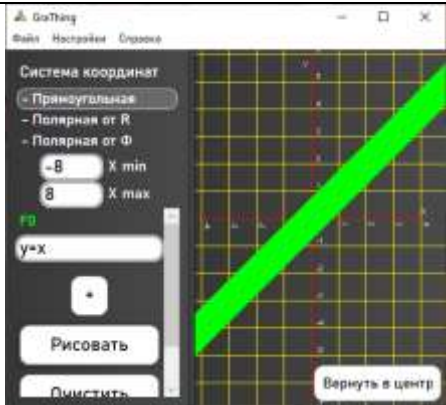
Приоритет тестирования	Высокий
Название	Сложная функция и экспорт
Краткое изложение	Ввести сложную и длинную функцию и экспортировать ее
Этапы теста	<ol style="list-style-type: none"> 1) Ввести 3 функции; 2) Нажать «Рисовать»; 3) Убрать галочку с «Рисовать сетку и оси»; 4) Раскрыть меню «Файл»; 5) Выбрать «Сохранить графики как...»; 6) Сохранить файл.
Тестовые данные	$\cos(3x) * e^{(-0.1x)} + \cos(2x) * (x^2 / 100) - 5 * \sin(x/3) + 10 * (0.5)^x * (1 + x^2)$
Ожидаемый результат	<p>Получен графический файл. На изображении нет координатной сетки; Построенный график идентичен данному изображению из Desmos:</p> 
Фактический результат	<p>Получен файл с содержимым:</p> 
Статус	Зачёт
Предварительное условие	Открыто главное окно, раскрыто 1 поле ввода, выбрана прямоугольная система координат

Таблица 14 – Тестовый пример №5

Приоритет тестирования	Высокий
Название	Сохранение настроек
Краткое изложение	Изменить настройки и перезапустить программу.
Этапы теста	<ol style="list-style-type: none"> 1) Открыть окно настроек; 2) Ввести данные; 3) Закрыть окно настроек; 4) Завершить программу; 5) Запустить программу заново; 6) Ввести функцию; 7) Нажать «Рисовать».
Тестовые данные	Ширина линии – 33; Цвет осей – «#FF0000»; Цвет сетки – «#FFFF00»; Цвет графика F0 – «#00FF00»; Функция – « $y=x$ »
Ожидаемый результат	После перезагрузки приложения цвет осей – красный, а цвет сетки – жёлтый. График нарисован толстой линией зелёного цвета. Подпись у поля ввода F0 тоже зелёная.
Фактический результат	
Статус	Зачёт
Предварительное условие	Открыто главное окно

4 Руководство пользователя

4.1. Описание установки

Условия для установки:

ОС Windows 10 1709 или более поздняя, 2 Гб Оперативной памяти, 128 Мб Графической памяти, 20 Мб свободного дискового пространства, Интернет-подключение (только для установки).

Шаги установки:

- 1) Зайдите электронный ресурс, где выкладываются выпуски программы по ссылке <https://github.com/TaniachiFractal/GraThing/releases>;
- 2) Скачайте последнюю версию приложения, архив с названием «GraThing.zip»;

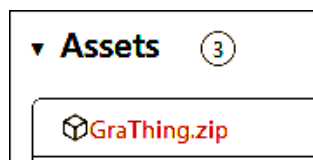


Рисунок 30 – Архив с программой на странице выпусков

- 3) Распакуйте архив любым удобным способом в любую подходящую пустую папку. Рекомендуется создать папку с названием «GraThing» в главном каталоге файлов пользователя и распаковать архив туда;
- 4) Перейдя в папку «GraThing», найдите файл GraThing.exe. Сделайте клик правой кнопкой мыши по нему и выберите «Копировать»;
- 5) Откройте рабочий стол, нажмите по нему правой кнопкой мыши и выберите «Вставить ярлык»;
- 6) Кликните правой кнопкой мыши по ярлыку, выберите «Переименовать» и задайте ему название «GraThing».

Проверьте работу программы: сделайте двойной клик по её ярлыку. Если происходят события, описанные в п. 4.2 или программа успешно запустилась, всё хорошо, установка завершена. Но если появилось сообщение об ошибке, как на рис. 31, это значит, что на вашем устройстве не установлен .NET Framework версии 4.8. Нажмите «Да», чтобы перейти на официальные ресурсы Microsoft с информацией о том, как обновить систему до нужной версии.

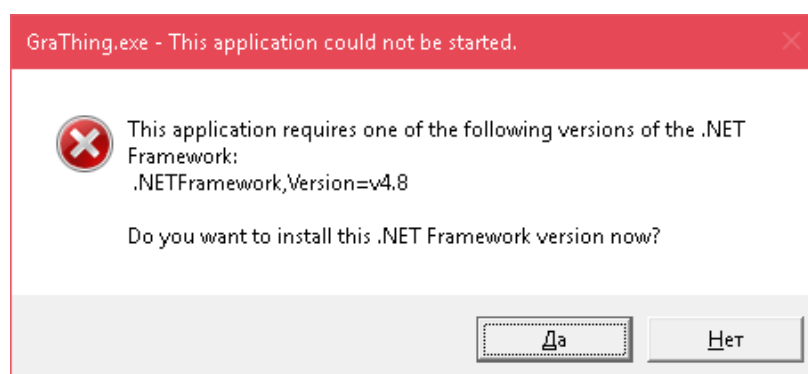


Рисунок 31 – Сообщение об ошибке «Не установлен .NET Framework 4.8»

4.2. Описание запуска

Для запуска приложения сделайте двойной клик по его ярлыку.



Рисунок 32 – Ярлык GraThing на рабочем столе

При первом запуске будет выведено сообщение об отсутствии файла. После этого будут загружены стандартные настройки и файл с ними будет сохранён на устройство. Окно сообщения показано на рис. 33.

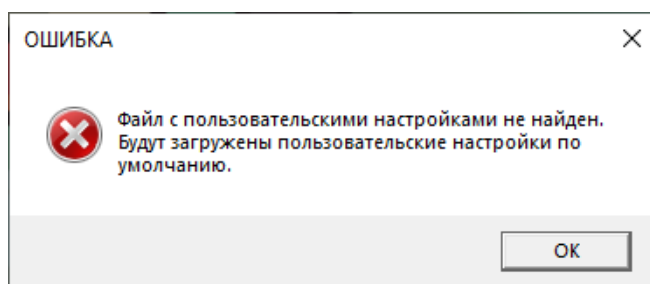


Рисунок 33 – Сообщение об ошибке «Файл настроек не найден»

Если программа не запускается, возможно, вы удалили один или несколько файлов из папки, куда вы установили программу. Попробуйте найти и вернуть файлы или выполнить установку заново.

4.3. Инструкции по работе

Перед началом работы ознакомьтесь со справкой в самой программе. Она идентична этому руководству.

4.3.1. Ввод данных

1) Выбор системы координат

Программа умеет строить графики в разных системах координат: прямоугольной, полярной от радиуса и полярной от градуса. Для выбора системы нажмите на пункт в списке как на рис. 34 или введите комбинации клавиш:

- **Ctrl+Alt+C** для прямоугольной;
- **Ctrl+Alt+R** для полярной от радиуса (r);
- **Ctrl+Alt+F** для полярной от градуса (φ).

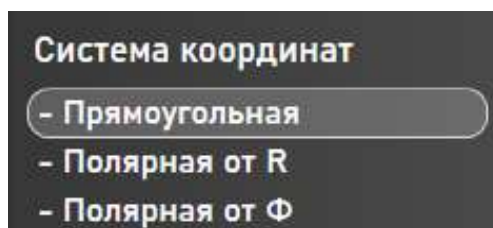


Рисунок 34 – Выбор системы координат

Подсказки комбинаций клавиш всплывают при наведении на элемент управления, как показано на рис. 35. Это распространяется на все элементы управления, описанные далее.

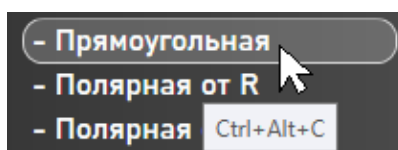


Рисунок 35 – Подсказка при наведении на элемент

2) Ввод диапазона вычисления функций

При необходимости введите минимальное и максимальное значения диапазона функции. Для выбора поля ввода нажмите на него мышкой или введите

- Ctrl + < для выбора поля ввода минимального значения;
- Ctrl + > для выбора поля ввода максимального значения.

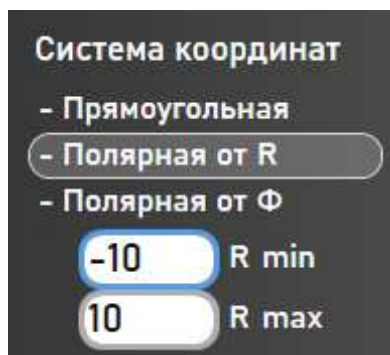


Рисунок 36 – Ввод диапазона вычисления функций

3) Ввод функций

Вводите математические функции одну за одной в нужные поля ввода. Для выбора поля ввода нажмите на него или введите комбинацию клавиш от Ctrl+0 до Ctrl+9. Можно воспользоваться цифрами в верхней части или на цифровой клавиатуре.

Для того, чтобы увеличить количество полей ввода на экране, нажмите на кнопку плюс или используйте Ctrl+Plus (клавиша Плюс в верхней части или на цифровой клавиатуре). Аналогично, для уменьшения нажмите кнопку минус или воспользуйтесь Ctrl+Minus. Поля ввода и экранные кнопки показаны на рис. 37.

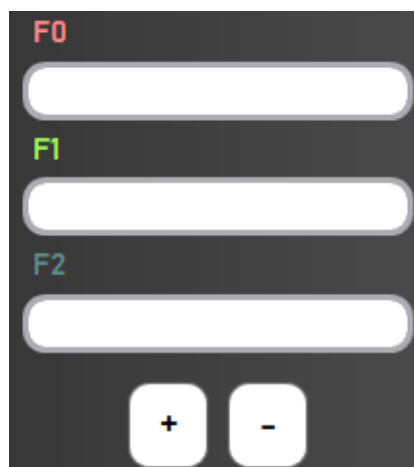


Рисунок 37 – Поля ввода функций и экранные кнопки.

Правильный формат математических функций будет описан в п. 4.3.5.

4) Запуск построения графиков

Если вы ввели все нужные функции, для запуска построения нажмите кнопку «Рисовать», **Ctrl+Enter** или **Ctrl+D**. Если вы хотите очистить все поля ввода, нажмите «Очистить» или **Ctrl+Shift+C**.

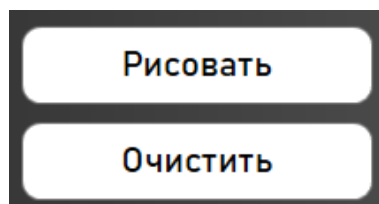


Рисунок 38 – Кнопки «Рисовать» и «Очистить»

4.3.2. Управление полем вывода

Для изменения масштаба координатной сетки покрутите колёсико компьютерной мыши вверх или вниз для увеличения и уменьшения. Аналогично комбинациям клавиш из шага 3 в п. 4.3.1., можно воспользоваться **Alt+Plus** и **Alt+Minus**. Масштабирование приведёт к сбросу диапазона вычисления функций.

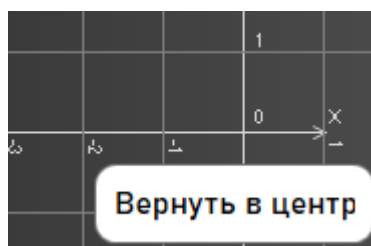


Рисунок 39 – Кнопка «Вернуть в центр»

Центр координатной сетки можно сдвинуть, зажав правую кнопку мыши. Чтобы вернуть его в начальное положение, нажмите «Вернуть в центр» или **Ctrl+Shift+R**.

Для изменения размера поля, зажмите левой кнопкой мыши разделитель экрана между блоками ввода и вывода, как на рис. 40, либо поменяйте размер окна приложения. Изменение размеров поля приведёт к возврату координатной сетки в начальное положение и также к сбросу диапазона вычисления функций.

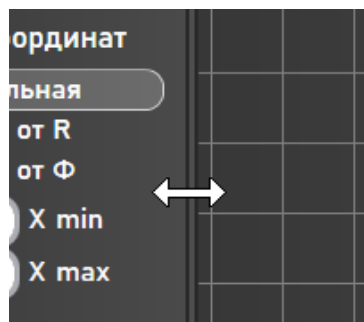


Рисунок 40 – Разделитель окна

Кроме того, можно скрыть координатную сетку и оси. Снимите галочку с «Рисовать сетку и оси» или нажмите **Alt+Shift+A**. Чтобы вернуть их, верните галочку или введите ту же комбинацию клавиш заново.



Рисунок 41 – Галочка «Рисовать сетку и оси»

4.3.3. Использование заглавного меню

В верхней части окна приложения есть меню с пунктами «Файл», «Настройки» и «Справка». При нажатии на «Файл» или «Настройки» будут выведены списки опций, которые могут быть вызваны комбинациями клавиш.

Опции «Файл»

- Загрузить функции из файла **Ctrl+O**

В программе можно открыть любой текстовый файл в качестве списка математических функций. В открывшемся окне файловой системы выберите .TXT файл, в котором находится список функций, и нажмите «Открыть». Будут загружены первые 9 строк файла.

- Сохранить функции как... **Ctrl+S**

Список функций из полей ввода можно сохранить как текстовый файл. В открывшемся окне файловой системы выберите нужное расположение файла, в который будет сохранён список функций, которые на данный момент написаны в полях ввода. Файлу будет задано стандартное название в формате «GraThing Функции 2000-01-01-000000.txt». Его можно поменять на другое. Для завершения нажмите «Сохранить».

- Сохранить графики как... **Ctrl+Shift+S**

Поле для рисования можно сохранить как картинку. В открывшемся окне файловой системы выберите нужное расположение файла. Файлу будет задано стандартное название в формате «GraThing Картинка 2000-01-01-000000.png». Его можно поменять на другое. Для завершения нажмите «Сохранить».

- Выход **Alt+F4**

При выборе данной опции программа завершит работу.

При работе с файлами будет создана папка «GraThing» в Документах пользователя, в которую будет предложено сохранить файл или загрузить его из неё при открытии окна файловой системы.

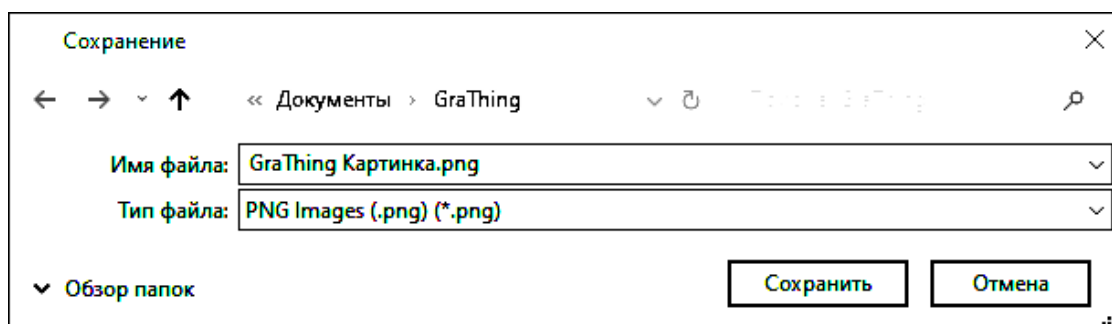


Рисунок 42 – Сохранение в папку «GraThing» в Документах пользователя

Опции «Настройки»:

- Открыть настройки **Ctrl+;** (Точка с запятой – клавиша Ж)

Будет открыто окно пользовательских настроек. Подробнее работа с ним будет описана в п. 4.3.4.

- Сбросить настройки **Ctrl+Shift+D**

В появившемся предупреждении о сбросе нажмите «ОК». После этого настройки пользователя вернутся к первоначальному виду.

- Импорт настроек **Ctrl+I**

Если у вас есть заранее экспортированный файл настроек, выберите его в открывшемся окне файловой системы и нажмите «Открыть». Если файл корректный, настройки будут загружены из него.

- Экспорт настроек **Ctrl+E**.

Настройки можно экспортировать. В открывшемся окне файловой системы выберите нужное расположение файла. Файлу будет задано стандартное название в формате «GraThing Настройки Пользователя 2000-01-01-000000.png». Его можно поменять на другое. Для завершения нажмите «Сохранить».

При нажатии пункта меню «Справка» или **Alt+F1** будет открыто окно справки с информацией, идентичной данному руководству пользователя.

4.3.4. Работа с окном настроек

В окне настроек можно изменить время ожидания прорисовки в миллисекундах, задать ширину линии графика и настроить цвета на поле для рисования.

Для выбора поля ввода времени ожидания используйте **Ctrl+T**, ширины линии – **Ctrl+W**.

Изменение цветов

Для изменения цвета графика, сетки и т.д. нажмите на соответствующий ему прямоугольник, либо при помощи **Ctrl+R** выберите первый прямоугольник с цветом и выбирайте нужный при помощи клавиш-стрелочек.

После того, как вы выберете тот или иной прямоугольник, его цвет отобразится на элементе выбора цвета. Двигайте бегунки на нём, чтобы изменять Тон, Насыщенность и

Яркость цвета [27]. Эти значения можно задать вручную, введя значения в поля ввода рядом с каждым бегунком, или удерживая

- **Ctrl+H+Plus** (см. шаг 3 п. 4.3.1.) или **Ctrl+H+Minus** для увеличения или уменьшения Тона;
- **Ctrl+S+Plus** или **Ctrl+S+Minus** для Насыщенности;
- **Ctrl+B+Plus** или **Ctrl+B+Minus** для Яркости.

Также можно ввести значение цвета в HEX формате [39] в поле ввода под цветовым кругом.

Чтобы сохранить изменения, закройте окно настроек, нажав Alt+F4.

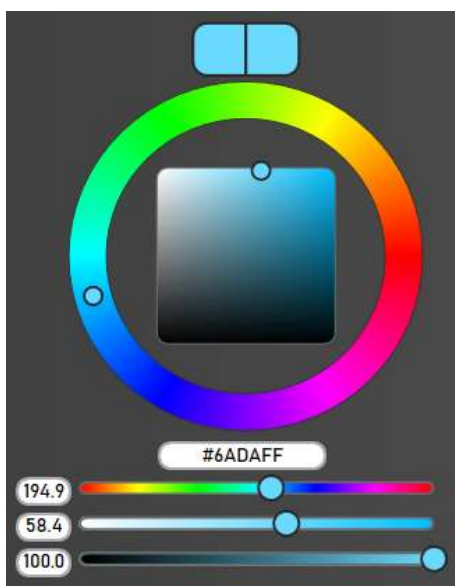


Рисунок 43 – Элемент выбора цвета с цветовым колесом, квадратом яркости и насыщенности и бегунками

4.3.5. Формат математических функций

В рамках данной программы, математическая функция – некоторое правило, по которому вычисляются координаты точек при подстановке множества значений переменных. Все графики в программе – двумерные. GraThing составляет множество точек. Их координатами являются значения полученное и переменной. Если же речь идёт о параметрической [7] записи, то координаты точек – это два полученных значения. В любом случае, приложение соединяет эти точки и получается готовый построенный график.

В GraThing доступен упрощённый ввод математических функций. Каждая функция помещается ровно в 1 строку. В начале не обязательно писать $y = \dots$, программа рассматривает только правую часть функции. Если левой нет – подразумевается, что написана именно правая часть.

Это значит, что записи

$$y = x + 1$$

(3)

и

$$x + 1$$

(4)

эквивалентны.

У переменных могут быть абсолютно любые названия, только если это не зарезервированный токен; также нельзя использовать специальные символы (см. далее). К примеру, можно написать функции

Доброе=Утро

(5)

или

Здравствуйте!

(6)

и они будут равноценны

$y=x$

(7)

В левой части функции, перед знаком равенства « = », можно написать любой текст. Он будет проигнорирован.

В правой части может быть только одна переменная с одним названием. К примеру,

val + val · 2

(8)

но не

~~*val + x · 2*~~

(9)

Программа не чувствительна к регистру, так что все функции и переменные можно писать как заглавными, так и строчными буквами.

Токены – составные элементы математической функции. Ими являются названия базовых функций, переменные, числовые константы, математические знаки и скобки. Список доступных токенов:

– Операнды

Операнд – это токен, чьё значение можно вычислить. К ним относятся переменные, константы и группы токенов, такие как выражения в скобках и базовые функции с параметрами.

Переменная – это величина, принимающая множество значений [30] из диапазона вычисления (см. п. 4.3.1). Она может включать в себя буквы, не специальные символы и цифры. Цифра не может быть первым символом в обозначении переменной.

Константа – это постоянная величина. Она может записываться несколькими способами:

– Специальная константа:

- Число пи (3,141592653...) [35]: «pi», «π», «пи»;
- Число е (2,718281828...) [22]: «e» (латиницей или кириллицей);
- Число фи (1,618033988...) [36]: «phi», «φ», «фи»;
- Целое число [33]: Набор цифр от 0 до 9 (123; 45678 и пр.);
- Десятичная дробь [17]: Набор цифр от 0 до 9 с разделителем запятой « , » или точкой « . » (123,456; 78.9 и пр.);
- Экспоненциальная запись [37]: в начале стоит целое число или десятичная дробь, затем символ «Е» (латиницей или кириллицей), далее плюс « + » или минус « - » (плюс можно опустить), и в конце целое число. Примеры: 123E4; 12,3E+4; 12.3E-4.

– Отрицательное число [29]: перед любой из предыдущих записей поставьте знак минуса « - » (примечание: в действительности, это префиксный знак минуса, см. далее).

Если два отдельных операнда стоят рядом друг с другом, то подразумевается, что между ними стоит знак умножения.

Пример функции со всеми четырьмя типами операндов:

$$x + \pi + 0,1(x - 8) + \cos x$$

(10)

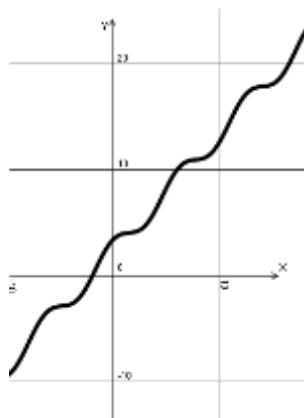


Рисунок 44 – График функции (10)

– Разделитель

Разделитель – это специальный знак, используемый в GraThing для деления половин параметрической [7] записи и для разделения аргументов базовых функций. Он обозначается символом точка с запятой « ; ».

Пример записи аргументов функции:

$$\max(1;x)$$

(11)

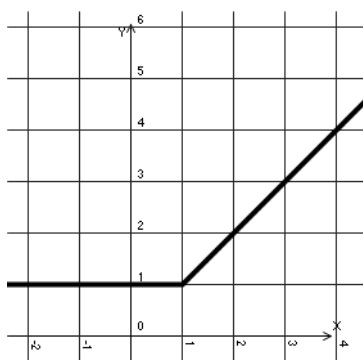


Рисунок 45 – График функции (11)

Пример параметрической записи функции:

$$x=t \cdot t; y=t$$

(12)

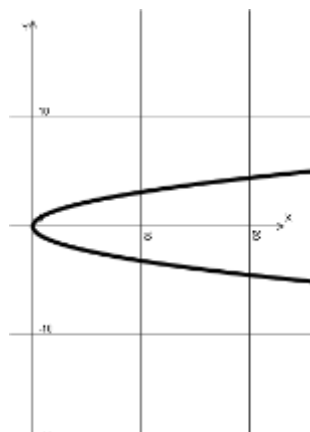


Рисунок 46 – График функции (12)

В первой половине параметрической записи вычисляется горизонтальная координата, а во второй – вертикальная.

– Скобки

Доступны круглые « () » скобки для задания приоритета операций и для записи аргументов функции.

Пример задания приоритета:

$$2 \cdot (x+1)$$

(13)

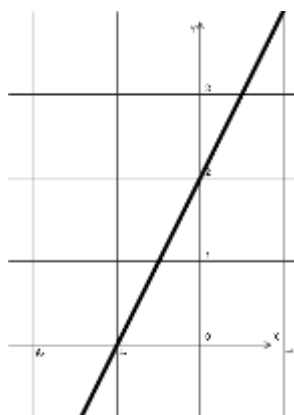


Рисунок 47 – График функции (13)

Пример записи аргументов функции:

$$3 \operatorname{sinc}(x)$$

(14)

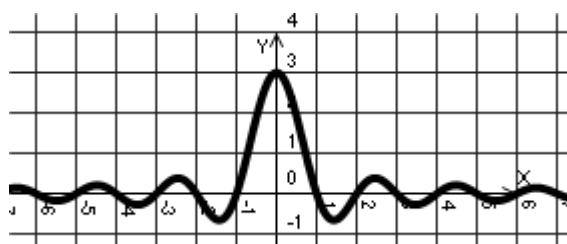


Рисунок 48 – График функции (14)

Также есть прямые скобки, используемые для записи модуля. Они могут быть символами «|» или «|» (это два разных символа).

Пример записи модуля:

$$y=|x|$$

(15)

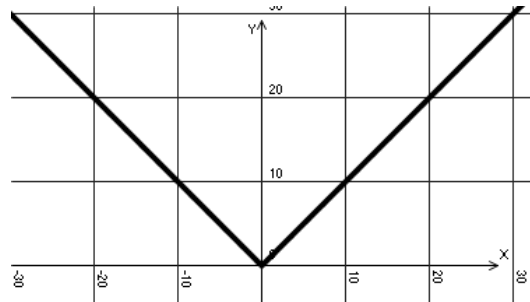


Рисунок 49 – График функции (15)

Другие типы скобок в программе не предусмотрены и не считаются специальными символами.

– Базовые функции

В программе доступен набор базовых математических функций. Все они возвращают только одно значение, но в качестве аргументов получают один и более операндов (см. выше).

– Функции одного аргумента:

- Логарифм по основанию 10 «**log10**»;
- Логарифм по основанию 2 «**log2**»;
- Натуральный логарифм «**ln**»;
- Синус «**sin**»;
- Косинус «**cos**»;
- Тангенс «**tg**» или «**tan**»;
- Косеканс «**csc**»;
- Секанс «**sec**»;
- Котангенс «**cot**»;
- Арксинус «**asin**» или «**arcsin**»;
- Арккосинус «**acos**» или «**arccos**»;
- Арктангенс «**atg**», «**arctg**», «**atan**» или «**arctan**»;
- Кардинальный синус «**sinc**»;
- Гиперболический синус «**sinh**»;
- Гиперболический косинус «**cosh**»;
- Гиперболический тангенс «**tgh**» или «**tanh**»;
- Округление вверх «**ceiling**»;
- Округление вниз «**floor**»;
- Модуль «**abs**»;
- Квадратный корень «**sqrt**»;
- Кубический корень «**cbrt**»;
- Отрицательное значение «**neg**»;
- Экспонента (число E в степени X) «**exp**».

Функции одного аргумента могут записываться как в формате **Функция(Аргумент)**, так и в формате без скобок **Функция Аргумент**, если аргумент – переменная или константа.

– Функции двух аргументов:

- Логарифм числа X по основанию A «**log(a; x)**»;
- Корень числа X степени A «**root(a; x)**».

Аргументы этих функций должны стоять в строгом порядке.

– Функции множества аргументов:

- Максимальное число из множества {a1, a2, a3...} «**max(1; 2; x)**»;
- Минимальное число из множества {a1, a2, a3...} «**min(1; 2; x)**»;
- Среднее арифметическое чисел из множества {a1, a2, a3...} «**avg(1; 2; x)**».

Не рекомендуется задавать более 5 аргументов. Общий лимит математических знаков (в т.ч. разделителей аргументов) в функции – 20.

При записи в скобках и между разделителями аргументами могут быть не только операнды, но и подфункции без скобок.

Пример использования всех 3 типов базовых функций:

$$avg(root(cot x; 8); 1+x; x)$$

(16)

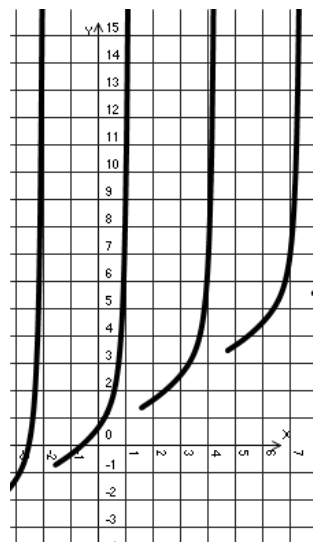


Рисунок 50 – График функции (16)

– Префиксные знаки

В GraThing предусмотрено два префиксных знака: минус «-» и квадратный корень « $\sqrt{\quad}$ ». Они работают так же, как функции одного аргумента, то есть после них должен стоять один операнд в формате **Знак(Аргумент)** или **Знак Аргумент**.

Префиксный знак минуса вычисляет отрицательное значение, а корня – квадратный корень числа. Их полные эквиваленты – функции «**neg**» и «**sqrt**» соответственно.

Пример использования префиксных знаков:

$$\sqrt{-x}$$

(17)



Рисунок 51 – График функции (17)

– Операторы

Операторы – это математические знаки. Они должны стоять между двух операндов в формате **Операнд Знак Операнд**.

Знак равенства « = » – особый оператор. Как говорилось ранее, он используется для разделения левой и правой части функции. Соответственно, в записи формулы может быть не больше двух знаков равенства: один в обычной записи или по одному в каждой половинке параметрической.

У всех остальных операторов есть приоритет вычисления.

- Знаки 1 приоритета:
 - Возведение в степень « ^ »;
- 2 приоритета:
 - Умножение « * », « × » или « · »;
 - Деление « / », « \ », « : » или « ÷ »;
 - Вычисление остатка от деления « % »;
- 3 приоритета:
 - Сложение « + »;
 - Вычитание « - ».

Технически, разделитель « ; » – тоже математический знак самого низкого приоритета, он возвращает массив операндов.

– Пробел

Пробел – это специальный токен. Любое количество пробелов подряд будет считаться за один. Если пробел стоит между двух операндов, то он будет обработан как оператор умножения. Если пробел стоит в других местах, он будет полностью проигнорирован.

Пример использования пробела как оператора умножения:

$$x \ x + 3x$$

(18)

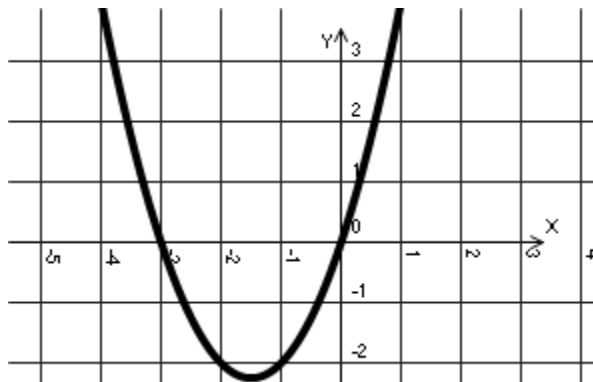


Рисунок 52 – График функции (18)

Полный список специальных знаков: « = », « ; », « (», «) », « | », « | », « ^ », « * », « × », « · », « / », « \ », « : », « ÷ », « % », « + », « - », « . », « , ».

В GraThing можно создавать довольно красивые изображения, если отключить рисование сетки и осей. Вот несколько примеров:

– Абстрактный цветок:

- Система координат: Полярная от радиуса;
- Диапазон: от -7 до 7;
- Список функций:
 - $-\cos t; \min(\sin t; \tan t)+1$
 - $-\cos t; \min(\sin t; \tan t)+2$
 - $-\cos t; \min(\sin t; \tan t)+3$
 - $-\cos t; \min(\sin t; \tan t)+4$
 - $-\cos t; \min(\sin t; \tan t)+5$
 - $-\cos t; \min(\sin t; \tan t)+6$
 - $-\cos t; \min(\sin t; \tan t)+8$
 - $-\cos t; \min(\sin t; \tan t)+7$
 - $-\cos t; \min(\sin t; \tan t)+9$
 - $-\cos t; \min(\sin t; \tan t)+10$



Рисунок 53 – Набор функций «Абстрактный цветок»

– Павлин:

- Система координат: Полярная от градуса;
- Диапазон: от -6 до 6;
- Список функций:
 - $(1+\sin t)*(1+0.9*\cos(16*t))*(1+0.1*\cos(24*t))*(0.97+0.05*\cos(200*t))$
 - $(1+\sin t)*(1+0.9*\cos(16*t))*(1+0.1*\cos(24*t))*(0.7+0.05*\cos(200*t))$
 - $\sin t$
 - $0.5\sin t; 0.5\cos(t+1)$
 - $0.1\sin t+17; \cos t+1.7$
 - $0.2\sin t+9; 0.2\cos t+2.5$
 - $0.55\pi; 2.5$
 - $0.1\sin t+16.6; 0.7\cos t * \sin(0.1t)+3$



Рисунок 54 – Набор функций «Павлин»

– Бабочка:

- Система координат: Прямоугольная;
- Диапазон: от -36 до 36;
- Список функций:
 - $\sin t(e^{\cos t}-3\cos(4t)+(\sin(1/12t)^5)); \cos t(e^{\cos t}-3\cos(4t)+(\sin(1/12t)^5))$
 - $\sin t(e^{\cos t}-2\cos(4t)+(\sin(1/12t)^5)); \cos t(e^{\cos t}-2\cos(4t)+(\sin(1/12t)^5))$
 - $\arccos t - 0.5\pi$
 - $a \sin t$
 - $\sin(t); -\cos(2t)+3$
 - $0.6 \sin t; 0.6 \cos t+2$
 - $0.3 \sin t; 1.3 \cos t$
 - $a \sin t; 0$
 - $0.3; 2$
 - $-0.3; 2$

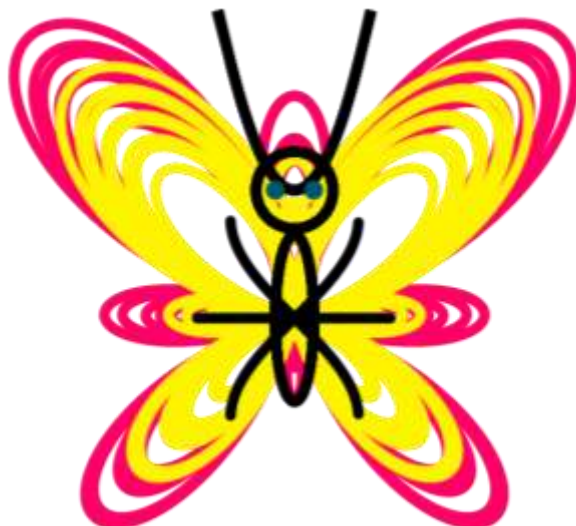


Рисунок 55 – Набор функций «Бабочка»

4.4. Сообщения пользователю

4.4.1. Ошибки ввода функций

Ошибки во введенных функциях выводятся списком во всплывающем сообщении. В каждой строчке стоит номер функции и примерное описание её проблемы. Даже если в функции есть несколько проблем, за раз будет выведена только одна ошибка.

Список ошибок

Сообщение:

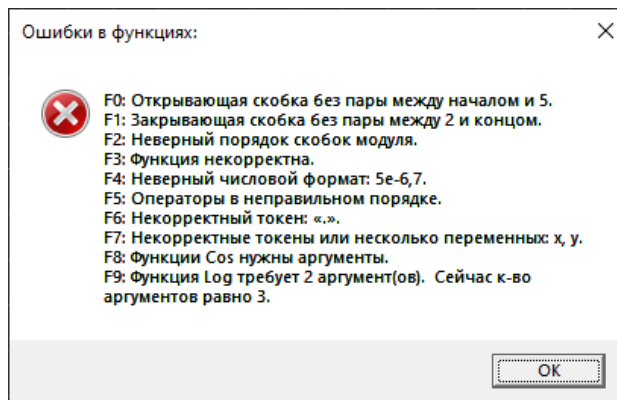


Рисунок 56 – Пример сообщения со списком ошибок

Действия пользователя: Попробуйте исправить ошибки. При необходимости прочитайте справку заново.

Скобка без пары

Сообщения:

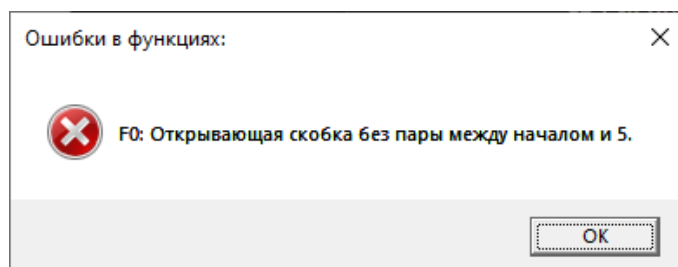


Рисунок 57 – Пример сообщения об ошибке «Открывающая скобка без пары»

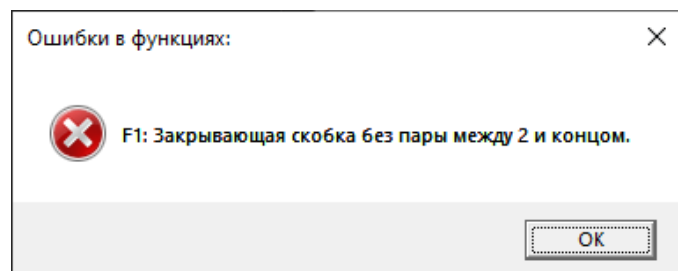


Рисунок 58 – Пример сообщения об ошибке «Закрывающая скобка без пары»

Действия пользователя: Постарайтесь найти место, где пропущена скобка. В описании ошибки будет указаны тип скобки без пары и два токена, которые стоят рядом с ней. Если скобка стоит в самом начале, будет написано «...между началом и..», а если в самом конце – «...между ... и концом.».

Неверный порядок скобок модуля

Сообщение:

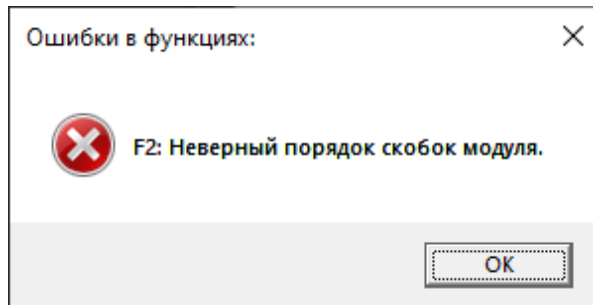


Рисунок 59 – Пример сообщения об ошибке «Неверный порядок скобок модуля»

Действия пользователя: Попробуйте найти и исправить ошибку. Возможно, где-то пропущена скобка модуля. Так же вероятно, что скобки модуля поставлены в нелогичном порядке. Ещё вариант – пользователь попытался сделать запись в скобках модуля внутри другой записи в скобках модуля, по типу $|1+|2-x||$. Это некорректная запись. Верным вариантом в данной ситуации было бы использование базовой функции модуля «abs»:

$$|1+abs(2-x)|$$

(19)

Некорректная функция

Сообщение:

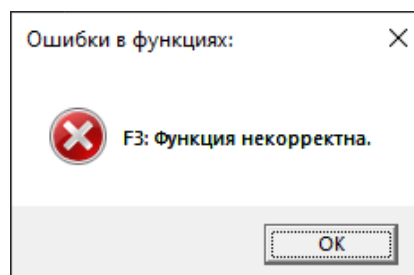


Рисунок 60 – Пример сообщения об ошибке «Функция некорректна»

Действия пользователя: Синтаксис функции абсолютно некорректный. Прочитайте справку заново и попробуйте исправить ошибку.

Неверный числовой формат

Сообщение:

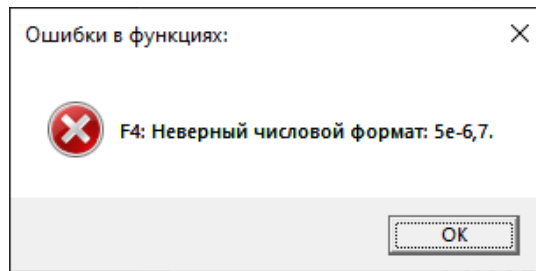


Рисунок 61 – Пример сообщения об ошибке «Неверный числовой формат»

Действия пользователя: Заново прочитайте информацию о константах в справке и попробуйте исправить ошибку в записи числа, указанного в сообщении.

Операторы в неправильном порядке

Сообщение:

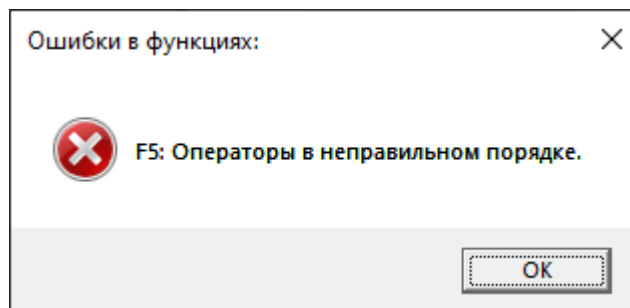


Рисунок 62 – Пример сообщения об ошибке «Операторы в неправильном порядке»

Действия пользователя: Проверьте, что каждый математический знак, т.е. оператор, стоит между двух операндов и попробуйте исправить ошибку.

Некорректный токен

Сообщение:

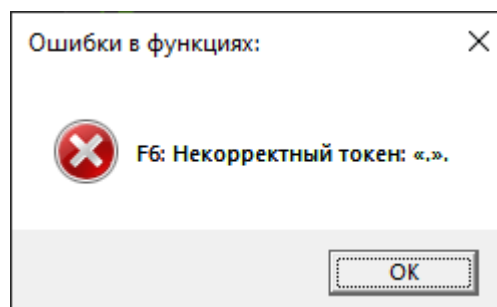


Рисунок 63 – Пример сообщения об ошибке «Некорректный токен»

Действия пользователя: Проверьте функцию на ненужные случайно попавшие символы, такие как точки или запятые, и уберите их.

Некорректные токены или несколько переменных

Сообщение:

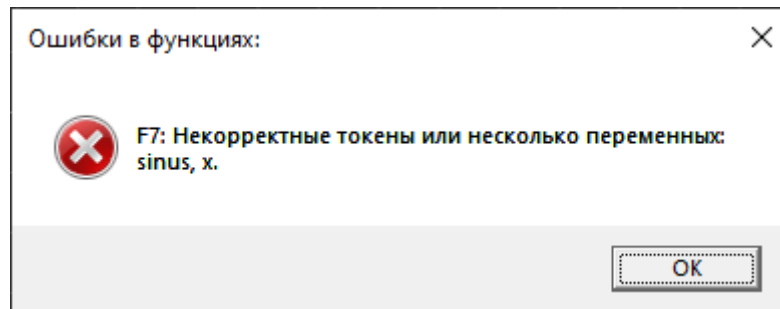


Рисунок 64 – Пример сообщения об ошибке «Некорректные токены или несколько переменных»

Действия пользователя: Ознакомьтесь со списком выведенных в сообщении токенов. Возможно, в нём находится некорректно записанное название базовой функции. Заново прочитайте справку и попытайтесь исправить ошибку.

Функция без аргументов

Сообщение:

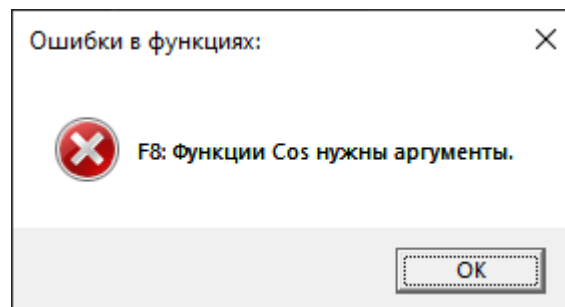


Рисунок 65 – Пример сообщения об ошибке «Функция без аргументов»

Действия пользователя: Проверьте свою функцию на правильность ввода. После токена функции должен стоять одиночный операнд или подфункция в скобках. Заново прочитайте справку и попытайтесь исправить ошибку.

Неверное количество аргументов

Сообщение:

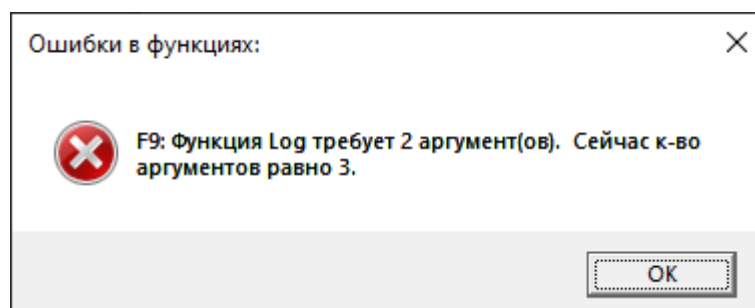


Рисунок 66 – Пример сообщения об ошибке «Неверное количество аргументов»

Действия пользователя: Проверьте свою функцию на правильность ввода. Убедитесь, что после функции с несколькими аргументами стоит группа операндов в скобках, разделённых знаком « ; ». Заново прочитайте справку и попытайтесь исправить ошибку.

Слишком много операторов

Сообщение:

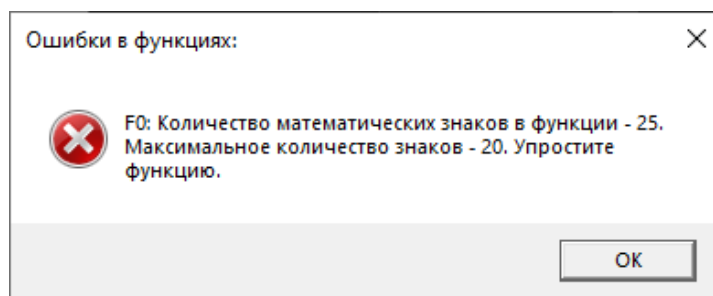


Рисунок 67 – Пример сообщения об ошибке «Слишком много операторов»

Действия пользователя: Введите более простую функцию с меньшим количеством операторов – математических знаков и разделителей.

4.4.2. Прочие сообщения**Прорисовка занимает слишком много времени**

Сообщение:

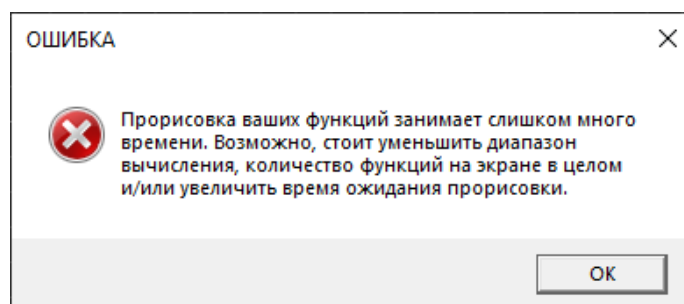


Рисунок 68 – Сообщение об ошибке «Прорисовка занимает слишком много времени»

Действия пользователя: Сделайте диапазон вычисления меньше: увеличьте его минимальное значение и уменьшите максимальное. Нажмите на кнопку минус или введите **Ctrl+MINUS**, чтобы уменьшить количество функций, или очистите несколько полей ввода. Также можно увеличить время ожидания в настройках.

Неверные данные в файле настроек

Сообщение:

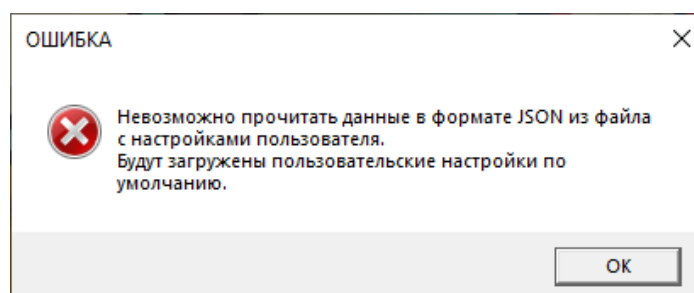


Рисунок 69 – Сообщение об ошибке «Неверные данные в файле настроек»

Действия пользователя: Ваши старые настройки удалены, работа будет продолжена со стандартными настройками. Что-то случилось с данными в файле настроек, который лежит в скрытой системной папке. Если вы заранее сохранили настройки в файл, импортируйте его.

Файл настроек не найден

Сообщение:

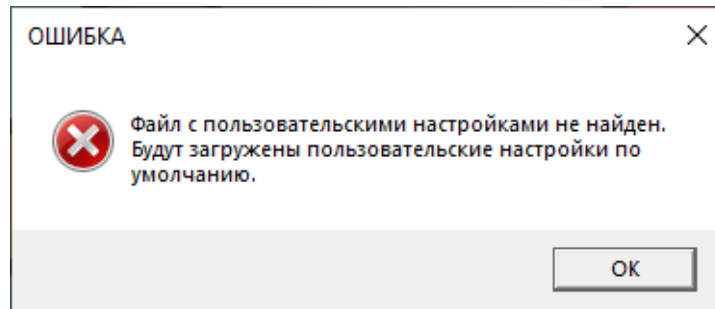


Рисунок 70 – Сообщение об ошибке «Файл настроек не найден»

Действия пользователя: Если это первый запуск программы, всё в порядке, проигнорируйте сообщение и начните работу. В ином случае это сообщение означает, что файл настроек в скрытой системной папке был удалён, перемещён или переименован. Если вы заранее сохранили настройки в файл, импортируйте его.

Импортирован некорректный файл

Сообщение:

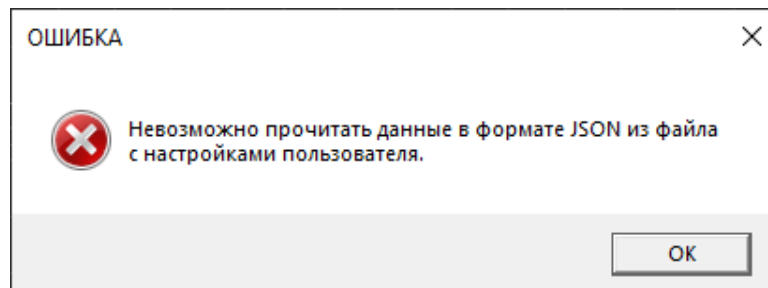


Рисунок 71 – Сообщение об ошибке «Импортирован некорректный файл»

Действия пользователя: Данные в файле, который вы попытались импортировать, невозможно прочитать и загрузить в качестве настроек, так как, возможно, в нём ошибка JSON или указаны некорректные данные. Попробуйте импортировать другой файл или настройте данные вручную.

Предупреждение о сбросе настроек

Сообщение:

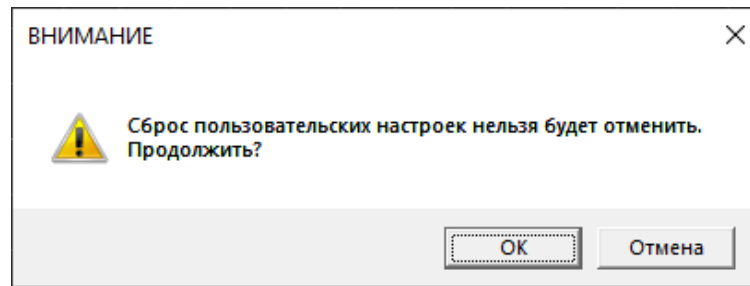


Рисунок 72 – Предупреждение о сбросе настроек

Действия пользователя: Нажмите «ОК», чтобы сбросить настройки, или нажмите «Отмена», чтобы оставить настройки в прежнем состоянии.

5 Мероприятия по информационной безопасности

5.1. Уязвимости программы

Основная возможная угроза информационной безопасности при эксплуатации программы – неправомерное распространение и изменение кода.

Чтобы третьи лица не смогли разобраться в исходном коде программы и внести туда вредоносные изменения, можно применить обфускацию кода.

Обфускация – изменение кода программы, не приводящее к изменениям в её поведении, но приводящее к затруднению понимания кода [24]. Для обфускации кода можно использовать специализированные утилиты.

Также под угрозой находятся личные данные пользователей, если они используют слабый пароль для аутентификации или скачали архив программы с непроверенного ресурса.

Для исполняемых файлов GraThing была выполнена обфускация.

Рекомендации по безопасной работе с программным продуктом описаны ниже.

5.2. Меры предотвращения угроз

Защита кода от неправомерного использования, копирования и взлома

Для обфускации исполняемых файлов GraThing была использована утилита ConfuserEx [40, 41]. ConfuserEx – это бесплатный инструмент с открытым кодом, предназначенный для обфускации приложений, использующих .NET Framework версий 2.0–4.8. Этот обфускатор преобразует собранные файлы так, чтобы затруднить понимание и анализ программы злоумышленниками.

Основные функции ConfuserEx:

- Обфускация имён, переименование классов, методов и т.п. в нечитаемый вид;
- Защита строковых данных;
- Модификация структуры кода, внедрение лишних инструкций;
- Защита кода от отладки и анализа;
- Скрытие или шифрование встроенных ресурсов.

Для исполняемых файлов GraThing были применены следующие настройки обфускации: переименование, добавление водяного знака, изменение метаданных типов, защита от отладки, обфускация констант, шифрование ресурсов и др.

Сравнение оригинальной сборки и сборки, прошедшей обфускацию показано на рис. 73. Как видно, декомпилятор больше не может вывести код программы и имена некоторых методов.

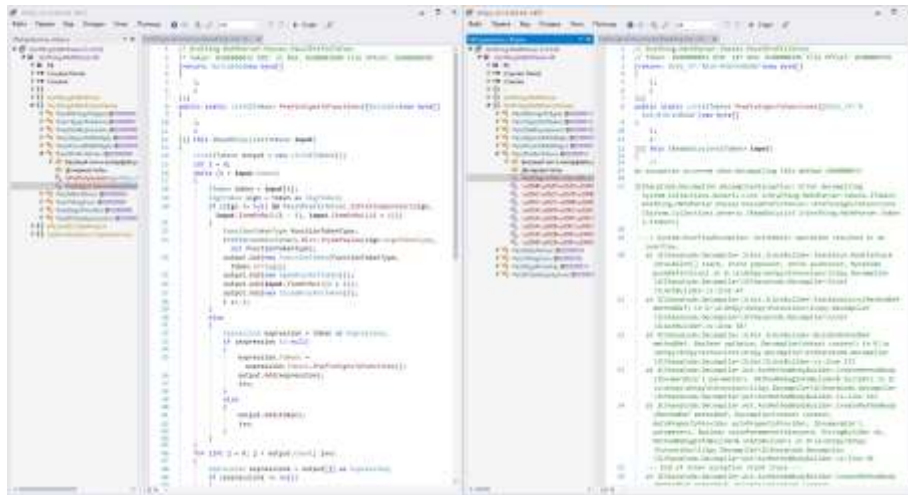


Рисунок 73 – Сравнение декомпилированного кода из оригинальной сборки (слева) и из той же сборки, прошедшей обфускацию (справа) в утилите dnSpy [44]

Защита исходного кода

Чтобы защитить исходный код программы от несанкционированных изменений и злоумышленников, он был сохранён в GitHub репозиторий. Репозиторий – хранилище всех файлов проекта и истории их изменений. Git – система контроля версий, позволяющая располагать файлы в удалённых репозиториях и работать над ними с нескольких устройств. GitHub – облачный сервис, основанный на Git, на серверах которого и хранятся данные репозитория [38].

Исходные файлы GraThing находятся в приватном репозитории, доступ к которому защищён аккаунтом с двухфакторной аутентификацией. Аутентификация – проверка подлинности субъекта или объекта [12, 16]. Двухфакторная аутентификация подразумевает использование двух методов проверки, к примеру, пароля и временного кода.

Рекомендации по работе

При использовании GraThing рекомендуется принимать следующие меры для обеспечения безопасности:

- Использовать межсетевой экран.

Межсетевой экран – программный или программно-аппаратный комплекс, защищающий устройство от несанкционированного доступа из внешней сети [20]. В Windows есть стандартный межсетевой экран, Брандмауэр Windows [23].

- Скачивать только с официального ресурса.

Скачать приложение можно только по ссылке <https://github.com/TaniachiFractal/GraThing/releases>, как было описано в п. 4.1. В архивы, скачанные из других мест, злоумышленники могли добавить опасные файлы или вредный код.

- Использовать антивирусное ПО.

Антивирус – приложение, которое защищает устройство от вредоносных программ [10]. Для того, чтобы убедиться в безопасности файлов GraThing, можно просканировать их подобной программой. Кроме того, безопаснее работать со включённым антивирусом-сканером, постоянно проверяющим запущенные программы.

- Использовать защищённую учётную запись Windows.

Чтобы ваши личные файлы, связанные с GraThing, не были под угрозой, не используйте учётную запись без пароля или с очень простым паролем.

Заключение

В ходе работы над проектом была успешно разработана программа для построения графиков функций. Все задачи проекта, в том числе реализация интерпретатора математических функций и визуализации графиков, были успешно выполнены.

Данная работа будет полезна как в образовании, для обучения школьников математическим функциям, так и для выполнения расчётов и создания графических работ. GraThing – абсолютно бесплатная российская разработка, доступная широкому кругу лиц.

Были разработаны архитектура и интерфейс приложения; для их реализации использовался Windows Presentation Framework с языком C#, разработка велась в среде Visual Studio. Для сохранения данных в файл был выбран формат JSON, для разметки справки HTML с CSS, а для модульного тестирования – инструмент xUnit.

Были реализованы 4 основных модуля:

- Модуль интерфейсов, отвечающий за ввод и вывод;
- Главный модуль, что связывает все модули;
- Модуль интерпретатора математических функций;
- Модуль визуализации графиков.

В проекте применён архитектурный паттерн MVVM и использованы принципы асинхронного программирования. Были созданы модульные тесты и руководство пользователя.

Была выполнена обфускация кода при помощи утилиты ConfuserEx.

Была достигнута цель проекта – разработана простая в использовании программа для построения графиков функций «GraThing», позволяющая визуализировать двумерные математические функции, в том числе параметрические, в прямоугольной и полярной системах координат.

У GraThing есть множество перспектив развития:

Во-первых, работа с левой частью функции. На данный момент она игнорируется. С такой опцией можно будет, к примеру, нарисовать круг радиуса 2 по формуле

$$4 = x^2 + y^2 \quad (20)$$

На данный момент это реализуемо при помощи параметрической функции

$$2 \sin x; 2 \cos x \quad (21)$$

Во-вторых, визуализация неравенств. Это позволит закрашивать целые области поля некоторым цветом или штриховкой, как на рис. 74.

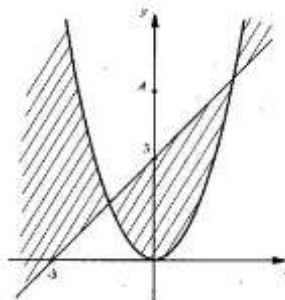


Рисунок 74 – Визуализация множества решений неравенства $(y - x^2)(y - x - 3) < 0$ [18]

В-третьих, работа с функциями 2 переменных. Их графики находятся в трёхмерном пространстве, их можно визуализировать как при помощи трёхмерной графики, так и двумерной, где третья координата показана цветом, как на рис. 75.

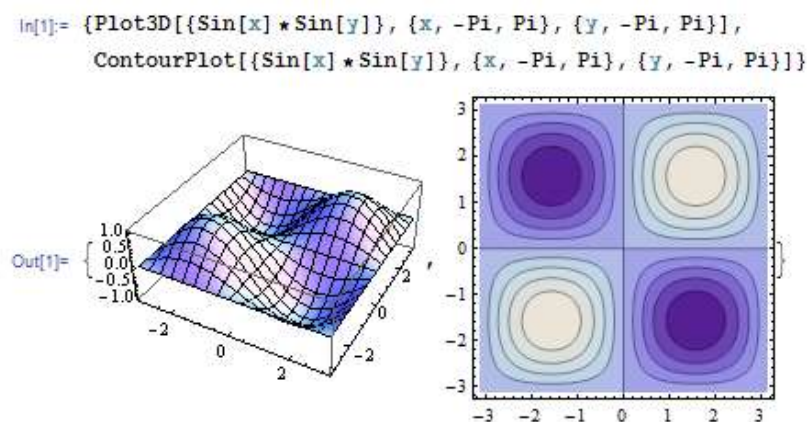


Рисунок 75 – Визуализация функции двух переменных средствами трёхмерной и двумерной графики [14]

Подводя итог, в программу для построения графиков функций можно добавлять бессчётное множество опций, у таких программ есть множество перспектив развития и их область применения очень широка.

Список источников

Нормативная документация

- 1) ГОСТ 19.106-78 Единая система программной документации. Требования к программным документам, выполненным печатным способом
[Электронный ресурс] – <https://protect.gost.ru/document.aspx?control=7&id=156370>
- 2) ГОСТ 19.401-78 Единая система программной документации. Текст программы. Требования к содержанию и оформлению
[Электронный ресурс] – <https://protect.gost.ru/document.aspx?control=7&id=155463>
- 3) ГОСТ 34.201-2020 Информационные технологии. Комплекс стандартов на автоматизированные системы. Виды, комплектность и обозначение документов при создании автоматизированных систем
[Электронный ресурс] – <https://protect.gost.ru/document1.aspx?control=31&id=241756>
- 4) ГОСТ 34.602-2020 Информационные технологии. Комплекс стандартов на автоматизированные системы. Техническое задание на создание автоматизированной системы
[Электронный ресурс] – <https://protect.gost.ru/document1.aspx?control=31&id=241754>
- 5) ГОСТ Р 2.105-2019 Единая система конструкторской документации. Общие требования к текстовым документам
[Электронный ресурс] – <https://protect.gost.ru/document1.aspx?control=31&id=237857>
- 6) ГОСТ Р ИСО/МЭК 25051-2017 - Информационные технологии. Системная и программная инженерия. Требования и оценка качества систем и программного обеспечения (SQuaRE). Требования к качеству готового к использованию программного продукта (RUSP) и инструкции по тестированию
[Электронный ресурс] – <https://protect.gost.ru/document.aspx?control=7&id=217667>

Монографическая и учебная литература

- 7) Выгодский М.Я. Справочник по высшей математике. М.: Издательство «Наука», 1964.
- 8) Петцольд Ч. Microsoft Windows Presentation Foundation Базовый курс. М.: Издательство «Русская Редакция»; СПб.: Питер, 2008.
- 9) Arnaud W. Learn WPF MVVM – XAML, C# and the MVVM pattern. Morrisville, NC: Lulu, 2017.

Интернет – ресурсы

- 10) Антивирус – что это такое и зачем он нужен [Электронный ресурс] – <https://blog.skillfactory.ru/chto-takoe-antivirus-i-zachem-on-nuzhen/>
- 11) Асинхронное программирование [Электронный ресурс] – <https://blog.skillfactory.ru/glossary/asinhronnoe-programmirovanie/>
- 12) Аутентификация [Электронный ресурс] – <http://www.sberbank.ru/ru/person/kibrary/vocabulary/autentifikaciya>
- 13) Введение во внедрение зависимостей [Электронный ресурс] – <https://metanit.com/sharp/dotnet/1.1.php>
- 14) Визуализация функций двух переменных на плоскости [Электронный ресурс] – <https://intuit.ru/studies/courses/4765/1039/lecture/15747?page=3>

- 15) Графический метод решения задач линейного программирования [Электронный ресурс] – https://zelcollege50.mskobr.ru/files/issledovanie_linejnoe_programmirovanie.pdf
- 16) Двухфакторная аутентификация [Электронный ресурс] – <http://www.sberbank.ru/ru/person/kibrary/articles/dvuhfaktornaya-autentifikaciya>
- 17) Десятичные дроби и действия с ними [Электронный ресурс] – <https://maximumtest.ru/spb/uchebnik/9-klass/matematika/desyatichnye-drobi-i-deystviya-s-nimi>
- 18) Изображение на координатной плоскости множества решений неравенств с двумя переменными и их систем [Электронный ресурс] – https://ru.ruwiki.ru/wiki/Изображение_на_координатной_плоскости_множества_решений_неравенств_с_двумя_переменными_и_их_систем
- 19) Интерфейсы — определение поведения для нескольких типов [Электронный ресурс] – <https://learn.microsoft.com/ru-ru/dotnet/csharp/fundamentals/types/interfaces>
- 20) Межсетевой экран, Firewall и Брандмауэр [Электронный ресурс] – <https://cloud.ru/blog/brandmauer-firewall-i-mejsetevoy-ekran-chto-eto-takoe>
- 21) На чем пишут десктопные приложения: обзор инструментов [Электронный ресурс] – <https://sky.pro/wiki/javascript/na-chem-pishut-desktopnye-prilozheniya-obzor-instrumentov/>
- 22) Неперово число [Электронный ресурс] – <https://bigenc.ru/c/neperovo-chislo-1ba02c>
- 23) Обзор брандмауэра Windows [Электронный ресурс] – <https://learn.microsoft.com/ru-ru/windows/security/operating-system-security/network-security/windows-firewall/>
- 24) Обфускация [Электронный ресурс] – <https://encyclopedia.kaspersky.ru/glossary/obfuscation/>
- 25) Общие сведения о WPF в Visual Studio [Электронный ресурс] – <https://learn.microsoft.com/ru-ru/dotnet/desktop/wpf/getting-started/introduction-to-wpf-in-vs>
- 26) Общие сведения о привязке данных (WPF .NET) [Электронный ресурс] – <https://learn.microsoft.com/ru-ru/dotnet/desktop/wpf/data>
- 27) Общие сведения о цветовых моделях [Электронный ресурс] – http://product.corel.com/help/CorelDRAW/540240626/Main/RU/Doc/wwhelp/wwhimpl/common/html/wwhelp.htm?context=CorelDRAW_Help&file=CorelDRAW-Understanding-color-models.html
- 28) Отмена в управляемых потоках [Электронный ресурс] – <https://learn.microsoft.com/ru-ru/dotnet/standard/threading/cancellation-in-managed-threads>
- 29) Отрицательные числа [Электронный ресурс] – https://uchi.ru/znaniya/math/otritsatelnye_chisla
- 30) Переменная [Электронный ресурс] – <https://bigenc.ru/c/peremennaia-bb7abf>
- 31) Проектирование приложений: основные принципы и методы [Электронный ресурс] – <https://sky.pro/wiki/javascript/proektirovanie-prilozhenij-osnovnye-principy-i-metody/>
- 32) Прямоугольная и полярная системы координат [Электронный ресурс] – <https://ilyachalov.livejournal.com/92853.html>
- 33) Целое число [Электронный ресурс] – <https://bigenc.ru/c/tseloe-chislo-e58eb7>
- 34) Численное дифференцирование [Электронный ресурс] – https://ru.wikipedia.org/wiki/Численное_дифференцирование
- 35) Число π [Электронный ресурс] – <https://bigenc.ru/c/chislo-p-bf3bee>
- 36) Число ϕ [Электронный ресурс] – <https://www.maeckes.nl/Getal%20phi%20RU.html>

- 37) Экспоненциальная запись чисел [Электронный ресурс] – <https://allcalc.ru/node/1103>
- 38) About GitHub and Git [Электронный ресурс] – <https://docs.github.com/en/get-started/start-your-journey/about-github-and-git>
- 39) Colors HEX [Электронный ресурс] – https://www.w3schools.com/colors/colors_hexadecimal.asp
- 40) ConfuserEx – Documentation [Электронный ресурс] – <https://github.com/mkaring/ConfuserEx/wiki/Documentation>
- 41) ConfuserEx - README [Электронный ресурс] – <https://github.com/mkaring/ConfuserEx/blob/master/README.md>
- 42) CSS: каскадные таблицы стилей [Электронный ресурс] – <https://developer.mozilla.org/ru/docs/Web/CSS>
- 43) Desmos [Электронный ресурс] – <https://www.desmos.com/calculator>
- 44) dnSpy – README [Электронный ресурс] – <https://github.com/dnSpy/dnSpy/blob/master/README.md>
- 45) HTML [Электронный ресурс] – <https://developer.mozilla.org/ru/docs/Web/HTML>
- 46) ICommand Интерфейс [Электронный ресурс] – <https://learn.microsoft.com/ru-ru/dotnet/api/system.windows.input.icommand>
- 47) IOException Класс [Электронный ресурс] – <https://learn.microsoft.com/ru-ru/dotnet/api/system.io.ioexception>
- 48) JSON против XML [Электронный ресурс] – <https://appmaster.io/ru/blog/json-protiv-xml>
- 49) Learning resources: MATLAB [Электронный ресурс] – <https://researchcomputing.princeton.edu/external-online-resources/matlab>
- 50) MathWay [Электронный ресурс] – <https://www.mathway.com/>
- 51) Microsoft Excel [Электронный ресурс] – <https://www.microsoft.com/ru-ru/microsoft-365/excel>
- 52) Mixing WPF and WinForms [Электронный ресурс] – <https://www.red-gate.com/simple-talk/development/dotnet-development/mixing-wpf-and-winforms/>
- 53) NUnit vs xUnit vs MSTest: .NET Unit Testing Framework Comparison [Электронный ресурс] – <https://daily.dev/blog/nunit-vs-xunit-vs-mstest-net-unit-testing-framework-comparison>
- 54) Painting by functions [Электронный ресурс] – <https://nrich.maths.org/problems/painting-functions>
- 55) System.Drawing Пространство имён [Электронный ресурс] – <https://learn.microsoft.com/ru-ru/dotnet/api/system.drawing>
- 56) Unit-тестирование [Электронный ресурс] – <https://blog.skillfactory.ru/glossary/unit-testirovanie/>
- 57) UserControl Класс [Электронный ресурс] – <https://learn.microsoft.com/ru-ru/dotnet/api/system.windows.controls.usercontrol>
- 58) WebBrowser Класс [Электронный ресурс] – <https://learn.microsoft.com/ru-ru/dotnet/api/system.windows.controls.webbrowser>
- 59) WolframAlpha [Электронный ресурс] – <https://www.wolframalpha.com/>
- 60) WPF vs. WinForms [Электронный ресурс] – <https://wpf-tutorial.com/ru/2/несколько-слов-о-wpf/wpf-vs-winforms/>
- 61) XAML preview tool window for WPF in Rider [Электронный ресурс] – <https://blog.jetbrains.com/dotnet/2018/03/29/xaml-preview-tool-window-wpf-rider/>