МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего 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* |
|  | Воронин Илья Евгеньевич |
|  | *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 3](#_Toc137147273)

[1.Описание предметной области 4](#_Toc137147274)

[2.Описание классов Graphics, Pen и Brush 5](#_Toc137147275)

[2.1. Класс Graphics 5](#_Toc137147276)

[2.2. Класс Pen 6](#_Toc137147277)

[2.3. Класс Brush 7](#_Toc137147278)

[3.Описание созданного приложения 7](#_Toc137147279)

[3.1. Постановка задачи 7](#_Toc137147280)

[3.2. Логическое проектирование 8](#_Toc137147281)

[3.3. Физическое проектирование 12](#_Toc137147282)

[4.4. Тестирование 13](#_Toc137147283)

[4.5. Результаты работы 14](#_Toc137147284)

[Заключение 14](#_Toc137147285)

[Источники 15](#_Toc137147286)

[Приложение 1. Техническое задание 16](#_Toc137147287)

[Приложение 2. Руководство пользователя 23](#_Toc137147288)

[Приложение 3. Программный код 28](#_Toc137147289)

Введение

Программирование на языке высокого уровня, является одной из ключевых областей современной информатики, позволяющей разработчикам создавать сложные и функциональные программы с помощью абстрактных и удобных для восприятия инструментов. В отличие от низкоуровневых языков программирования, он может использовать элементы естественного языка, быть более простым в использовании или может автоматизировать значительные области вычислительных систем таких как, управление памятью, делая процесс разработки программы проще и понятнее, чем при использовании языка более низкого уровня [1].

Язык С++ является одним из самых популярных высокоуровневых языков программирования. Он является развитием языка С и добавляет возможности объектно-ориентированного программирования, что делает его мощным инструментом для создания сложных программных систем [2].

Задача данной курсовой работы заключается в разработке программы, способной генерировать графическое изображение орнамента и обеспечивать возможность изменения его размера и цвета, а также реализовывать сохранение получившегося графического изображения.

Разработанная программа, способная создавать и изменять орнамент, позволит пользователям создавать уникальные и привлекательные графические элементы, придавая им индивидуальный стиль.

1.Описание предметной области

Предметная область данной курсовой работы связана с разработкой программы для создания орнаментов в Microsoft Forms. Орнаменты играют важную роль в дизайне, добавляя украшение, стиль и эстетическую ценность к графическим элементам и интерфейсам приложений.

Windows Forms — это платформа пользовательского интерфейса для создания классических приложений Windows. Она обеспечивает один из самых эффективных способов создания классических приложений с помощью визуального конструктора в Visual Studio. Такие функции, как размещение визуальных элементов управления путем перетаскивания, упрощают создание классических приложений [3].

Microsoft Forms представляет собой инструмент для разработки форм и интерфейсов приложений, который предоставляет возможность создавать пользовательские элементы управления, добавлять графические элементы и настраивать их свойства. Разработка программы для создания орнаментов в Microsoft Forms позволит пользователям создавать уникальный орнамент, настраивать его параметры, масштабировать и менять цвета. Вместо ручного рисования орнаментов или использования сложных графических инструментов, программа позволит пользователю выбирать и настраивать орнаменты с помощью удобного интерфейса. Таким образом, разработка программы значительно упрощает и ускоряет процесс создания орнаментов и повышает эффективность работы с графическими элементами в Microsoft Forms.

Предметной областью является орнамент множества пересекающихся шестиугольников (рис.1).

Орнамент представляет из себя множество шестиугольников, в которых нарисованы шестиугольники меньшего размера, которые определённым образом пересекаются между собой.

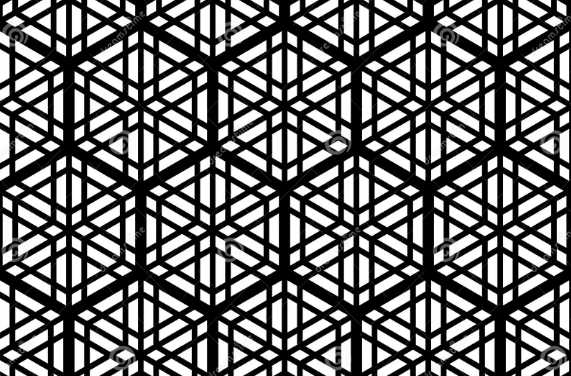


Рис.1. Орнамент множества пересекающихся шестиугольников

2.Описание классов Graphics, Pen и Brush

Рисование фигур является одним из важных аспектов в разработке графических приложений. В Windows Forms это может быть реализовано с помощью классов Graphics, Pen и Brush в языке программирования C++. Пространство имен System.Drawing (Рисование) обеспечивает доступ к функциональным возможностям графического интерфейса GDI+ , используя около 50 классов, в том числе класс Graphics, Pen и Brush.

2.1. Класс Graphics

Класс Graphics предоставляет методы рисования на устройстве отображения. Данный класс входит в пространство имен System.Drawing, как и большинство классов для работы с графикой. Класс Graphics определяет набор методов для вывода текста, изображений и геометрических фигур [4].

Методов в этом классе огромное количество, поэтому рассмотрим некоторую часть из них в табл. 1:

Таблица 1

Методы класса Graphics

|  |  |
| --- | --- |
| Имя | Описание |
| AddMetafileComment | Добавляет комментарий к текущему объекту Metafile. |
| BeginContainer() | Сохраняет графический контейнер, содержащий текущее состояние данного объекта Graphics, а затем открывает и использует новый графический контейнер. |
| Clear | Очищает всю поверхность рисования и выполняет заливку поверхности указанным цветом фона. |
| CopyFromScreen(Int32, Int32, Int32, Int32, Size) | Выполняет передачу данных о цвете, соответствующих прямоугольной области пикселей, блоками битов с экрана на поверхность рисования объекта Graphics. |
| Dispose | Освобождает все ресурсы, используемые данным объектом Graphics. |
| DrawArc(Pen, Int32, Int32, Int32, Int32, Int32, Int32) | Рисует дугу, которая является частью эллипса, заданного парой координат, шириной и высотой. |
| DrawBezier(Pen, Point, Point, Point, Point) | Рисует кривую Безье, определяемую четырьмя структурами Point. |
| DrawLine(Pen, Int32, Int32, Int32, Int32) | Проводит линию, соединяющую две точки, задаваемые парами координат. |
| FillClosedCurve(Brush, Point[], FillMode) | Заполняет внутреннюю часть замкнутой фундаментальной сплайновой кривой, определяемой массивом структур Point, используя указанный режим заливки. |
| FillEllipse(Brush, Int32, Int32, Int32, Int32) | Заполняет внутреннюю часть эллипса, определяемого ограничивающим прямоугольником, заданным с помощью пары координат, ширины и высоты. |
| FillPath | Заполняет внутреннюю часть объекта GraphicsPath. |
| FillPie(Brush, Rectangle, Single, Single) | Заполняет внутреннюю часть сектора, определяемого эллипсом, который задан структурой RectangleF, и двумя радиальными линиями. |
| FillPie(Brush, Int32, Int32, Int32, Int32, Int32, Int32) | Заполняет внутреннюю часть сектора, определяемого эллипсом, который задан парой координат, шириной, высотой и двумя радиальными линиями. |
| FillPolygon(Brush, Point[]) | Заполняет внутреннюю часть многоугольника, определяемого массивом точек, заданных структурами Point. |
| FillRectangle(Brush, Rectangle) | Заполняет внутреннюю часть прямоугольника, определяемого структурой Rectangle. |

2.2. Класс Pen

Класс Pen определяет объект, используемый для рисования прямых линий и кривых [4].

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 определяет объекты, которые используются для заливки внутри графических фигур, таких как прямоугольники, эллипсы, круги, многоугольники и дорожки.

Это абстрактный базовый класс, который не может быть реализован. Для создания объекта "кисть" используются классы, производные от Brush, такие как SolidBrush, TextureBrush и LinearGradientBrush [4].

3.Описание созданного приложения

В данном разделе описаны основные этапы разработки приложения для построения орнамента пересекающихся шестиугольников.

3.1. Постановка задачи

Для корректной работы программы требуется выполнить следующие задачи:

* Программа должна создавать графическое изображение орнамента при нажатии соответствующей кнопки;
* программа должна позволять выбирать цвет заднего фона;
* программа должна позволять менять масштаб и цвет орнамента;
* программа должна позволять сохранять в файловом виде изображение при нажатии соответствующей кнопки;
* программа должна иметь возможность повторно создать графическое изображение при изменении масштаба и цвета.

3.2. Логическое проектирование

Для построения орнамента (рис.1) необходимо определиться с составными частями орнамента. Для его реализации нам понадобится 2 функции: функция «А», которая рисует большой шестиугольник и функция «Б», которая рисует меньший шестиугольник внутри большего. Эти функции будут вызываться общей функцией, которая предназначена для рисования всего орнамента.

Общая функция построения орнамента принимает значение размера элемента орнамента, то есть шестиугольника, и цвет, которым будет реализовано рисование, эти параметры предусмотрены для того, чтобы можно было реализовать изменение масштаба и цвета рисунка. Затем определяются значения ширины и высоты картинки исходя из размеров окна приложения, значение количества шестиугольников путём отношения ширины окна, на котором происходит рисование, к половине размера шестиугольника, это необходимо для того, чтобы шестиугольники находились вплотную друг к другу. Задаются начальные координаты по оси х и по оси у, равные нулю. Позже вычисляется количество рядов шестиугольников по оси у отношением высоты окна, на котором будет производится реализация рисунка, к половине размера шестиугольника. Эти шестиугольники будут рисоваться сначала по оси х, затем дублироваться по оси у. Для начала в функции необходимо создать цикл, который будет выполняться пока переменная, отвечающая за счётчик цикла, меньше количества рядов по оси у, которое было вычислено ранее. В теле цикла задаются координаты для ряда по оси у, путём суммы начальных координат и разности половины размера шестиугольника с одной четырнадцатой размера шестиугольника, координаты вычисляются таким образом, для того чтобы не было отступов между рядами шестиугольников. Для правильного пересечения рядов шестиугольников нам необходимо условие, которое будет изменять координаты ряда по оси х, прибавляя сумму половины размера шестиугольника и четверти его размера. Тем самым координаты начала каждого второго ряда будут находится так, что верхняя вершина правильного шестиугольника окажется в центре предыдущего шестиугольника. Далее эта функция вызывает функцию «А», которая реализует рисование ряда больших шестиугольников, в которых будут находится маленькие. Но в конце этой функции необходимо вернуть координаты ряда по оси х, если этот ряд чётный.

Функция «А» принимает значения общего количества шестиугольников в ряду, размер шестиугольника, координаты ряда по оси х и у, поверхность для рисования, и объект для рисования, и цвет, которым будут рисоваться шестиугольники. Для реализации рисования ряда шестиугольников по оси х, необходим цикл, который будет выполняться пока счётчик цикла меньше общего количества шестиугольников в ряду. В цикле вычисляется координата х, для текущего шестиугольника через сумму координаты ряда и произведения половины размера и переменной счётчика, тем самым каждый шестиугольник будет вплотную находится с предыдущим. Для построения правильного шестиугольника нам необходимо создать массив шести точек. Заполнение этого массива происходит по формулам: angle = 2 \* Math::PI / 6 \* j; pointX = x + (int)(size / 2 \* Math::Cos(angle)); pointY = rowY + (int)(size / 2 \* Math::Sin(angle)). Где angle – угол, то есть точка на числовой окружности, pointX, pointY – значения координат точки вершины по оси х и у соответственно, size – размер шестиугольника , x – координата х центра текущего шестиугольника , rowY– координата у, текущего ряда шестиугольников. Обратив внимание на исходный рисунок, можно заметить, что каждый третий большой шестиугольник имеет большую толщину, чем другие. Для реализации этого, необходимо условие которое будет рисовать каждый третий шестиугольник с толщиной 3 по массиву точек вычисленному ранее, иначе рисуется обычный шестиугольник по массиву точек. Затем эта функция должна вызвать функцию «Б», которая будет рисовать меньший шестиугольник.

Функция «Б» должна принимать значения размера шестиугольника, координаты ряда по оси у, поверхность для рисования, и объект, используемый для рисования. В теле функции рассчитывается размер малого шестиугольника, он будет на 20% меньше большого. Далее, как и в функции «А», вычисляется массив точек по тем же формулам, однако вместо переменной размера шестиугольника, применяется вычисленная ранее переменная, означающая размер малого шестиугольника. Тем самым, координаты центров двух этих шестиугольников совпадают, и малый находится в центре большого.

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

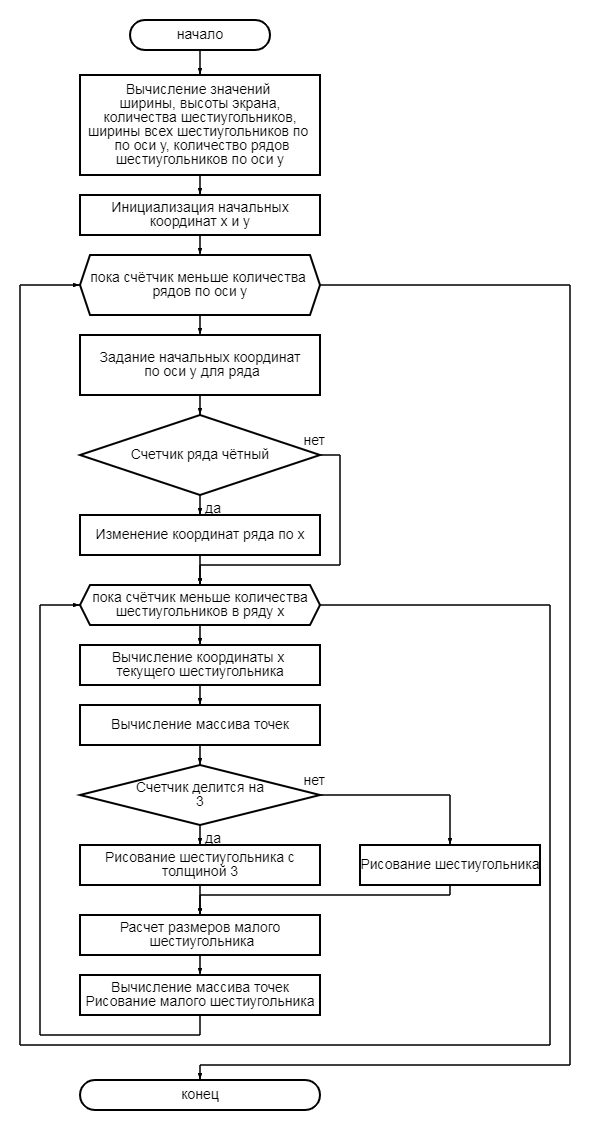


Рис. 2. Алгоритм работы программы в блок-схеме

3.3. Физическое проектирование

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

Таблица 3

Переменные

|  |  |  |
| --- | --- | --- |
| Наименование | Обозначение | Тип данных |
| Значение размера шестиугольника | size | int |
| Ширина блока рисования | width | int |
| Высота блока рисования | height | int |
| Количество шестиугольников по оси х | totalHexagons | int |
| Начальная координата х | startX | int |
| Начальная координата у | startY | int |
| Количество шестиугольников по оси у | numHexagonY | int |
| Координата ряда шестиугольников по оси х | rowX | int |
| Координата ряда шестиугольников по оси у | rowY | int |
| Значение угла | angle | double |
| Координаты вершины шестиугольника по оси х | pointX | int |
| Координаты вершины шестиугольника по оси у | pointY | int |
| Размер малого шестиугольника | smallSize | int |
| Координата х центра малого шестиугольника | smallX | int |
| Координата у центра малого шестиугольника | smallY | int |
| Координата х центра большого шестиугольника | x | Int |
| Цвет линий орнамента | color | Color |

Спецификация функций представлена в табл. 4:

Таблица 4

Спецификация функций

|  |  |  |  |
| --- | --- | --- | --- |
| Имя модуля | Заголовок процедуры или функции | Формальные параметры | Выполняемое действие |
| kukimo.sln | button1\_Click | System::Object^ sender, System::EventArgs^ e | Функция выполняет вызов функции построения орнамента. |
|  | SaveButton\_Click | System::Object^ sender, System::EventArgs^ e | Функция сохраняет полученное изображение. |
|  | numericUpDown1\_ValueChanged | System::Object^ sender, System::EventArgs^ e | Функция задаёт масштаб шестиугольника и обновляет поле для рисования. |
|  | RandomColorButton\_Click | System::Object^ sender, System::EventArgs^ e | Функция задаёт орнаменту рандомный цвет и строит орнамент с новым цветом. |
|  | colorButton\_Click | System::Object^ sender, System::EventArgs^ e | Функция вызывает диалоговое окно для выбора пользователю цвета орнамента и вызывает функцию построения орнамента. |
|  | button2\_Click | System::Object^ sender, System::EventArgs^ e | Функция меняет цвет заднего фона. |
|  | SmallHexagonDraw | int size, int x, int rowY, Graphics^ graghics, Pen^ pen | Функция рисует малый шестиугольник внутри большого |
|  | BigHexagonDraw | int totalHexagons,int size, int rowX, int rowY, Graphics^ graghics, Pen^ pen, Color color | Функция рисует ряд больших шестиугольников и вызывает функцию для рисования маленьких шестиугольников. |
|  | OrnamentDraw | int size, Color color | Функция рисует полноценный орнамент. |

4.4. Тестирование

Результаты тестирования представлены в табл. 5:

Таблица 5

Результаты тестирования

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Дата тестирования | Тестируемый модуль | Кто проводил тестирование | Описание теста | Результат тестирования |
| 20.05.2023 | kukimo.sln | Воронин И.Е. | Проверка работы кнопки построения орнамента | Успех |
| 21.05.2023 | kukimo.sln | Удальцов А.П. | Проверка работы сохранения изображения орнамента | Успех |
| 22.05.2023 | kukimo.sln | Воронин И.Е. | Изменение цвета | Успех |
| 22.05.2023 | kukimo.sln | Райков А.Р. | Масштабирование | Успех |
| 25.05.2023 | kukimo.sln | Цикин Т.И. | Изменение фона | Успех |
| 30.05.2023 | kukimo.sln | Воронин И.Е. | Проверка всех функций работы программы | Успех |

4.5. Результаты работы

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

Интерфейс понятен и функционален (рис. 3).

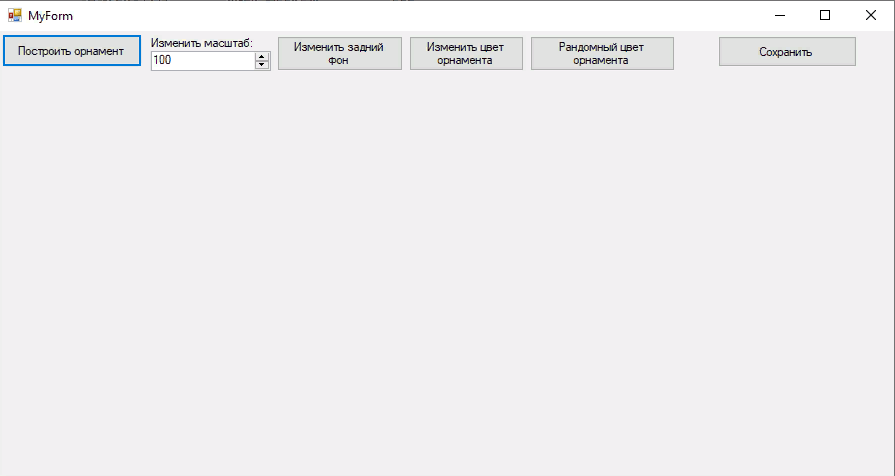


Рис. 3. Интерфейс программы

Заключение

В рамках данного проекта была разработана программа в Windows Forms на языке С++, реализующая графическое изображение орнамента множества пересекающихся шестиугольников.

В ходе работы, были получены навыки в работе с Windows Forms, освоены возможности классов Graphics, Pen и Brush.

Поставленные задачи были выполнены в полном объеме.

Источники

* 1. Высокоуровневые языки программирования [электронный ресурс]: https://en.wikipedia.org/wiki/High-level\_programming\_language. Дата доступа: 20.05.2023
  2. Язык программирования С++ [электронный ресурс]: https://metanit.com/cpp/tutorial/1.1.php Дата доступа: 21.05.2023
  3. Windows Forms [электронный ресурс]: https://learn.microsoft.com/ru-ru/dotnet/desktop/winforms/overview/?view=netdesktop-6.0 Дата доступа: 21.06.2023
  4. Основы библиотеки System.Drawing [электронный ресурс]: https://c-sharp.pro/общие-замечания-классы-graphics-pen-и-brush/. Дата доступа: 01.06.2023

Приложение 2. Руководство пользователя

1. Общие сведения о программе

Программа предназначена для графического изображения орнамента множества пересекающихся шестиугольников (рис. П2.1).

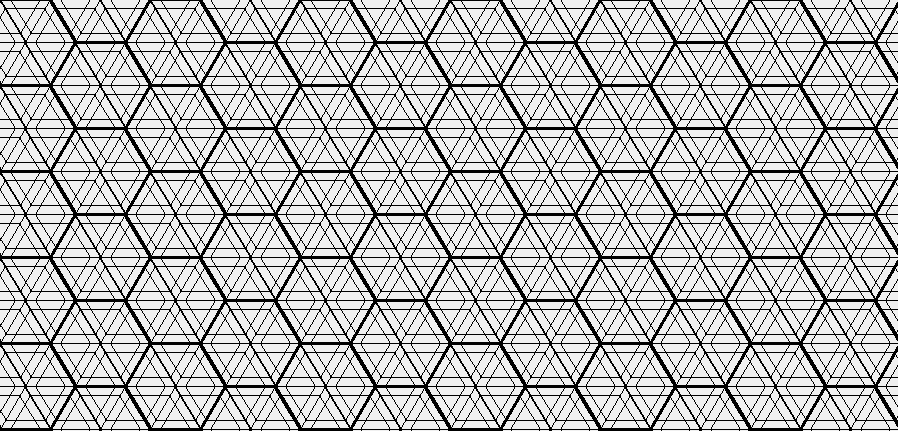


Рис. П2.1. Орнамент множества пересекающихся шестиугольников

Программа создает орнамент по параметрам: масштаб и цвет изображения.

2. Описание установки

Установка программы не требуется. Чтобы запустить программу, нужно иметь доступ к файлу программы и Visual Studio 2019 или выше.

3. Описание запуска

1. Открыть папку, в которой находится файл программы.

2. Открыть программу.

3. Появляется окно программы, в котором можно работать (рис. П2.2).

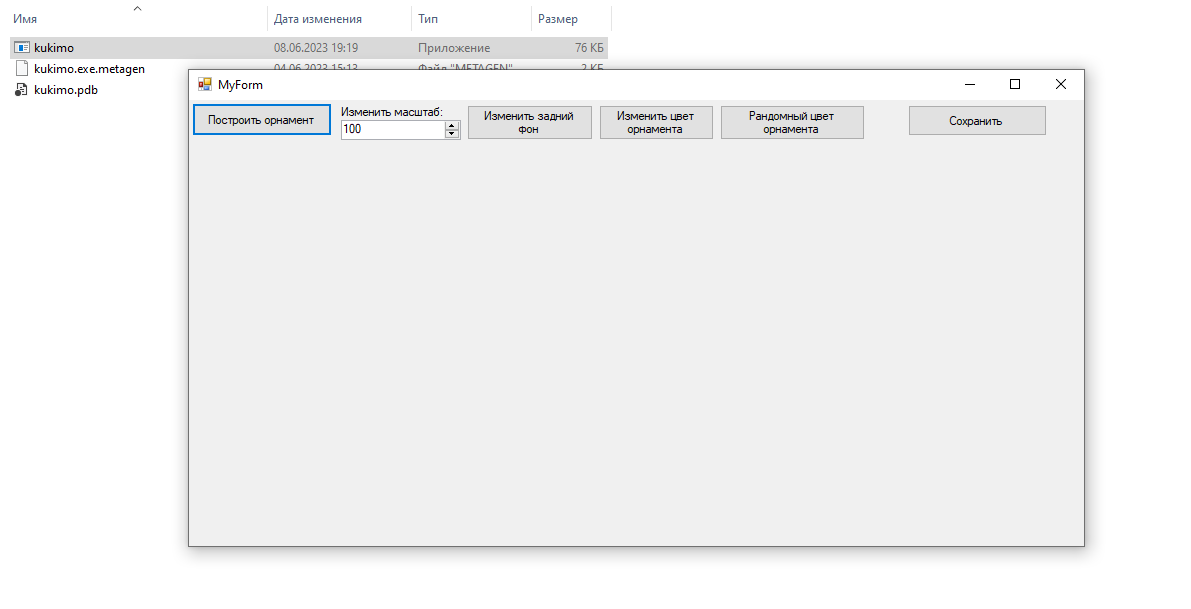


Рис. П2.2. Окно программы

4. Инструкция по работе

Параметры построения орнамента окружности по окружности задаются с помощью элементов интерфейса управления (рис. П2.3).

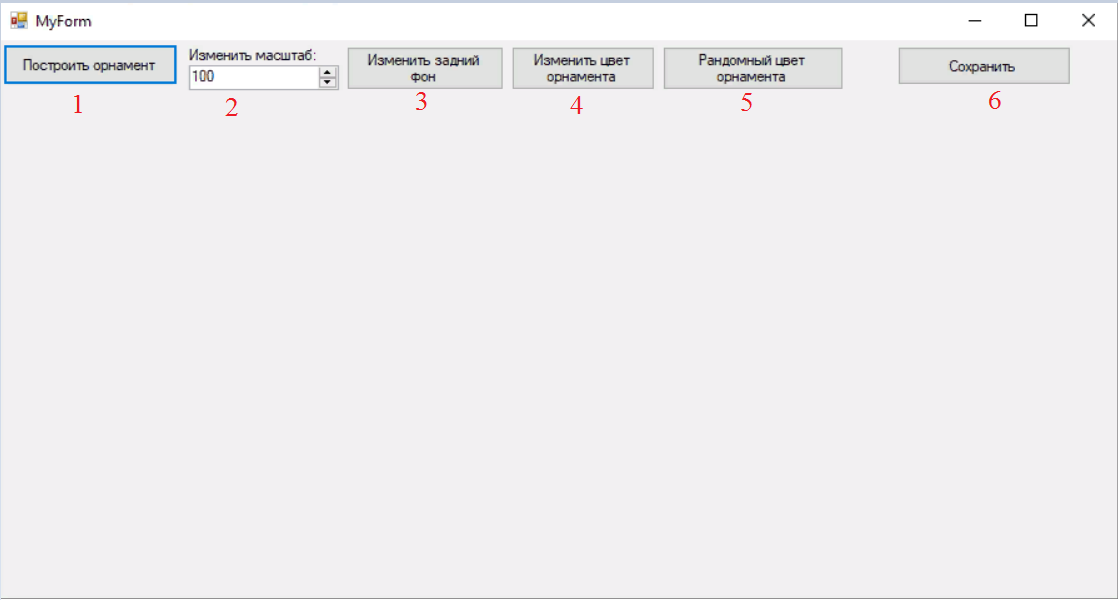


Рис. П2.3. Элементы управления

1. Построение изначального орнамента, цвет по стандарту чёрный, размер зависит от значения поля размера.
2. Выбор масштаба. Для выбора масштаба, можно ввести в соответствующее окно значение размера элемента орнамента(от 50 до 300) либо нажимать на треугольник и изменять значение на 1.
3. Выбор заднего фона. В этом случае всё поле для рисования заполнится одним цветом.
4. Выбор цвета орнамента. При нажатии на эту кнопку появляется диалоговое окно и при выборе необходимого цвета построится орнамет с соответствующим цветом.
5. Построение орнамента с случайным цветом. Для построения орнамента с случайным цветом нужно нажать на кнопку «Рандомный цвет».
6. Сохранение изображения. Для сохранения изображения нужно нажать на кнопу «Сохранить».Появится диалоговое окно, где пользователь может выбрать путь и формат сохранения файла.

После изменения масштаба и цвета программа построит орнамент (рис. П2.4) и сохранит его изображение (рис. П2.5).

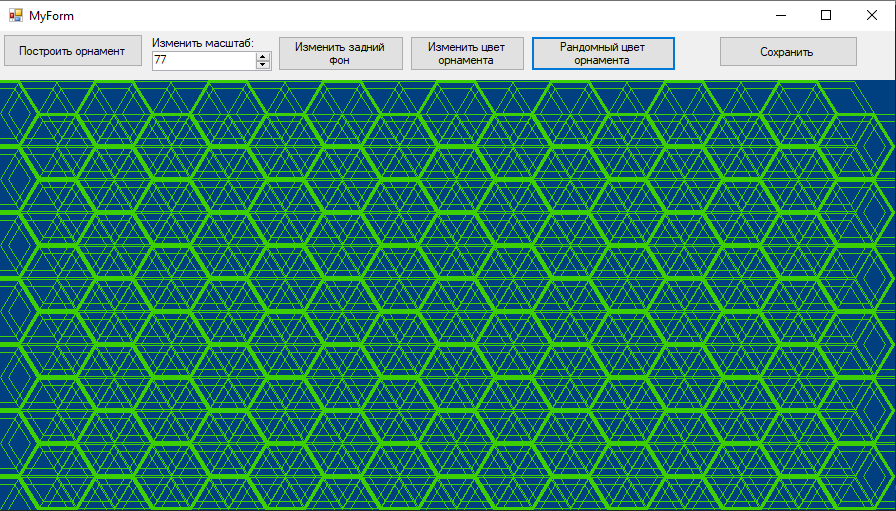


Рис. П2.4. Построенный орнамент

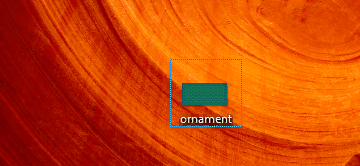


Рис. П2.5. Сохраненное изображение

5. Сообщения пользователю

При нажатии на кнопку изменить цвет орнамента появится диалоговое окно, представленное на рис. П2.6:

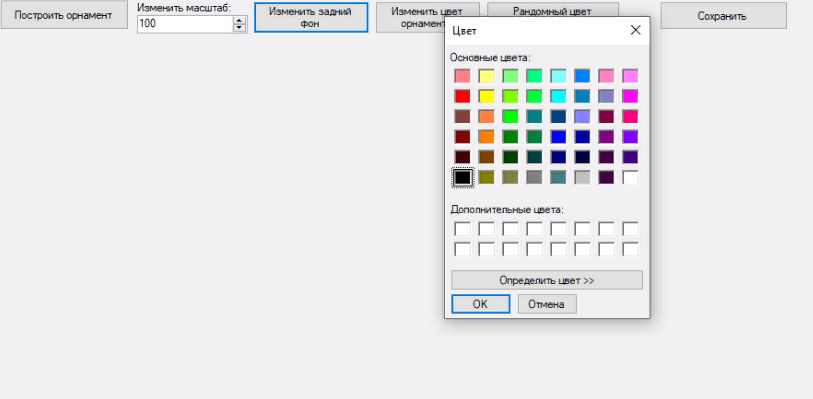


Рис. П2.6. Диалоговое окно выбора цвета

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

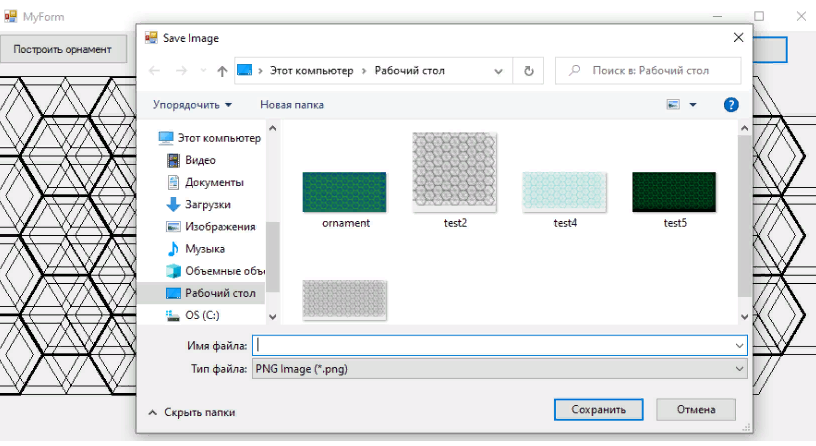


Рис. П2.7. Диалоговое окно сохранения изображения

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);

kukimo::MyForm form;

Application::Run(% form);

return 0;

}

Код MyForm.h:

#pragma once

#include <ctime>

#include <cstdlib>

namespace kukimo {

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::PictureBox^ pictureBox1;

protected:

private: System::Windows::Forms::Panel^ panel1;

private: System::Windows::Forms::Button^ button1;

private: System::Windows::Forms::Button^ SaveButton;

private: System::Windows::Forms::SaveFileDialog^ saveFileDialog1;

private: System::Windows::Forms::Button^ RandomColorButton;

private: System::Windows::Forms::Label^ label1;

private: System::Windows::Forms::NumericUpDown^ numericUpDown1;

private: System::Windows::Forms::Button^ colorButton;

private: System::Windows::Forms::ColorDialog^ colorDialog1;

private: System::Windows::Forms::Button^ button2;

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)

{

this->pictureBox1 = (gcnew System::Windows::Forms::PictureBox());

this->panel1 = (gcnew System::Windows::Forms::Panel());

this->button2 = (gcnew System::Windows::Forms::Button());

this->colorButton = (gcnew System::Windows::Forms::Button());

this->label1 = (gcnew System::Windows::Forms::Label());

this->numericUpDown1 = (gcnew System::Windows::Forms::NumericUpDown());

this->RandomColorButton = (gcnew System::Windows::Forms::Button());

this->SaveButton = (gcnew System::Windows::Forms::Button());

this->button1 = (gcnew System::Windows::Forms::Button());

this->saveFileDialog1 = (gcnew System::Windows::Forms::SaveFileDialog());

this->colorDialog1 = (gcnew System::Windows::Forms::ColorDialog());

(cli::safe\_cast<System::ComponentModel::ISupportInitialize^>(this->pictureBox1))->BeginInit();

this->panel1->SuspendLayout();

(cli::safe\_cast<System::ComponentModel::ISupportInitialize^>(this->numericUpDown1))->BeginInit();

this->SuspendLayout();

//

// pictureBox1

//

this->pictureBox1->Anchor = static\_cast<System::Windows::Forms::AnchorStyles>((((System::Windows::Forms::AnchorStyles::Top | System::Windows::Forms::AnchorStyles::Bottom)

| System::Windows::Forms::AnchorStyles::Left)

| System::Windows::Forms::AnchorStyles::Right));

this->pictureBox1->Location = System::Drawing::Point(0, 49);

this->pictureBox1->Name = L"pictureBox1";

this->pictureBox1->Size = System::Drawing::Size(895, 397);

this->pictureBox1->TabIndex = 0;

this->pictureBox1->TabStop = false;

//

// panel1

//

this->panel1->Anchor = System::Windows::Forms::AnchorStyles::Top;

this->panel1->Controls->Add(this->button2);

this->panel1->Controls->Add(this->colorButton);

this->panel1->Controls->Add(this->label1);

this->panel1->Controls->Add(this->numericUpDown1);

this->panel1->Controls->Add(this->RandomColorButton);

this->panel1->Controls->Add(this->SaveButton);

this->panel1->Controls->Add(this->button1);

this->panel1->Location = System::Drawing::Point(0, 0);

this->panel1->Name = L"panel1";

this->panel1->Size = System::Drawing::Size(895, 43);

this->panel1->TabIndex = 1;

//

// button2

//

this->button2->Location = System::Drawing::Point(278, 5);

this->button2->Name = L"button2";

this->button2->Size = System::Drawing::Size(126, 35);

this->button2->TabIndex = 6;

this->button2->Text = L"Изменить задний фон";

this->button2->UseVisualStyleBackColor = true;

this->button2->Click += gcnew System::EventHandler(this, &MyForm::button2\_Click);

//

// colorButton

//

this->colorButton->Location = System::Drawing::Point(410, 5);

this->colorButton->Name = L"colorButton";

this->colorButton->Size = System::Drawing::Size(115, 35);

this->colorButton->TabIndex = 5;

this->colorButton->Text = L"Изменить цвет орнамента";

this->colorButton->UseVisualStyleBackColor = true;

this->colorButton->Click += gcnew System::EventHandler(this, &MyForm::colorButton\_Click);

//

// label1

//

this->label1->AutoSize = true;

this->label1->Location = System::Drawing::Point(149, 5);

this->label1->Name = L"label1";

this->label1->Size = System::Drawing::Size(109, 13);

this->label1->TabIndex = 4;

this->label1->Text = L"Изменить масштаб:";

//

// numericUpDown1

//

this->numericUpDown1->Location = System::Drawing::Point(152, 20);

this->numericUpDown1->Maximum = System::Decimal(gcnew cli::array< System::Int32 >(4) { 300, 0, 0, 0 });

this->numericUpDown1->Minimum = System::Decimal(gcnew cli::array< System::Int32 >(4) { 50, 0, 0, 0 });

this->numericUpDown1->Name = L"numericUpDown1";

this->numericUpDown1->Size = System::Drawing::Size(120, 20);

this->numericUpDown1->TabIndex = 3;

this->numericUpDown1->Value = System::Decimal(gcnew cli::array< System::Int32 >(4) { 100, 0, 0, 0 });

this->numericUpDown1->ValueChanged += gcnew System::EventHandler(this, &MyForm::numericUpDown1\_ValueChanged);

//

// RandomColorButton

//

this->RandomColorButton->Location = System::Drawing::Point(531, 5);

this->RandomColorButton->Name = L"RandomColorButton";

this->RandomColorButton->Size = System::Drawing::Size(145, 35);

this->RandomColorButton->TabIndex = 2;

this->RandomColorButton->Text = L"Рандомный цвет орнамента";

this->RandomColorButton->UseVisualStyleBackColor = true;

this->RandomColorButton->Click += gcnew System::EventHandler(this, &MyForm::RandomColorButton\_Click);

//

// SaveButton

//

this->SaveButton->Location = System::Drawing::Point(719, 5);

this->SaveButton->Name = L"SaveButton";

this->SaveButton->Size = System::Drawing::Size(139, 31);

this->SaveButton->TabIndex = 1;

this->SaveButton->Text = L"Сохранить";

this->SaveButton->UseVisualStyleBackColor = true;

this->SaveButton->Click += gcnew System::EventHandler(this, &MyForm::SaveButton\_Click);

//

// button1

//

this->button1->Location = System::Drawing::Point(3, 3);

this->button1->Name = L"button1";

this->button1->Size = System::Drawing::Size(140, 33);

this->button1->TabIndex = 0;

this->button1->Text = L"Построить орнамент";

this->button1->UseVisualStyleBackColor = true;

this->button1->Click += gcnew System::EventHandler(this, &MyForm::button1\_Click\_1);

//

// MyForm

//

this->AutoScaleDimensions = System::Drawing::SizeF(6, 13);

this->AutoScaleMode = System::Windows::Forms::AutoScaleMode::Font;

this->ClientSize = System::Drawing::Size(895, 446);

this->Controls->Add(this->panel1);

this->Controls->Add(this->pictureBox1);

this->Name = L"MyForm";

this->Text = L"MyForm";

(cli::safe\_cast<System::ComponentModel::ISupportInitialize^>(this->pictureBox1))->EndInit();

this->panel1->ResumeLayout(false);

this->panel1->PerformLayout();

(cli::safe\_cast<System::ComponentModel::ISupportInitialize^>(this->numericUpDown1))->EndInit();

this->ResumeLayout(false);

}

#pragma endregion

private:

int hexagonSize;

int hexagonColor;

Color GetRandimColor() {

srand(time(NULL));

int r = rand() % 256;

int g = rand() % 256;

int b = rand() % 256;

return Color::FromArgb(r, g, b);

}

private: System::Void numericUpDown1\_ValueChanged(System::Object^ sender, System::EventArgs^ e) {

pictureBox1->Refresh();

}

private: System::Void button1\_Click\_1(System::Object^ sender, System::EventArgs^ e) {

// Очищаем box

pictureBox1->Refresh();

OrnamentDraw((int)numericUpDown1->Value, Color::Black);

}

private: System::Void SaveButton\_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;

}

private: System::Void RandomColorButton\_Click(System::Object^ sender, System::EventArgs^ e) {

// Очищаем box

pictureBox1->Refresh();

OrnamentDraw((int)numericUpDown1->Value, GetRandimColor());

}

void OrnamentDraw(int size, Color color) {

Graphics^ graphics = pictureBox1->CreateGraphics();

Pen^ pen = gcnew Pen(color);

//размеры picture box

int width = pictureBox1->Width;

int height = pictureBox1->Height;

// Размер каждого шестиугольника

int totalHexagons = width / (size / 2);

// Общая ширина всех шестиугольников и промежутков между ними

int startX = 0; // Начальная позиция по X

int startY = 0;

int numHexagonsY = height / (size / 2);

int rowY = startY;

int rowX = startX;

for (int ii = 0; ii < numHexagonsY; ii++) {

// кооринаты y для текущего ряда

rowY += size / 2 - size/14;

// условие чередования местоположения

if (ii % 2 != 0) {

rowX += (size / 2 + size / 4);

}

BigHexagonDraw(totalHexagons, size, rowX, rowY, graphics, pen,color);

//при чередовании возвращениие в предыдущее

if (ii % 2 != 0) {

rowX = startX;

}

}

}

void BigHexagonDraw(int totalHexagons, int size, int rowX, int rowY, Graphics^ graphics, Pen^ pen, Color color) {

int i = 0;

while (i < totalHexagons || size \* i + size <= pictureBox1->Width) {

int x = rowX + (size / 2) \* i; // Позиция X текущего шестиугольника

// Рассчитываем координаты точек для текущего big шестиугольника

array<Point>^ bighexagonPoints = gcnew array<Point>(6);

for (int j = 0; j < 6; j++) {

double angle = 2 \* Math::PI / 6 \* j;

int pointX = x + (int)(size / 2 \* Math::Cos(angle));

int pointY = rowY + (int)(size / 2 \* Math::Sin(angle));

bighexagonPoints[j] = Point(pointX, pointY);

}

// Рисуем текущий big шестиугольник

if (i % 3 == 0) {

Pen^ thickPen = gcnew Pen(color, 3);

graphics->DrawPolygon(thickPen, bighexagonPoints);

}

else {

graphics->DrawPolygon(pen, bighexagonPoints);

}

SmallHexagonDraw(size, x, rowY, graphics, pen);

i++;

}

}

void SmallHexagonDraw(int size,int x,int rowY,Graphics^ graphics,Pen^ pen) {

// Рассчитываем размеры и координаты для внутреннего малого шестиугольника

int smallSize = (int)(size \* 0.8);

int smallX = x;

int smallY = rowY;

// Рассчитываем координаты точек для текущего малого шестиугольника

array<Point>^ smallHexagonPoints = gcnew array<Point>(6);

for (int jj = 0; jj < 6; jj++) {

double angle = 2 \* Math::PI / 6 \* jj;

int pointX = smallX + (int)(smallSize / 2 \* Math::Cos(angle));

int pointY = smallY + (int)(smallSize / 2 \* Math::Sin(angle));

smallHexagonPoints[jj] = Point(pointX, pointY);

} // Рисуем текущий малый шестиугольник

graphics->DrawPolygon(pen, smallHexagonPoints);

}

private: System::Void colorButton\_Click(System::Object^ sender, System::EventArgs^ e) {

ColorDialog^ colorDialog1 = gcnew ColorDialog();

if (colorDialog1->ShowDialog() == System::Windows::Forms::DialogResult::OK) {

Color selectedcolor = colorDialog1->Color;

OrnamentDraw((int)numericUpDown1->Value, selectedcolor);

}

}

private: System::Void button2\_Click(System::Object^ sender, System::EventArgs^ e) {

ColorDialog^ colorDialog2 = gcnew ColorDialog();

if (colorDialog2->ShowDialog() == System::Windows::Forms::DialogResult::OK) {

Color selectedcolor = colorDialog2->Color;

pictureBox1->BackColor = selectedcolor;

}

}

};

}