МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждениевысшего профессионального образования
«ЧЕРЕПОВЕЦКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Институт (факультет) Институт информационных технологий
Кафедра МПО ЭВМ

КУРСОВАЯ РАБОТА

по дисциплине Структурное программирование

на тему Программирование на языке высокого уровня

Выполнил студент группы 1ПИб-02-2оп-22
группа

Направления подготовки (специальности)
09.03.04 Программная инженерия
шифр, наименование

Овчинников Максим Владимирович
фамилия, имя, отчество

Руководитель
Пышницкий Константин Михайлович
фамилия, имя, отчество

Старший преподаватель
должность

Дата представления работы
«___»_____2023 г.

Заключение о допуске к защите

Оценка _____, _____
количество баллов

Подпись преподавателя _____

2023 год

Оглавление

Введение

Программирование на языке высокого уровня представляет собой важную область современной информатики, позволяющую разработчикам создавать сложные и функциональные программы с использованием абстрактных и удобных инструментов. В отличие от низкоуровневых языков, язык высокого уровня может использовать элементы естественного языка, быть более простым в использовании и автоматизировать значительные аспекты вычислительных систем, таких как управление памятью. Это делает процесс разработки программы более простым и понятным по сравнению с использованием языка низкого уровня [1].

Один из наиболее популярных высокоуровневых языков программирования - C++. Он является развитием языка C и добавляет функциональность объектно-ориентированного программирования, что делает его мощным инструментом для создания сложных программных систем [2].

В данной курсовой работе требуется разработать программу, которая способна генерировать графические орнаменты, а также обеспечивать возможность изменения их размера и цвета. Программа также должна поддерживать сохранение полученного графического изображения.

Пользователи смогут использовать разработанную программу для создания и настройки орнаментов, что позволит им придавать уникальный стиль и привлекательность своим графическим элементам.

1.Описание предметной области

Предметная область курсовой работы связана с разработкой программы для создания орнаментов в Microsoft Forms. Орнаменты играют важную роль в дизайне, поскольку они способны добавить украшение, стиль и эстетическую ценность к графическим элементам и интерфейсам приложений. Они помогают создать уникальный и запоминающийся облик, придают композиции графический интерес и усиливают визуальное впечатление от дизайна.

Орнаменты могут быть использованы на разных уровнях дизайна. Они могут быть включены в основные графические элементы, такие как рамки, баннеры или заголовки, чтобы придать им декоративный вид и привлекательность. Орнаменты также могут использоваться в качестве фона или заливки для создания текстур и узоров, которые добавляют глубину и интерес к дизайну.

Важно отметить, что орнаменты могут быть в различных стилях и формах, от классических и ретро до современных и абстрактных. Они могут быть геометрическими, флоральными, в виде гравюр или символов. Выбор орнамента зависит от общего стиля дизайна и целей, которые нужно достичь.

Кроме эстетической ценности, орнаменты также могут выполнять функциональные задачи. Они могут использоваться для разделения различных разделов на странице или в приложении, создания визуальных связей между элементами или указания на важные части информации [3].

Windows Forms представляет собой платформу для разработки пользовательского интерфейса классических приложений под операционную систему Windows. Она предлагает эффективный и удобный способ создания таких приложений с помощью визуального конструктора в среде разработки Visual Studio. Благодаря возможности перетаскивания визуальных элементов управления, размещение их на форме становится процессом, который легко освоить, что облегчает создание классических приложений [4].

Предметной областью является орнамент множества пересекающихся квадратов, вписанных в окружности (рис.1).

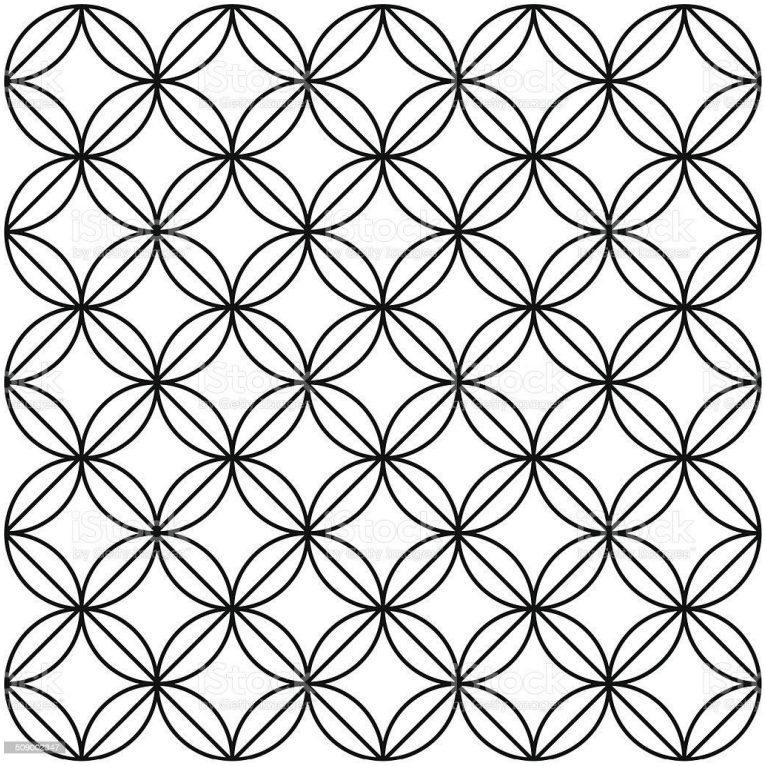

Рис.1. Орнамент множества пересекающихся квадратов, вписанных в окружности.

2.Описание классов Graphics, Pen и Brush

Реализация рисования фигур является одним из важных аспектов в разработке графических приложений. В языке программирования C++ с использованием Windows Forms можно воспользоваться классами Graphics, Pen и Brush для этой цели. Пространство имен System.Drawing предоставляет доступ к функциональным возможностям графического интерфейса GDI+, предлагая около 50 классов, включая классы Graphics, Pen и Brush. Эти классы позволяют разработчикам управлять графическими объектами, выбирать стили рисования, цвета и другие параметры для создания привлекательных и интерактивных пользовательских интерфейсов [5].

2.1. Класс Graphics

Класс Graphics в Windows Forms и языке программирования C++ предоставляет функциональность для рисования и взаимодействия с

графическими объектами. Он позволяет разработчикам создавать и управлять элементами графического интерфейса пользователя, рисовать линии, фигуры, текст и изображения, а также управлять атрибутами рисования, такими как цвет, шрифт и прозрачность.

Основные методы представлены в табл. 1:

Таблица 1

Методы класса Graphics

Имя	Описание
AddMetafileComment	Добавляет комментарий к текущему объекту Metafile.
BeginContainer()	Сохраняет текущее состояние объекта Graphics в графическом контейнере и открывает новый контейнер для использования.
Clear	Очищает всю поверхность рисования и заливает её указанным цветом фона.
CopyFromScreen(Int32, Int32, Int32, Int32, Size)	Копирует данные о цветах пикселей с экрана на поверхность рисования объекта Graphics.
Dispose	Освобождает ресурсы, используемые объектом Graphics.
DrawArc(Pen, Int32, Int32, Int32, Int32, Int32, Int32)	Рисует дугу, являющуюся частью эллипса, заданного координатами, шириной и высотой.
DrawBezier(Pen, Point, Point, Point, Point)	Рисует кривую Безье, определяемую четырьмя точками.
DrawLine(Pen, Int32, Int32, Int32, Int32)	Рисует линию, соединяющую две точки, заданные координатами.
FillClosedCurve(Brush, Point[], FillMode)	Заполняет внутреннюю часть замкнутой сплайновой кривой, определенной массивом точек, используя указанный режим заливки.
FillEllipse(Brush, Int32, Int32, Int32, Int32)	Заполняет внутреннюю часть эллипса, заданного координатами, шириной и высотой.
FillPath	Заполняет внутреннюю часть объекта GraphicsPath.
FillPie(Brush, Rectangle, Single, Single)	Заполняет внутреннюю часть сектора эллипса, заданного структурой RectangleF и двумя радиальными линиями.
FillPie(Brush, Int32, Int32, Int32, Int32, Int32, Int32)	Заполняет внутреннюю часть сектора эллипса, заданного координатами, шириной, высотой и двумя радиальными линиями.
FillPolygon(Brush, Point[])	Заполняет внутреннюю часть многоугольника, заданного массивом точек.
FillRectangle(Brush, Rectangle)	Заполняет внутреннюю часть прямоугольника, заданного структурой Rectangle.

2.2. Класс Pen

Класс Pen представляет перо, которое используется для рисования линий и контуров в графическом приложении. Он определяет свойства, такие как цвет, толщина и стиль линии.

Методы представлены в табл. 2:

Таблица 2

Методы класса Pen

Имя	Описание
Pen(Color)	Инициализирует новый экземпляр класса Pen с указанным цветом.
Pen(Color, Single)	Инициализирует новый экземпляр класса Pen с указанными свойствами Color и Width. (Width - устанавливает ширину пера Pen, в единицах объекта Graphics, используемого для рисования)

2.3. Класс Brush

Класс Brush представляет кисть, которая используется для заливки областей и элементов в графическом приложении. Он определяет различные типы кистей, такие как сплошная кисть, текстурная кисть или градиентная кисть, позволяя создавать разнообразные эффекты заливки.

Класс Brush является абстрактным базовым классом, который не предназначен для прямого использования. Вместо этого, для создания объектов "кисть" необходимо использовать классы, унаследованные от Brush, такие как SolidBrush, TextureBrush и LinearGradientBrush, которые предоставляют конкретные реализации для различных типов кистей.

3.Описание созданного приложения

В данном разделе описаны основные этапы разработки приложения для построения орнамента пересекающихся квадратов, вписанных в окружность. пересекающихся квадратов, вписанных в окружности.

3.1. Постановка задачи

Для корректной работы программы требуется выполнить следующие задачи:

- При нажатии определенной кнопки, программа должна создавать графическое изображение орнамента;
- Пользователям должна быть предоставлена возможность выбора цвета заднего фона;
- Программа должна позволять изменять масштаб и цвет орнамента;
- При необходимости, пользователи должны иметь возможность сохранять созданное изображение в файл;
- При изменении масштаба и цвета, программа должна позволять повторно создавать графическое изображение орнамента.

3.2. Логическое проектирование

Для построения орнамента (рис.1) необходимы две функции: «DrawCircle» и OrnamentDraw. Функция DrawCircle отвечает за построение основной фигуры орнамента, квадрата, вписанного в круг. Квадрат развернут на 45 градусов относительно стандартного расположения, а его вершины расположены на окружности в точках 0, 90, 180 и 270 градусов.

Функция OrnamentDraw реализует алгоритм построения орнамента на элементе pictureBox1. Орнамент состоит из серии элементов, которые рисуются с помощью функции DrawCircle. В начале функции создаются объекты Graphics и Pen, которые будут использоваться для рисования на элементе pictureBox1. Параметры цвета и толщины линии передаются в функцию. Затем определяются

размеры элемента pictureBox1 и вычисляются размеры каждого элемента орнамента и количество элементов в ряду. Далее задаются значения для начальной позиции первого ряда. Координаты X и Y определяются таким образом, чтобы ряд выходил за пределы экрана сверху. Затем вычисляется количество рядов, необходимых для заполнения всей высоты элемента pictureBox1. Запускается двойной цикл, который проходит по всем рядам и элементам в каждом ряду. Внутри цикла определяются координаты центра текущего элемента орнамента и его радиус. Затем вызывается функция DrawCircle, которая рисует круг с использованием переданных параметров. После завершения цикла для элементов в ряду, происходит корректировка позиции для следующего ряда. Координата X сбрасывается в начальное значение, а координата Y увеличивается на сумму размера элемента и расстояния между рядами. Также происходит переключение между типами рядов (1 и 2) на основе номера текущего ряда. Если номер ряда нечетный, координата X корректируется. После завершения всех циклов, орнамент будет нарисован на элементе pictureBox1.

Функция DrawCircle рисует круг и квадрат в переданном объекте Graphics с использованием заданного пера Pen. Круг определяется центром (centerX, centerY) и радиусом radius. Квадрат строится на основе координат круга. Сначала вычисляются координаты верхней, правой, нижней и левой точек квадрата на основе центра и радиуса круга. Затем создается массив diamondPoints, содержащий эти координаты в определенном порядке. Функция DrawPolygon используется для рисования квадрата, принимая объект Graphics, перо Pen и массив точек квадрата. Кроме того, с помощью функции DrawEllipse рисуется круг, используя объект Graphics, перо Pen, координаты левого верхнего угла описывающего прямоугольника (x, y) и его ширину и высоту (diameter, diameter). Это создает окружность, вписанную в данный прямоугольник. Таким образом, функция DrawCircle выполняет отрисовку круга и квадрата в заданных координатах, используя заданное перо и объект Graphics.

Алгоритм работы программы представлен на блок-схеме (рис. 2):

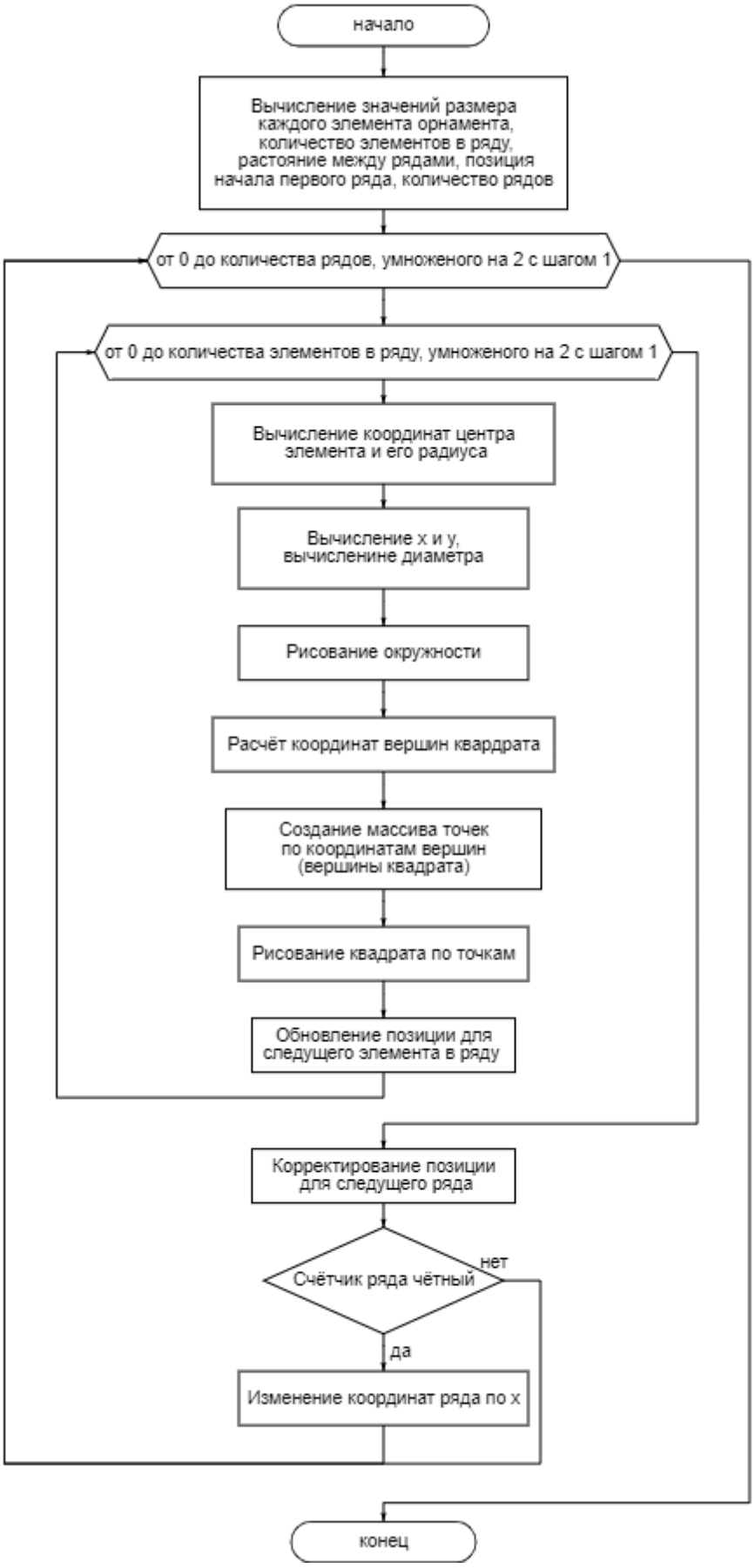

Рис. 2. Алгоритм работы программы в блок-схеме

3.3. Физическое проектирование

Переменные, примененные для реализации программы, представлены в табл. 3:

Таблица 3

Переменные

Наименование	Обозначение	Тип данных
Размер окружности	radius	int
Горизонтальная позиция центра окружности	centerX	int
Вертикальная позиция центра окружности	centerY	int
Ширина блока рисования	width	int
Высота блока рисования	height	int
Размер каждого элемента орнамента	elementSize	int
Количество элементов в ряду	elementsPerRow	int
Расстояние между рядами	rowSpacing	int
Позиция начала первого ряда	startX	int
Позиция начала первого ряда	startY	int
Координата ряда фигур по оси x	rowX	int
Координата ряда фигур по оси y	rowY	int
Количество рядов	numRows	int
Толщина линии	WL	double
Цвет фона	bf	Color
Размер орнамента	size	int
Цвет линий орнамента	f	Color
Координата вершины квадрата по оси x	topX	int
Координата вершины квадрата по оси y	topY	int
Координата вершины квадрата по оси x	rightX	int
Координата вершины квадрата по оси y	rightY	int
Координата вершины квадрата по оси x	bottomX	int
Координата вершины квадрата по оси y	bottomY	int
Координата вершины квадрата по оси x	leftX	int

Координата вершины квадрата по оси y	leftY	int

Спецификация функций представлена в табл. 4:

Таблица 4

Спецификация функций

Имя модуля	Заголовок процедуры или функции	Формальные параметры	Выполняемое действие
Project7.sln	button1_Click	System::Object^ sender, System::EventArgs^ e	Функция выполняет вызов функции построения орнамента
Project7.sln	numericUpDown1_ValueChanged	System::Object^ sender, System::EventArgs^ e	Функция задает толщину линии орнамента
Project7.sln	colorButton_Click	System::Object^ sender, System::EventArgs^ e	Функция вызывает диалоговое окно для выбора цвета орнамента
Project7.sln	button2_Click	System::Object^ sender, System::EventArgs	Функция вызывает диалоговое окно для выбора цвета фона
Project7.sln	button3_Click	System::Object^ sender, System::EventArgs	Функция вызывает диалоговое окно для сохранения изображения орнамента
Project7.sln	trackBar1_Scroll	System::Object^ sender, System::EventArgs	Функция задает масштаб орнамента.
Project7.sln	DrawCircle	Graphics^ graphics, Pen^ pen, int centerX, int centerY, int radius	Функция рисует квадрат, вписанный в окружность
Project7.sln	OrnamentDraw	int size, Color color, int WL	Функция рисует весь орнамент путем пересечения множества фигур

3.4. Тестирование

Тестовые данные и результаты тестирования представлены в табл. 5, 6:

Таблица 5

Тестовые данные

Исходные данные	Тестируемая функция	Ожидаемый результат
Нажатие кнопки «Построить орнамент»	button1_Click	Построение орнамента
Нажатие кнопки «Изменить цвет орнамента»	colorButton_Click	Изменение цвета орнамента на выбранный
Изменение толщины линии орнамента	numericUpDown1_ValueChanged	Изменения значения толщины линии (от 1 до 10)
Нажатие кнопки «Изменить цвет фона»	button2_Click	Изменение цвета фона на выбранный
Нажатие кнопки «Сохранить»	button3_Click	Сохранение изображения в выбранном формате (PNG, JPEG, Bitmap)
Изменение масштаба орнамента	trackBar1_Scroll	Изменение масштаба орнамента на выбранный (от 1 до 5)

Таблица 6

Результаты тестирования

Дата тестирования	Тестируемый модуль	Кто проводил тестирование	Описание теста	Результат тестирования
21.05.2023	Project7.sln	Овчинников М.В.	Проверка работы кнопки «Построить орнамент»	Успех
21.05.2023	Project7.sln	Удальцов А.П.	Проверка работы изменения цвета орнамента	Успех
21.05.2023	Project7.sln	Воронин И.Е.	Проверка работы изменения цвета фона	Успех
21.05.2023	Project7.sln	Симаньков А.Е	Проверка работы сохранения	Успех

			изображения орнамента	
21.05.2023	Project7.sln	Милавин Д.А.	Проверка работы изменения масштаба орнамента	Успех

3.5. Результаты работы

В результате работы было создано приложение для построения орнамента множества пересекающихся квадратов, вписанных в окружности в Windows Forms на языке C++. Орнамент строится по изначальным значениям, присутствует возможность изменения масштаба, задания цвета орнамента и цвета заднего фона, а также возможность сохранения изображения.

Интерфейс понятен и функционален (рис. 3).

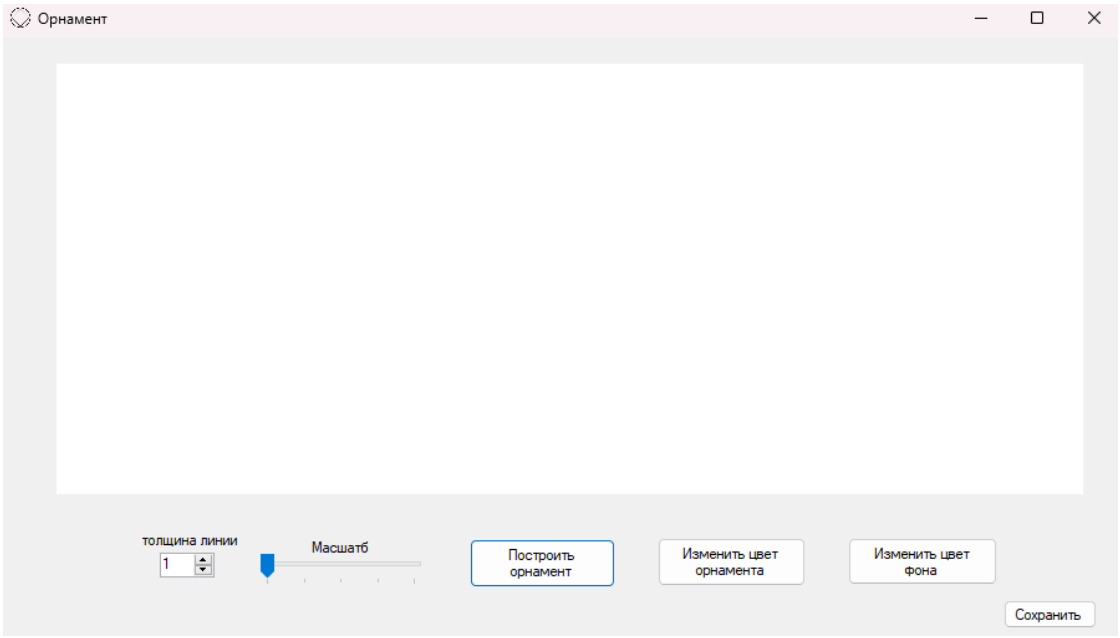

Рис. 3. Интерфейс программы

Заключение

В рамках проекта была разработана программа на языке C++ с использованием Windows Forms, которая создает графическое изображение орнамента из пересекающихся квадратов, вписанных в окружность.

В процессе работы над проектом были получены навыки работы с Windows Forms и овладены возможностями классов Graphics, Pen и Brush.

Все поставленные задачи были успешно выполнены.

Источники

1. Высокоуровневые языки программирования [электронный ресурс]: https://en.wikipedia.org/wiki/High-level_programming_language. Дата доступа: 05.06.2023

2. Язык программирования C++ [электронный ресурс]: https://ru.wikipedia.org/wiki/C%2B%2B Дата доступа: 05.06.2023

3. https://ru.wikipedia.org/wiki/Орнамент [электронный ресурс]: Дата доступа: 05.06.2023

4. Windows Forms [электронный ресурс]: https://ru.wikipedia.org/wiki/Windows_Forms Дата доступа: 05.06.2023

5. Библиотека System.Drawing [электронный ресурс]: https://grafika.me/node/24 Дата доступа: 05.06.2023

6. Блок-схема [электронный ресурс]: https://programforyou.ru/block-diagram-redactor Дата доступа: 05.06.2023

Приложение 1. Техническое задание

МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждениевысшего профессионального образования
«ЧЕРЕПОВЕЦКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Институт информационных технологий
наименование института (факультета)

Кафедра математического и программного обеспечения ЭВМ
наименование кафедры

Структурное программирование
наименование дисциплины в соответствии с учебным планом

УТВЕРЖДАЮ
Зав. кафедрой_____
д.т.н., профессор___Ершов Е.В.
«___»_____2023г.

ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ ВЫСОКОГО УРОВНЯ

Техническое задание на курсовую работу
Листов 7

Руководитель: старший преподаватель
Пышницкий К.М.
Исполнитель: студент гр. 1ПИб-02-2оп-22
Овчинников М.В.

2023 год

Введение

Программа для построения орнамента пересекающихся квадратов, вписанных в окружности, в Windows Forms рисует изображение и сохраняет полученный результат.

Данный проект предназначен для формирования знаний и навыков по дисциплине «Структурное программирование».

1. Основания для разработки

Основанием для разработки является задание на курсовую работу по дисциплине «Структурное программирование», выданное на кафедре МПО ЭВМ ИИТ ЧГУ.

Наименование темы разработки: программирование на языке высокого уровня.

2. Назначение разработки

Цель разработки этой программы заключается в создании приложения, которое позволяет генерировать и визуализировать орнаменты, состоящие из пересекающихся квадратов, вписанных в окружности. Пользователь имеет возможность изменять размеры элементов орнамента с помощью графического интерфейса, а также выбирать цвет заднего фона и цвет линий орнамента. Программа также позволяет сохранять полученный орнамент в файлы форматов JPEG, PNG и BMP для дальнейшего использования. Это приложение может быть полезным для создания уникальных орнаментов и использования их изображений в дизайне.

3. Требования к программе

3.1. Требования к функциональным характеристикам

Функциональные требования программы включают:

1. Генерация орнамента: Пользователь может создавать орнаменты, состоящие из пересекающихся квадратов, вписанных в окружности. Генерированный орнамент должен быть наглядно отображен на экране, чтобы пользователь мог оценить его внешний вид.

2. Изменение параметров орнамента: Пользователь может настраивать параметры орнамента, такие как радиус, цвет и масштаб. При каждом изменении параметров должны отображаться соответствующие изменения визуализации орнамента.

3. Сохранение орнамента: Пользователь имеет возможность сохранить сгенерированный орнамент в файле, чтобы использовать его позже в своем дизайне или проекте.

4. Интуитивный интерфейс: Интерфейс программы должен быть интуитивно понятным и удобным в использовании. Пользователь должен легко настраивать параметры орнамента и мгновенно видеть результаты своих изменений.

3.2. Требования к надежности

Программа должна обладать высокой стабильностью, исключать возникновение непредвиденных ошибок и обрабатывать все возможные ситуации и ошибки. Она должна корректно отображать пересекающиеся квадраты, вписанные в окружности при любых допустимых значениях параметров, чтобы гарантировать правильность и надежность визуализации орнамента.

3.3. Условия эксплуатации

Необходимо обеспечить совместимость программы с операционной системой пользователя, а также учесть условия эксплуатации персонального

компьютера при определении условий эксплуатации программы.

3.4. Требования к составу и параметрам технических средств

Для работы программы требуются компьютер с определенными минимальными техническими характеристиками, включающими:

- Процессор с тактовой частотой не менее 1,2 ГГц.
- Оперативная память объемом не менее 1 ГБ.
- Жесткий диск с доступным пространством не менее 2 ГБ.
- Монитор для отображения программы и визуализации орнаментов.
- Мышь и клавиатура для ввода данных и взаимодействия с программой.

3.5. Требования к информационной и программной совместимости

Для обеспечения совместимости программы необходимо наличие Visual Studio 2022 или более новой версии на операционной системе Windows 7 или выше.

3.6. Требования к маркировке и упаковке

Обычно требования к маркировке и упаковке не применимы к программе, поскольку она является цифровым продуктом, распространяемым в электронном формате.

3.7. Требования к транспортированию и хранению

Для правильной работы программы необходимо расположить соответствующие файлы на флеш-накопителе или в памяти компьютера. Рекомендуется сохранить программу на внешнем носителе, чтобы предотвратить потерю информации.

3.8. Специальные требования

Интерфейс программного обеспечения должен быть простым и понятным даже для пользователей средней квалификации.

4. Требования к программной документации

4.1. Содержание расчетно-пояснительной записки

Программная документация должна содержать расчётно-пояснительную записку с содержанием:

Титульный лист

Оглавление

Введение

Описание предметной области

Описание классов Graphics, Pen и Brush

Описание созданного приложения

1. Постановка задачи
2. Логическое проектирование
3. Физическое проектирование
4. Тестирование
5. Результаты работы

Заключение

Источники

Приложения

4.2. Требования к оформлению

Программная документация должна удовлетворять следующему оформлению (табл. П1.1):

Таблица П1.1

Требования к оформлению

Документ	Печать на отдельных листах формата А4 (210х297 мм); оборотная сторона не заполняется; листы нумеруются. Печать возможна ч/б. Файлы предъявляются на компакт-диске: РПЗ с ТЗ; программный код. Листы и диск в конверте вложены в пластиковую папку скоросшивателя.
Страницы	Ориентация – книжная; отдельные страницы, при необходимости, альбомная. Поля: верхнее, нижнее – по 2 см, левое – 3 см , правое – 1 см.
Абзацы	Межстрочный интервал – 1, перед и после абзаца – 0.
Шрифты	Кегль – 14. В таблицах шрифт 12. Шрифт листинга – 10 (возможно в 2 колонки).
Рисунки	Подписывается под ним по центру: Рис.Х. Название В приложениях: Рис.П1.3. Название
Таблицы	Подписывается: над таблицей, выравнивание по правому: «Таблица Х». В следующей строке по центру Название Надписи в «шапке» (имена столбцов, полей) – по центру. В теле таблицы (записи) текстовые значения – выровнены по левому краю, числа, даты – по правому.

5. Технико-экономические показатели

Технико-экономические показатели к данной программе не предъявляются.

6. Стадии и этапы разработки

Стадии и этапы разработки представлены в табл. П1.2:

Таблица П1.2

Стадии и этапы разработки

Наименование этапа разработки	Сроки разработки	Результат выполнения	Отметка о выполнении
Разработка ТЗ	до 15.05.2023	Разработанное ТЗ	
Разработка программы	25.04.2023-06.06.2023	Разработанная программа	
Разработка руководства пользователя	10.05.2023-25.05.2023	Разработанное руководство пользователя	
Разработка РПЗ	15.05.2023-06.06.2023	Разработанная РПЗ	

7. Порядок контроля и приемки

Порядок контроля и приемки представлен в табл. П1.3:

Таблица П1.3

Порядок контроля и приемки

Наименование контрольного этапа выполнения курсовой работы	Сроки контроля	Результат выполнения	Отметка о приемке результата контрольного этапа
Оформление ТЗ	15.05.2023	Оформленное ТЗ	
Разработка программы	20.05.2023	Неконечная версия программы	
Оформление руководство пользователя	25.05.2023	Оформленное руководство пользователя	
Доработка программы	06.06.2023	Конечная версия программы	
Оформление РПЗ	06.06.2023	Оформленная РПЗ	
Сдача РПЗ	09.06.2023	Оценка за курсовую работу	

Приложение 2. Руководство пользователя

1. Общие сведения о программе

Программа предназначена для графического изображения орнамента множества пересекающихся квадратов, вписанных в окружности (рис. П2.1).

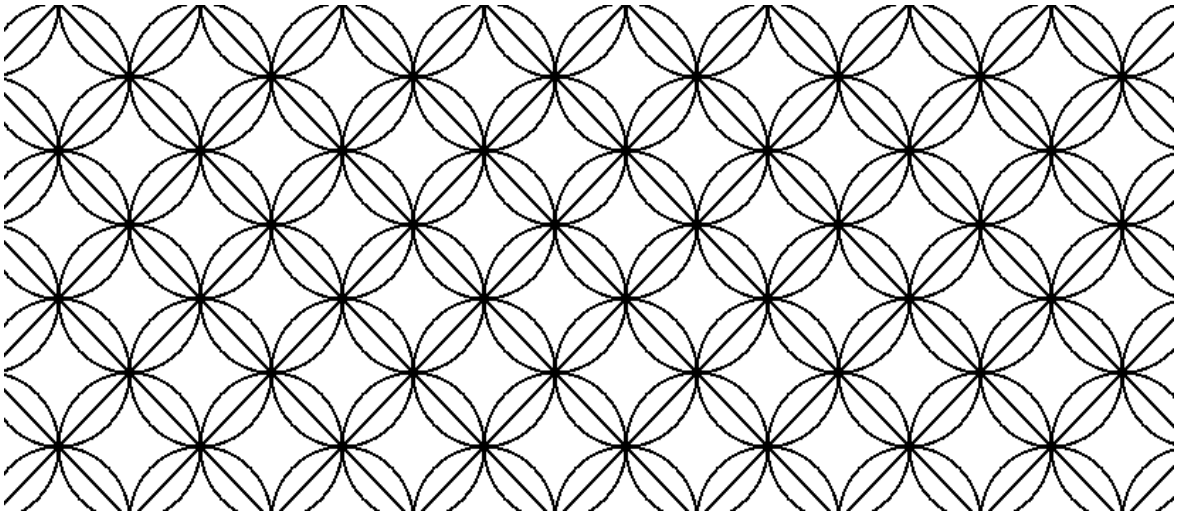

Рис. П2.1. Орнамент множества пересекающихся квадратов, вписанных в окружности

Программа создает орнамент по параметрам: масштаб, толщина линии и цвет изображения.

2. Описание установки

Для запуска программы необходимо иметь доступ к файлу программы и использовать программу Visual Studio 2022 или более поздней версии.
Установка программы не требуется.

3. Описание запуска

1. Необходимо открыть папку, в которой находится файл программы.
2. Запустить программу.
3. После запуска появится окно программы, готовое к использованию (см. рисунок П2.2).

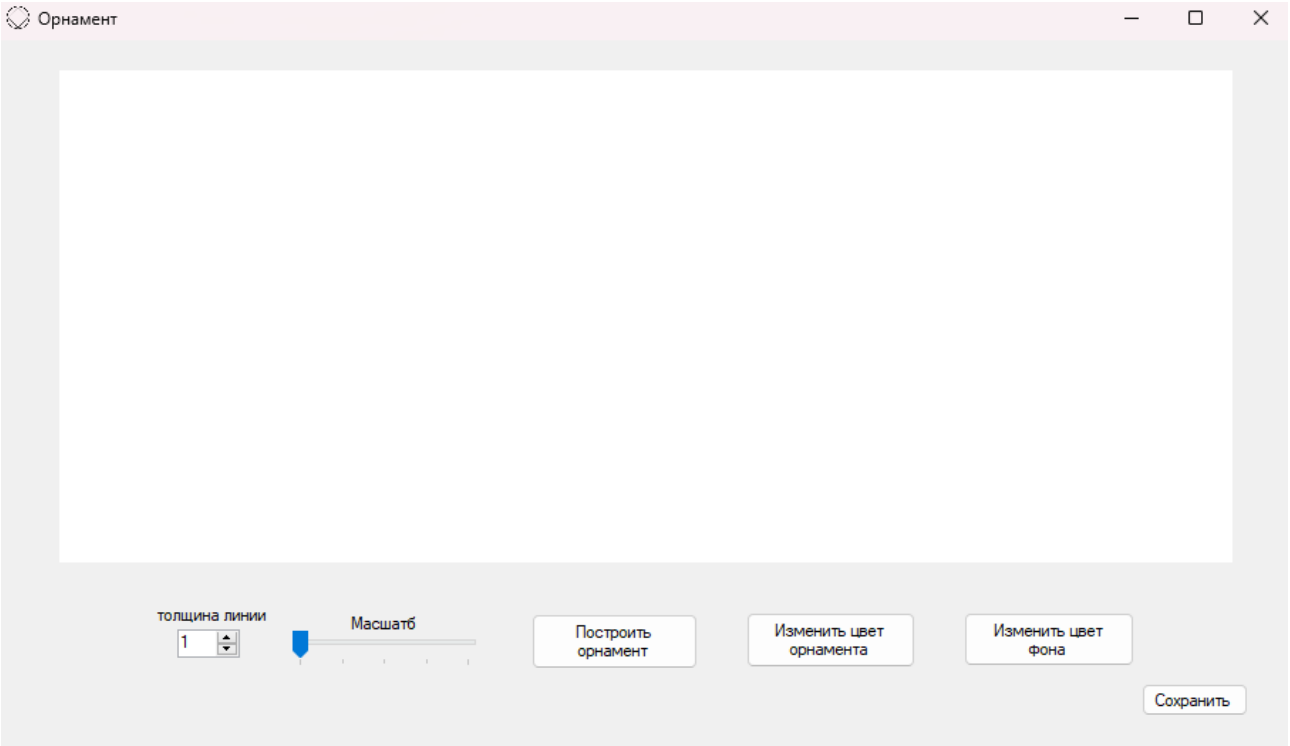

Рис. П2.2. Окно программы

4. Инструкция по работе

Параметры для построения орнамента могут быть заданы с помощью элементов интерфейса, указанных на рис. П2.3.

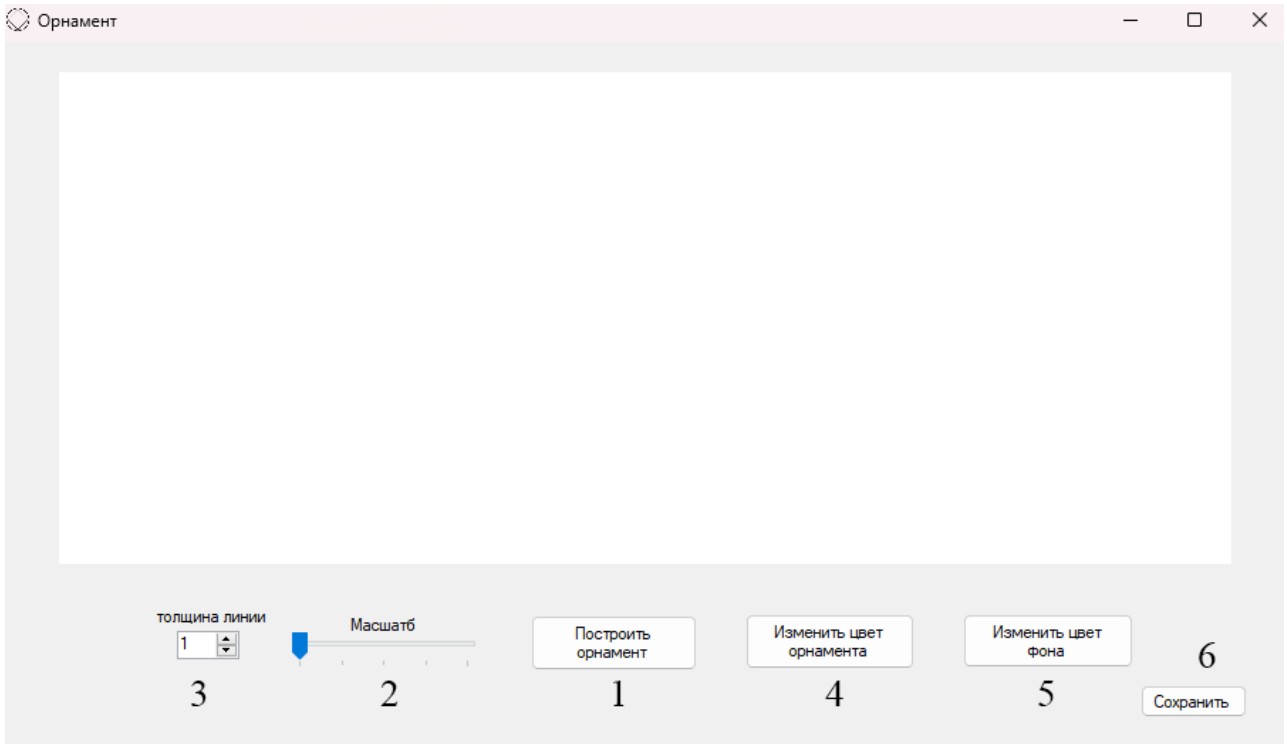

Рис. П2.3. Элементы управления

1. Построение изначального орнамента, масштаб орнамента по стандарту 1, толщина линии 1, цвет чёрный.

2. Выбор масштаба. Для выбора масштаба, нужно менять значение ползунка (от 1 до 5).

3. Выбор толщины линии спирали. Для выбора толщины линии можно ввести в соответствующее окно значение толщины линии, либо переключать значения с помощью треугольников и менять значения на 1 (от 1 до 10).

4. Выбор цвета орнамента. При нажатии на эту кнопку появляется диалоговое окно и при выборе необходимого цвета изменится значение цвета линии орнамента.

5. Выбор цвета фона. При нажатии на эту кнопку появляется диалоговое окно и при выборе необходимого цвета изменится значение цвета фона.

Сохранение изображения. Для сохранения изображения необходимо нажать на указанную кнопку. После этого появится диалоговое окно, где пользователь сможет выбрать путь сохранения файла и указать формат. После изменения масштаба, толщины и цвета необходимо нажать на кнопку «Построить орнамент», тогда программа построит орнамент (рис. П2.4) и после нажатия кнопки «Сохранить» она сохранит изображение (рис. П2.5).

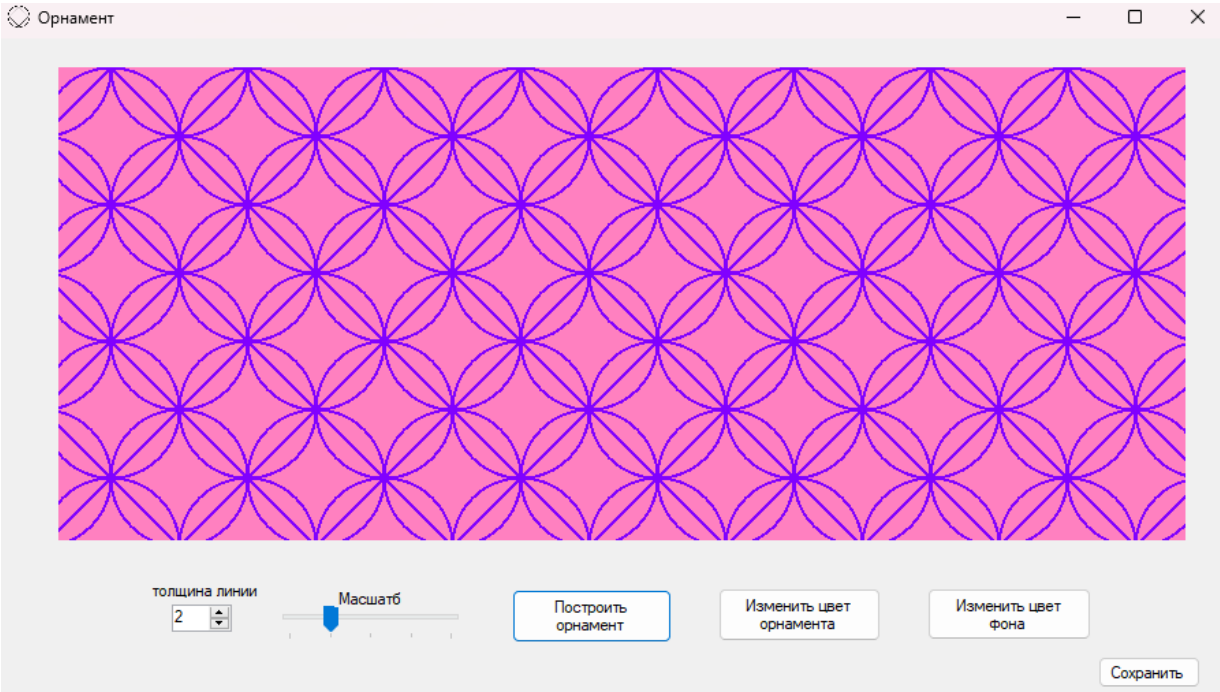

Рис. П2.4. Построенный орнамент

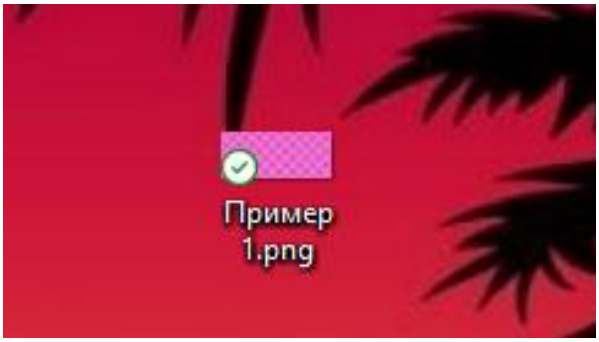

Рис. П2.5. Сохраненное изображение

5. Сообщения пользователю

При нажатии на кнопку «Изменить цвет орнамента» или «Изменить цвет фона» появится диалоговое окно, представленное на рис. П2.6:

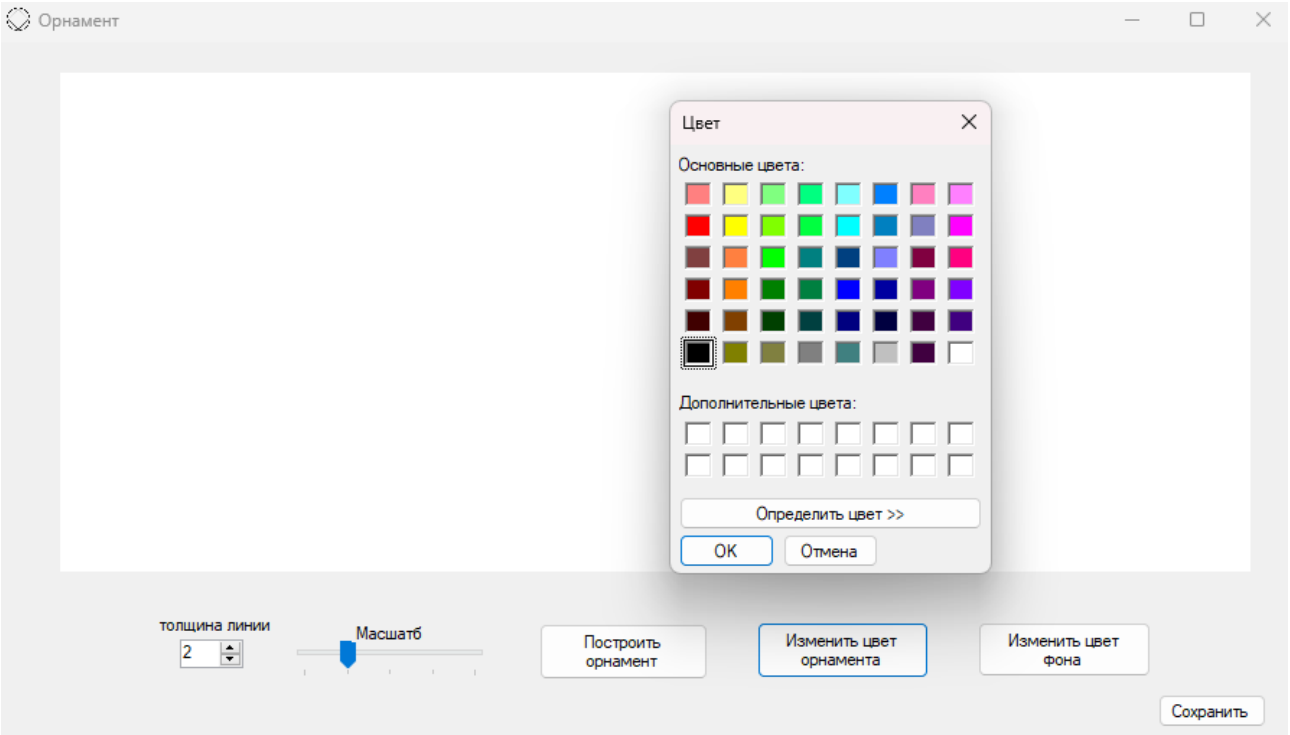

Рис. П2.6. Диалоговое окно выбора цвета

При нажатии на кнопку сохранить появится диалоговое окно, представленное на рис. П2.7:

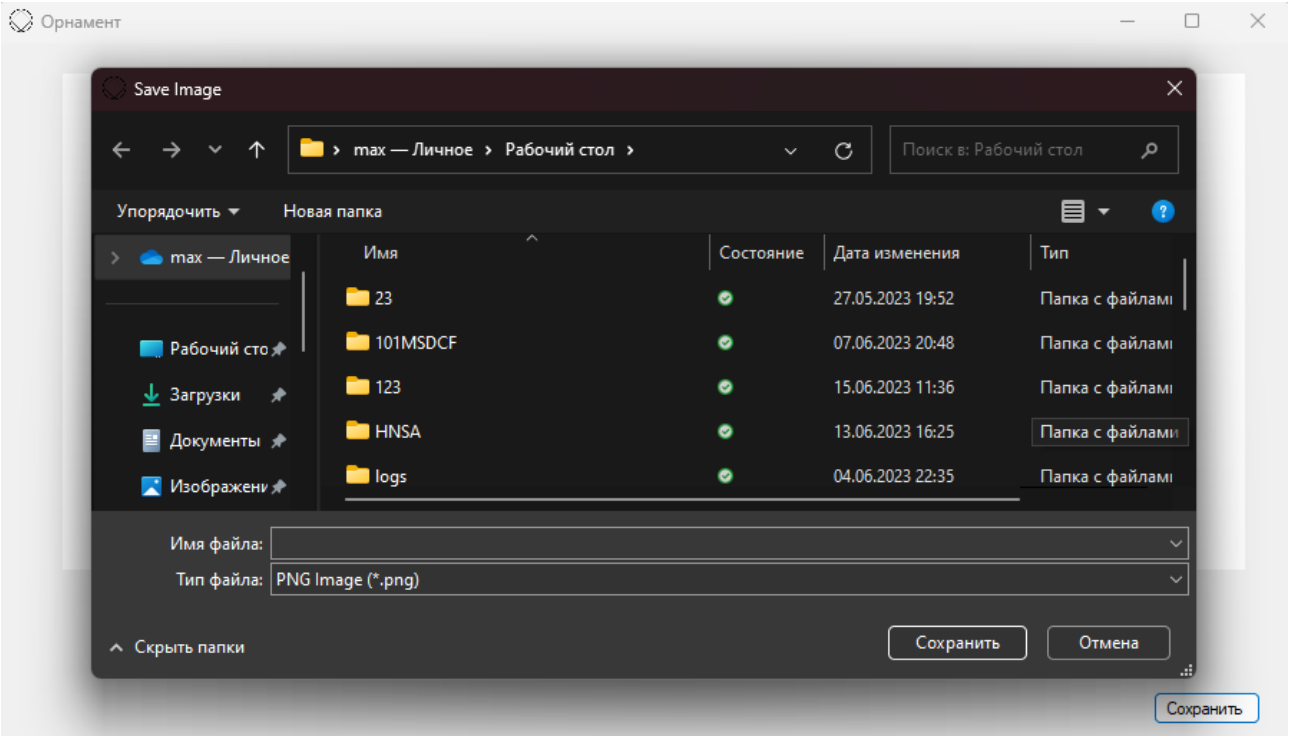

Рис. П2.7. Диалоговое окно сохранения изображения

Приложение 3. Программный код

Код MyForm.cpp:

```
#include "MyForm.h"

using namespace System;
using namespace System::Windows::Forms;
[STAThread]
int main(array<String^>^ args)
{
	Application::EnableVisualStyles();
	Application::SetCompatibleTextRenderingDefault(false);

	Project7::MyForm form;
	Application::Run(% form);
}
```

Код MyForm.h:

```
#pragma once
namespace Project7 {
	using namespace System;
	using namespace System::ComponentModel;
	using namespace System::Collections;
	using namespace System::Windows::Forms;
	using namespace System::Data;
	using namespace System::Drawing;

	/// <summary>
	/// Сводка для MyForm
	/// </summary>
	public ref class MyForm : public System::Windows::Forms::Form
	{
	public:
		MyForm(void)
		{
			InitializeComponent();
			//
			//TODO: добавьте код конструктора
			//
		}

	protected:
		/// <summary>
		/// Освободить все используемые ресурсы.
		/// </summary>
		~MyForm()
		{
			if (components)
			{
				delete components;
			}
		}
	private: System::Windows::Forms::Button^ button1;
	private: System::Windows::Forms::PictureBox^ pictureBox1;
	private: System::Windows::Forms::Button^ colorButton;
	private: System::Windows::Forms::TrackBar^ trackBar1;
	private: System::Windows::Forms::NumericUpDown^ numericUpDown1;

	private: System::Windows::Forms::Label^ label2;
	private: System::Windows::Forms::Label^ label3;
	private: System::Windows::Forms::Button^ button2;
	private: System::Windows::Forms::Button^ button3;
```

```
	protected:

	private:
		/// <summary>
		/// Обязательная переменная конструктора.
		/// </summary>
		System::ComponentModel::Container^ components;

#pragma region Windows Form Designer generated code
		/// <summary>
		/// Требуемый метод для поддержки конструктора — не изменяйте
		/// содержимое этого метода с помощью редактора кода.
		/// </summary>
		void InitializeComponent(void)
		{
			System::ComponentModel::ComponentResourceManager^ resources = (gcnew
System::ComponentModel::ComponentResourceManager(MyForm::typeid));
			this->button1 = (gcnew System::Windows::Forms::Button());
			this->pictureBox1 = (gcnew System::Windows::Forms::PictureBox());
			this->colorButton = (gcnew System::Windows::Forms::Button());
			this->trackBar1 = (gcnew System::Windows::Forms::TrackBar());
			this->numericUpDown1 = (gcnew System::Windows::Forms::NumericUpDown());
			this->label2 = (gcnew System::Windows::Forms::Label());
			this->label3 = (gcnew System::Windows::Forms::Label());
			this->button2 = (gcnew System::Windows::Forms::Button());
			this->button3 = (gcnew System::Windows::Forms::Button());
			(cli::safe_cast<System::ComponentModel::ISupportInitialize^>(this->pictureBox1))-
>BeginInit();
			(cli::safe_cast<System::ComponentModel::ISupportInitialize^>(this->trackBar1))-
>BeginInit();
			(cli::safe_cast<System::ComponentModel::ISupportInitialize^>(this->numericUpDown1))-
>BeginInit();
			this->SuspendLayout();
			//
			// button1
			//
			this->button1->Location = System::Drawing::Point(377, 403);
			this->button1->Name = L"button1";
			this->button1->Size = System::Drawing::Size(117, 39);
			this->button1->TabIndex = 0;
			this->button1->Text = L"Построить орнамент";
			this->button1->UseVisualStyleBackColor = true;
			this->button1->Click += gcnew System::EventHandler(this, &MyForm::button1_Click);
			//
			// pictureBox1
			//
			this->pictureBox1->BackColor = System::Drawing::SystemColors::ControlLightLight;
			this->pictureBox1->ErrorImage = (cli::safe_cast<System::Drawing::Image^>(resources-
>GetObject(L"pictureBox1.ErrorImage")));
			this->pictureBox1->Location = System::Drawing::Point(45, 21);
			this->pictureBox1->Name = L"pictureBox1";
			this->pictureBox1->Size = System::Drawing::Size(825, 346);
			this->pictureBox1->TabIndex = 1;
			this->pictureBox1->TabStop = false;
			this->pictureBox1->Click += gcnew System::EventHandler(this,
&MyForm::pictureBox1_Click);
			//
			// colorButton
			//
			this->colorButton->Location = System::Drawing::Point(528, 402);
			this->colorButton->Name = L"colorButton";
			this->colorButton->RightToLeft = System::Windows::Forms::RightToLeft::No;
			this->colorButton->Size = System::Drawing::Size(119, 39);
```

```
                this->colorButton->TabIndex = 2;
                this->colorButton->Text = L"Изменить цвет орнамента\r\n";
                this->colorButton->UseVisualStyleBackColor = true;
                this->colorButton->Click += gcnew System::EventHandler(this,
&MyForm::colorButton_Click);
                //
                // trackBar1
                //
                this->trackBar1->Location = System::Drawing::Point(201, 413);
                this->trackBar1->Maximum = 5;
                this->trackBar1->Minimum = 1;
                this->trackBar1->Name = L"trackBar1";
                this->trackBar1->Size = System::Drawing::Size(145, 45);
                this->trackBar1->TabIndex = 3;
                this->trackBar1->Value = 1;
                this->trackBar1->Scroll += gcnew System::EventHandler(this,
&MyForm::trackBar1_Scroll);
                //
                // numericUpDown1
                //
                this->numericUpDown1->Location = System::Drawing::Point(128, 414);
                this->numericUpDown1->Maximum = System::Decimal(gcnew cli::array< System::Int32
>(4) { 10, 0, 0, 0 });
                this->numericUpDown1->Name = L"numericUpDown1";
                this->numericUpDown1->Size = System::Drawing::Size(44, 20);
                this->numericUpDown1->TabIndex = 4;
                this->numericUpDown1->Value = System::Decimal(gcnew cli::array< System::Int32 >(4) {
1, 0, 0, 0 });
                this->numericUpDown1->ValueChanged += gcnew System::EventHandler(this,
&MyForm::numericUpDown1_ValueChanged);
                //
                // label2
                //
                this->label2->AutoSize = true;
                this->label2->Location = System::Drawing::Point(247, 403);
                this->label2->Name = L"label2";
                this->label2->Size = System::Drawing::Size(53, 13);
                this->label2->TabIndex = 6;
                this->label2->Text = L"Масшатб";
                this->label2->Click += gcnew System::EventHandler(this, &MyForm::label2_Click);
                //
                // label3
                //
                this->label3->AutoSize = true;
                this->label3->Location = System::Drawing::Point(111, 397);
                this->label3->Name = L"label3";
                this->label3->Size = System::Drawing::Size(84, 13);
                this->label3->TabIndex = 7;
                this->label3->Text = L"толщина линии";
                this->label3->Click += gcnew System::EventHandler(this, &MyForm::label3_Click);
                //
                // button2
                //
                this->button2->Location = System::Drawing::Point(681, 402);
                this->button2->Name = L"button2";
                this->button2->RightToLeft = System::Windows::Forms::RightToLeft::No;
                this->button2->Size = System::Drawing::Size(120, 38);
                this->button2->TabIndex = 8;
                this->button2->Text = L"Изменить цвет фона";
                this->button2->UseVisualStyleBackColor = true;
                this->button2->Click += gcnew System::EventHandler(this, &MyForm::button2_Click);
                //
                // button3
```

```
			//
			this->button3->Location = System::Drawing::Point(806, 452);
			this->button3->Name = L"button3";
			this->button3->Size = System::Drawing::Size(75, 23);
			this->button3->TabIndex = 9;
			this->button3->Text = L"Сохранить";
			this->button3->UseVisualStyleBackColor = true;
			this->button3->Click += gcnew System::EventHandler(this, &MyForm::button3_Click);
			//
			// MyForm
			//
			this->AutoScaleDimensions = System::Drawing::SizeF(6, 13);
			this->AutoScaleMode = System::Windows::Forms::AutoScaleMode::Font;
			this->ClientSize = System::Drawing::Size(902, 487);
			this->Controls->Add(this->button3);
			this->Controls->Add(this->button2);
			this->Controls->Add(this->label3);
			this->Controls->Add(this->label2);
			this->Controls->Add(this->numericUpDown1);
			this->Controls->Add(this->trackBar1);
			this->Controls->Add(this->colorButton);
			this->Controls->Add(this->pictureBox1);
			this->Controls->Add(this->button1);
			this->Icon = (cli::safe_cast<System::Drawing::Icon^>(resources-
>GetObject(L"$this.Icon")));
			this->Name = L"MyForm";
			this->Text = L"Орнамент";
			this->Load += gcnew System::EventHandler(this, &MyForm::MyForm_Load);
			(cli::safe_cast<System::ComponentModel::ISupportInitialize^>(this->pictureBox1))-
>EndInit();
			(cli::safe_cast<System::ComponentModel::ISupportInitialize^>(this->trackBar1))-
>EndInit();
			(cli::safe_cast<System::ComponentModel::ISupportInitialize^>(this->numericUpDown1))-
>EndInit();
			this->ResumeLayout(false);
			this->PerformLayout();

		}
#pragma endregion

		void DrawCircle(Graphics^ graphics, Pen^ pen, int centerX, int centerY, int radius)
		{
			int x = centerX - radius;
			int y = centerY - radius;
			int diameter = radius * 2;
			graphics->DrawEllipse(pen, x, y, diameter, diameter);

			// Расчет координат для квадрата
			int topX = centerX;
			int topY = centerY - radius;
			int rightX = centerX + radius;
			int rightY = centerY;
			int bottomX = centerX;
			int bottomY = centerY + radius;
			int leftX = centerX - radius;
			int leftY = centerY;
			// Рисование ромба без отображения точек
			array<Point>^ diamondPoints = gcnew array<Point>(4);
			diamondPoints[0] = Point(topX, topY);
			diamondPoints[1] = Point(rightX, rightY);
			diamondPoints[2] = Point(bottomX, bottomY);
			diamondPoints[3] = Point(leftX, leftY);
			graphics->DrawPolygon(pen, diamondPoints);
```

```
                }
                void OrnamentDraw(int size, Color color, int WL) {
    Graphics^ graphics = pictureBox1->CreateGraphics();
    Pen^ pen = gcnew Pen(color);
        pen->Width = WL; // Толщина линии
    // Размеры PictureBox
    int width = pictureBox1->Width;
    int height = pictureBox1->Height;

    // Размер каждого элемента орнамента
    int elementSize = size / 2;

    // Количество элементов в ряду
    int elementsPerRow = width / elementSize;

    // Расстояние между рядами (уменьшено на 3 радиуса)
    int rowSpacing = elementSize - (3 * (elementSize / 2));

    // Позиция начала первого ряда
    int startX = (width - ((elementsPerRow + 1.5)* elementSize)) / 2;
    int startY = -(elementSize*2); // Ряд выходит за пределы экрана сверху

    int rowY = startY;
    int rowX = startX;

    int numRows = (height + elementSize) / (elementSize + rowSpacing); // Количество рядов, чтобы заполнить всю
высоту

    for (int row = 0; row < numRows* 2; row++) {
        for (int i = 0; i < elementsPerRow * 2; i++) {
            // Рисуем элемент орнамента в текущей позиции
            int centerX = rowX + (elementSize / 2);
            int centerY = rowY + (elementSize / 2);
            int radius = elementSize / 2;

            // Рисуем круг
            DrawCircle(graphics, pen, centerX, centerY, radius);

            // Обновляем позицию для следующего элемента в ряду
            rowX += elementSize;
        }

        // Корректируем позицию для следующего ряда
        rowX = startX;
        rowY += elementSize + rowSpacing;

        // Переключение между типами рядов (1 и 2)
        if (row % 2 != 0) {
            // Нечетный ряд
            rowX += elementSize / 2;
        }
    }
}
            private: Color f = Color::Black;
            private: Color bf = Color::White;
            private: int size = 100;
            private: double WL = 1;
            private: System::Void pictureBox1_Click(System::Object^ sender, System::EventArgs^ e) {
            }
            private: System::Void trackBar1_Scroll(System::Object^ sender, System::EventArgs^ e) {
                    // Получаем текущее значение ползунка
                    int value = trackBar1->Value;
```

```
		// Вычисляем новое значение переменной size в диапазоне от 100 до 500
		size = value * 100;
	}
	private: System::Void button1_Click(System::Object^ sender, System::EventArgs^ e) {
		// Цвет фона
		pictureBox1->BackColor = bf;
		// Очистка pictureBox1
		 pictureBox1->Refresh();

		int centerX = pictureBox1->Width / 2; // Горизонтальная позиция центра окружности
		int centerY = pictureBox1->Height / 2; // Вертикальная позиция центра окружности
		int radius = 50; // Радиус окружности
		OrnamentDraw(size, f, WL);
	}
	private: System::Void MyForm_Load(System::Object^ sender, System::EventArgs^ e) {
	}
	private: System::Void colorButton_Click(System::Object^ sender, System::EventArgs^ e) {
	// Создаем экземпляр ColorDialog
	ColorDialog^ colorDialog = gcnew ColorDialog();

	// Показываем диалоговое окно выбора цвета
	if (colorDialog->ShowDialog() == System::Windows::Forms::DialogResult::OK) {
		// Получаем выбранный цвет и присваиваем его полю f
		f = colorDialog->Color;
	}
}


	private: System::Void label2_Click(System::Object^ sender, System::EventArgs^ e) {
	}
	private: System::Void label3_Click(System::Object^ sender, System::EventArgs^ e) {
}
	private: System::Void numericUpDown1_ValueChanged(System::Object^ sender, System::EventArgs^ e) {
	// Присваивание нового значения переменной WL
	WL = System::Convert::ToDouble(numericUpDown1->Value);
}
	private: System::Void button2_Click(System::Object^ sender, System::EventArgs^ e) {
	// Создаем экземпляр ColorDialog
	ColorDialog^ colorDialog = gcnew ColorDialog();

	// Показываем диалоговое окно выбора цвета
	if (colorDialog->ShowDialog() == System::Windows::Forms::DialogResult::OK) {
		// Получаем выбранный цвет и присваиваем его полю f
		bf = colorDialog->Color;
	}
}
	private: System::Void button3_Click(System::Object^ sender, System::EventArgs^ e) {
	// Определяем область, которую хотим сохранить (в данном случае - область pictureBox)
	Rectangle rect = pictureBox1->Bounds;
	// Создаем Bitmap с размерами области
	Bitmap^ bitmap = gcnew Bitmap(rect.Width, rect.Height);
	// Создаем Graphics для Bitmap
	Graphics^ graphics = Graphics::FromImage(bitmap);
	// Копируем область pictureBox на Graphics
	Point location = pictureBox1->PointToScreen(Point(0, 0));
	graphics->CopyFromScreen(location, Point(0, 0), rect.Size);
	// Освобождаем ресурсы Graphics
	delete graphics;
	// Открываем диалоговое окно для выбора места сохранения файла
	SaveFileDialog^ saveFileDialog = gcnew SaveFileDialog();
	saveFileDialog->Filter = "PNG Image (*.png)|*.png|JPEG Image(*.jpg)|*.jpg|Bitmap Image(*.bmp)|*.bmp";
	saveFileDialog->Title = "Save Image";
	if (saveFileDialog->ShowDialog() == System::Windows::Forms::DialogResult::OK) {
```

```
                // Если пользователь выбрал место сохранения файла
                if (saveFileDialog->FileName != "") {
                        // Сохраняем Bitmap в выбранном месте
                        bitmap->Save(saveFileDialog->FileName, Imaging::ImageFormat::Png);
                }
        }
        // Освобождаем ресурсы Bitmap
        delete bitmap;

}
};
}
```