|  |  |
| --- | --- |
| Институт (факультет) | Институт информационных технологий |
| Кафедра | Математическое и программное обеспечение ЭВМ |

**КУРСОВАЯ РАБОТА**

|  |
| --- |
| по дисциплине Структурное программирование |

|  |  |
| --- | --- |
| на тему | Программирование на языке высокого уровня |

|  |
| --- |
| Выполнил студент группы |
| 1ПИб-02-2оп-23 |
| *группа* |
| направления подготовки (специальности) |
| 09.03.04 Программная инженерия |
| *шифр, наименование* |
| Кринкин Олег Алексеевич |
| *фамилия, имя, отчество* |

|  |
| --- |
| Руководитель |
| Пышницкий Константин Михайлович |
| *фамилия, имя, отчество* |
| Старший преподаватель |
| *должность* |

|  |
| --- |
| Дата представления работы |
| «\_\_\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2024 г. |
|  |
| Заключение о допуске к защите |
|  |
|  |
|  |
|  |
| Оценка \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_, \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
| количество баллов |
| Подпись преподавателя\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |

Череповец, 2024

*год*

Аннотация

Курсовой проект на тему «Программирование на языке высокого уровня: построение орнамента "Окружности по окружности"».

Выполнил студент группы 1ПИб-02-2оп-23 Института информационных технологий Кринкин Олег Алексеевич.

Цель курсовой работы: разработка программы для построения орнамента «Окружности по окружности» с различными параметрами.

Курсовая работа содержит в себе: введение, описание предметной области, описание использованных классов, заключение, руководство пользователя, а также код программы.

Оглавление

[1. Введение 4](#_Toc168925274)

[2. Описание предметной области 5](#_Toc168925275)

[3. Описание классов Graphics, Pen, Brush. 6](#_Toc168925276)

[3.1. Класс Graphics 6](#_Toc168925277)

[3.2. Класс Pen 8](#_Toc168925278)

[3.3. Класс Brush 9](#_Toc168925279)

[4.1. Постановка задачи 10](#_Toc168925280)

[4.2. Логическое проектирование 10](#_Toc168925281)

[4.3. Физическое проектирование 12](#_Toc168925282)

[5. Тестирование 15](#_Toc168925283)

[6. Результаты работы 16](#_Toc168925284)

[Список литературы 17](#_Toc168925285)

[Приложение 1. Техническое задание 18](#_Toc168925286)

[Приложение 2. Руководство пользователя 25](#_Toc168925287)

[Приложение 3. Программный код 31](#_Toc168925288)

# 1. Введение

В настоящее время программное обеспечение становится все более комплексным и всё более сложным в написании. Для того, чтобы упростить написание таких программ, а также сократить время на разработку были созданы языки программирования высокого уровня.

Языки программирования высокого уровня отличаются абстракцией – более понятной для человека формой записи программного кода. Также языки высокого уровня позволяют создавать программное обеспечение для различных аппаратных платформ благодаря трансляции языковых конструкций высокого языка в машинный код конкретной платформы. [1]

Одним из представителей языков высокого уровня является язык C++.

Язык программирования C++ является мощным инструментом, широко применяемым для разработки различных приложений, включая графические программы. Благодаря своей эффективности, гибкости и высокой производительности, C++ стал предпочтительным выбором для создания программ, работающих с графикой, пользовательскими интерфейсами и другими сложными задачами [2].

В целях изучения принципов разработки программного обеспечения на языках высокого уровня в рамках курсовой работы будет написана программа с графическим интерфейсом на языке C++, задача которой – построение орнамента «Окружности по окружности».

# 2. Описание предметной области

Орнамент окружности по окружности представляет собой узор, состоящий из множества окружностей, центры которых находятся на общей окружности. Пример одного из таких орнаментов представлен на рис. 1.

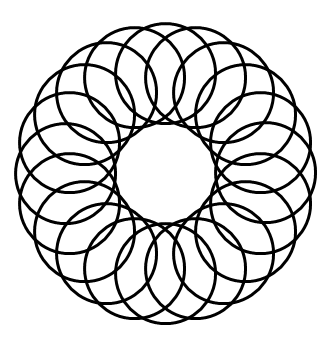


Рис. 1. Пример орнамента

Окружности могут иметь различные радиусы, а также их может быть различное количество, но при этом окружности имеют одинаковый размер. Центры окружностей находятся на одинаковом расстоянии. Рассматриваемый орнамент является симметричной фигурой, которая может использоваться в различных сферах. В качестве примера проявления такого орнамента в архитектуре на рис. 2 приведён витраж готической архитектуры.



Рис. 2: Готический витраж

# 3. Описание классов Graphics, Pen, Brush.

Для написания программы с функцией отрисовки изображения на экране устройства пользователя использованы классы Graphics, Pen и Brush. В следующих разделах описаны используемые классы.

## 3.1. Класс Graphics

Содержится в пространстве имён System::Drawing и предоставляет методы для рисования объектов на устройстве отображения. [3]

Некоторые предоставляемых классом методов представлены в табл. 1:

Таблица 1

Методы класса Graphics

|  |  |
| --- | --- |
| Название | Описание |
| AddMetafileComment(Byte[]) | Добавляет комментарий к текущему метафайлу. |
| BeginContainer() | Сохраняет графический контейнер с текущим состоянием этого графического контейнера, открывает и использует новый графический контейнер. |
| BeginContainer(Rectangle, Rectangle, GraphicsUnit) | Сохраняет графический контейнер с текущим состоянием этого графического объекта, открывает и использует новый графический контейнер с указанным преобразованием масштаба. |
| Clear(Color) | Очищает всю поверхность рисования и заполняет ее заданным цветом фона. |
| CopyFromScreen(Int32, Int32, Int32, Int32, Size) | Выполняет передачу цветовых данных, соответствующих прямоугольнику пикселей, с экрана на поверхность рисования графического объекта. |
| CopyFromScreen(Point, Point, Size) | Выполняет передачу цветовых данных, соответствующих прямоугольнику пикселей, с экрана на поверхность рисования графического элемента. |
| Dispose() | Освобождает все ресурсы, используемые объектом Graphics. |
| DrawArc(Pen, Int32, Int32, Int32, Int32, Int32, Int32) | Рисует дугу, представляющую часть эллипса, заданную парой координат, шириной и высотой. |
| DrawBezier(Pen, Point, Point, Point, Point) | Рисует сплайн Безье, определяемый четырьмя структурами Point. |
| DrawCachedBitmap(CachedBitmap, Int32, Int32) | Рисует заданное кэшированное растровое изображение. |
| DrawCurve(Pen, Point[]) | Рисует кардинальный сплайн через указанный массив точечных структур. |
| DrawEllipse(Pen, Int32, Int32, Int32, Int32) | Рисует эллипс, определяемый ограничивающим прямоугольником, заданным координатами верхнего левого угла прямоугольника, высотой и шириной. |
| DrawIcon(Icon, Int32, Int32) | Рисует изображение, представленное указанным значком, в указанных координатах. |
| DrawImage(Image, Int32, Int32) | Рисует указанное изображение, используя его исходный физический размер, в местоположении, заданном парой координат. |
| DrawLine(Pen, Int32, Int32, Int32, Int32) | Рисует линию, соединяющую две точки, заданные парами координат. |
| DrawPath(Pen, GraphicsPath) | Рисует объект GraphicsPath. |
| DrawPie(Pen, Int32, Int32, Int32, Int32, Int32, Int32) | Рисует фигуру круговой диаграммы, определяемую эллипсом, заданным парой координат, шириной, высотой и двумя радиальными линиями. |
| DrawPolygon(Pen, Point[]) | Рисует многоугольник, определяемый массивом точечных структур. |
| DrawRectangle(Pen, Int32, Int32, Int32, Int32) | Рисует прямоугольник, заданный парой координат, шириной и высотой. |
| DrawString(ReadOnlySpan<Char>, Font, Brush, PointF) | Рисует указанную текстовую строку в указанном месте с указанными объектами Brush и Font. |
| EndContainer(GraphicsContainer) | Закрывает текущий графический контейнер и восстанавливает состояние этого графического объекта до состояния, сохраненного вызовом метода BeginContainer(). |
| Finalize() | Позволяет объекту попытаться освободить ресурсы и выполнить другие операции очистки до того, как он будет освобожден сборщиком мусора. |
| Flush() | Принудительно выполняет все ожидающие графические операции и немедленно возвращается, не дожидаясь завершения операций. |
| MultiplyTransform(Matrix) | Умножает мировое преобразование этой Графики и задает Матрицу. |
| ResetTransform() | Сбрасывает матрицу преобразования мира этого объекта Graphics в единичную матрицу. |
| Restore(GraphicsState) | Восстанавливает состояние этого объекта Graphics до состояния, представленного объектом GraphicsState. |
| RotateTransform(Single) | Применяет заданный поворот к матрице преобразования данного графического объекта. |
| Save() | Сохраняет текущее состояние этого объекта Graphics и идентифицирует сохраненное состояние с помощью GraphicsState. |
| ScaleTransform(Single, Single) | Применяет указанную операцию масштабирования к матрице преобразования данного объекта Graphics, добавляя ее к матрице преобразования объекта. |
| TranslateTransform(Single, Single) | Изменяет начало координат путем добавления указанного преобразования в матрицу преобразования данного объекта Graphics. |

## 3.2. Класс Pen

Содержится в пространстве имён System::Drawing и определяет объект, используемый для рисования прямых линий и кривых. Этот класс не наследуется. [4]

В таблице ниже представлены методы класса Pen (табл. 2):

Таблица 2

Методы класса Pen

|  |  |
| --- | --- |
| Название | Описание |
| Pen(Brush) | Инициализирует новый экземпляр класса Pen с указанным свойством Brush. |
| Clone() | Создает точную копию объекта Pen. |
| Dispose() | Освобождает все ресурсы, используемые объектом Pen. |
| Finalize() | Позволяет объекту попытаться освободить ресурсы и выполнить другие операции очистки до того, как он будет освобожден сборщиком мусора. |
| MultiplyTransform(Matrix) | Умножает матрицу трансформации для этого пера на указанную матрицу. |
| MultiplyTransform(Matrix, MatrixOrder) | Умножает матрицу преобразования для этого пера на указанную матрицу в указанном порядке. |
| ResetTransform() | Сбрасывает матрицу геометрического преобразования для этого пера в идентичность. |
| RotateTransform(Single) | Поворот локального геометрического преобразования на заданный угол. Этот метод предваряет поворот к преобразованию. |
| RotateTransform(Single, MatrixOrder) | Поворот локального геометрического преобразования на заданный угол в указанном порядке. |
| ScaleTransform(Single, Single) | Масштабирует локальное геометрическое преобразование по заданным коэффициентам. Этот метод добавляет матрицу масштабирования к преобразованию. |
| ScaleTransform(Single, Single, MatrixOrder) | Масштабирует локальное геометрическое преобразование по заданным коэффициентам в указанном порядке. |
| SetLineCap(LineCap, LineCap, DashCap) | Задает значения, определяющие стиль прописных элементов, используемых для завершения линий, нарисованных этим пером. |
| TranslateTransform(Single, Single) | Преобразует локальное геометрическое преобразование в заданные размеры. Этот метод предшествует преобразованию в начале преобразования. |
| TranslateTransform(Single, Single, MatrixOrder) | Преобразует локальное геометрическое преобразование по заданным размерам в указанном порядке. |

## 3.3. Класс Brush

Содержится в пространстве имён System::Drawing и определяет объекты для заливки внутри графических фигур, таких как прямоугольники, эллипсы, круги, многоугольники и пути. [5]

В таблице ниже представлены методы класса Brush (табл. 3):

Таблица 3

Методы класса Brush

|  |  |
| --- | --- |
| Название | Описание |
| Brush() | Инициализирует новый экземпляр класса Brush. |
| Clone() | При переопределении в производном классе создает точную копию этого объекта Brush. |
| Dispose() | Освобождает все ресурсы, используемые этим объектом Brush. |
| Dispose(Boolean) | Освобождает неуправляемые ресурсы, используемые кистью, и при необходимости освобождает управляемые ресурсы. |
| Finalize() | Позволяет объекту попытаться освободить ресурсы и выполнить другие операции очистки до того, как он будет освобожден сборщиком мусора. |
| SetNativeBrush(IntPtr) | В производном классе задает ссылку на объект кисти GDI+. |

4. Описание созданного приложения

В этом разделе описаны основные этапы проектирования и разработки конечной программы для построения орнамента «Окружности по окружности».

## 4.1. Постановка задачи

Для того, чтобы программа успешно выполняла поставленную перед ней задачу необходимо чтобы она:

* получала от пользователя радиусы внешних и внутренней окружностей;
* предоставляла возможности для изменения цвета линий и фона орнамента;
* создавала графическое изображение необходимого орнамента, построенного по введённым данным;
* сохраняла отображаемый на экране орнамент в файл по нажатию соответствующей кнопки;
* предоставляла возможность вращения орнамента;
* повторно строила орнамент при изменении параметров;
* предоставляла возможность установки фонового изображения.

## 4.2. Логическое проектирование

Для построения орнамента используется следующий алгоритм работы:

Из элементов графического пользовательского интерфейса программа получает данные о количестве окружностей, о радиусе окружностей, о толщине и цвете линий. После получения необходимых данных по отдельности для каждой из окружностей создаётся объект класса Pen, устанавливается для него цвет и толщина линии и вызывается функция, которая располагает окружность на экране, принимая в качестве параметра номер окружности. Все остальные значения для работы функция получает из элементов графического интерфейса. Расчёт координат расположения центров окружностей по осям и в окне программы происходит по следующей формуле:

где, - длина области рисования, - ширина области рисования, – радиус внешних окружностей, – радиус внутренней окружности, – угол размещения окружности. Угол размещения окружности рассчитывается по следующей формуле:

где, – количество окружностей, – номер окружности, – угол поворота орнамента.

Алгоритм построения орнамента представлен в следующей блок-схеме (рис. 3):

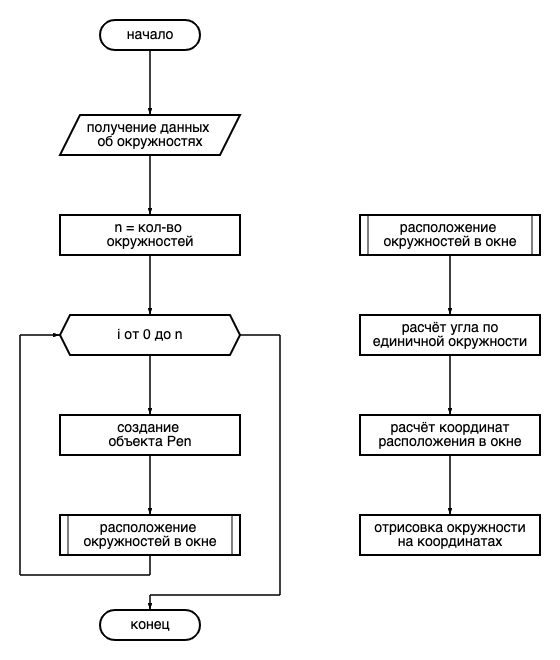


Рис. 3. Блок-схема построения орнамента

## 4.3. Физическое проектирование

В этом подразделе описаны использованные переменные и функции, содержащиеся в программе.

Все переменные, используемые в основной части программы представлены в табл. 4.

Таблица 4

Переменные

|  |  |  |
| --- | --- | --- |
| Содержание | Обозначение | Тип данных |
| График отрисован | drawed | bool |
| Включено вращение | rotation | bool |
| Включён случайный цвет линий | randcolor | bool |
| Угол поворота орнамента | rotation\_angle | float |
| Таймер отрисовки | rotTimer | Forms::Timer^ |
| Массив цветов окружностей | colorArr | array <Drawing::Color>^ |
| Объект, рисующий окружности | drawingPen | Pen^ |
| Кол-во окружностей | count | int |
| Радиус внутренней окружности | loc\_radius | int |
| Радиус внешних окружностей | radius | int |
| Значение радианы | radians | double |
| Значение угла для окружности | angle | double |
| Координата X окружности | x | int |
| Координата Y окружности | y | int |

В следующей таблице расписаны спецификации для каждой из функций, содержащейся в основной части программы (табл. 5):

Таблица 5

Спецификация функций

|  |  |  |  |
| --- | --- | --- | --- |
| Имя модуля | Заголовок процедуры или функции | Формальные параметры | Выполняемое действие |
| MyForm.h | radiusIn\_ValueChanged | System::Object^ sender, System::EventArgs^ e | Обновляет содержимое поля рисования при изменении радиуса внутренней окружности |
| MyForm.h | radiusOut\_ValueChanged | System::Object^ sender, System::EventArgs^ e | Обновляет содержимое поля рисования при изменении радиуса внешней окружности |
| MyForm.h | lineThickness\_ValueChanged | System::Object^ sender, System::EventArgs^ e | Обновляет содержимое поля рисования при изменении толщины линии |
| MyForm.h | circlesCount\_ValueChanged | System::Object^ sender, System::EventArgs^ e | Обновляет содержимое поля рисования при изменении количества окружностей |
| MyForm.h | rotationSpeed\_ValueChanged | System::Object^ sender, System::EventArgs^ e | Обновляет содержимое поля рисования при изменении скорости вращения |
| MyForm.h | buttonDraw\_Click | System::Object^ sender, System::EventArgs^ e | Запускает или останавливает процесс отрисовки орнамента по нажатию на кнопку |
| MyForm.h | buttonRotate\_Click | System::Object^ sender, System::EventArgs^ e | Запускает или останавливает вращение орнамента по нажатию на кнопку |
| MyForm.h | buttonColor\_Click | System::Object^ sender, System::EventArgs^ e | Вызывает диалог выбора цвета для окружностей и обновляет поле для рисования |
| MyForm.h | GetRandColor | int index | Возвращает случайный цвет |
| MyForm.h | ButtonRainbow\_Click | System::Object^ sender, System::EventArgs^ e | Переключает цвет окружностей на случайный или выбранный в диалоге |
| MyForm.h | Draw | System::Object^ sender, System::Windows::Forms::PaintEventArgs^ event | Основная функция отрисовки, создающая объекты рисования для окружностей |
| MyForm.h | DrawCircle | Pen^ drawer, System::Windows::Forms::PaintEventArgs^ event, int index | Отрисовывает окружность на полотне |
| MyForm.h | Rotate | System::Object^ sender, System::EventArgs^ e | Увеличивает угол вращения орнамента с заданной скоростью |
| MyForm.h | buttonSave\_Click | System::Object^ sender, System::EventArgs^ e | Сохраняет отрисованное на экране в Bitmap и вызывает диалог сохранения файла |
| MyForm.h | MyForm\_Resize | System::Object^ sender, System::EventArgs^ e | Обновляет содержимое поля рисования при изменении размера окна |
| MyForm.h | buttonBackSettings\_Click | System::Object^ sender, System::EventArgs^ e | Вызывает форму с параметрами для фона |
| BackSettings.h | buttonBackColor\_Click | System::Object^ sender, System::EventArgs^ e | Вызывает диалог выбора цвета и устанавливает выбранный цвет как цвет фона |
| BackSettings.h | buttonBackRandom\_Click | System::Object^ sender, System::EventArgs^ e | Устанавливает случайный цвет фона по нажатию на кнопку |
| BackSettings.h | buttonBackImage\_Click | System::Object^ sender, System::EventArgs^ e | Вызывает диалог выбора файла и устанавливает выбранный файл как фон |
| BackSettings.h | buttonReset\_Click | System::Object^ sender, System::EventArgs^ e | Устанавливает цвет фона по умолчанию и удаляет изображение |
| BackSettings.h | comboBox\_SelectedIndexChanged | System::Object^ sender, System::EventArgs^ e | Устанавливает режим расположения фонового изображения |

# 5. Тестирование

Во время разработки программы несколько раз были протестированы некоторые её функции. Результаты тестирований представлены в табл. 6.

Таблица 6

Результаты тестирования

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Дата тестирования | Тестируемый модуль | Кто проводил тестирование | Описание теста | Результат тестирования |
| 28.04.24 | MyForm.h | Кринкин О. А. | Рисование окружностей при случайных данных | Успех |
| 30.04.24 | MyForm.h | Кринкин О. А. | Вращение орнамента с различной скоростью | Успех |
| 05.05.24 | MyForm.h | Кринкин О. А. | Задание случайного цвета | Неудача |
| 20.05.24 | MyForm.h | Кринкин О. А. | Повторное задание случайного цвета | Успех |
| 26.05.24 | MyForm.h | Кринкин О. А. | Сохранение орнамента в файл | Неудача (NullPointerException) |
| 27.05.24 | MyForm.h | Кринкин О. А. | Повторное сохранение орнамента | Успех |
| 06.06.24 | BackSettings.h | Кринкин О. А. | Применение цвета для фона | Неудача (NullPointerException) |
| 07.06.24 | BackSettings.h | Кринкин О. А. | Повторное применение цвета для фона | Успех |
| 07.06.24 | MyForm.h | Кринкин О. А. | Полное тестирование программы | Успех |

# 6. Результаты работы

В результате выполнения курсовой работы была разработана программа на языке C++ c использованием библиотек Windows Forms из состава .NET Framework.

Разработанная программа позволяет построить графическое представление орнамента «Окружности по окружности» и сохранить полученный результат в виде растрового изображения. Программа позволяет изменять параметры орнамента: радиус внешних и внутренней окружностей, количество окружностей, цвет линий, цвет фона, фоновое изображение, а также задавать орнаменту вращение. При этом интерфейс программы достаточно прост, чтобы быстро выполнить поставленную задачу (рис. 4).

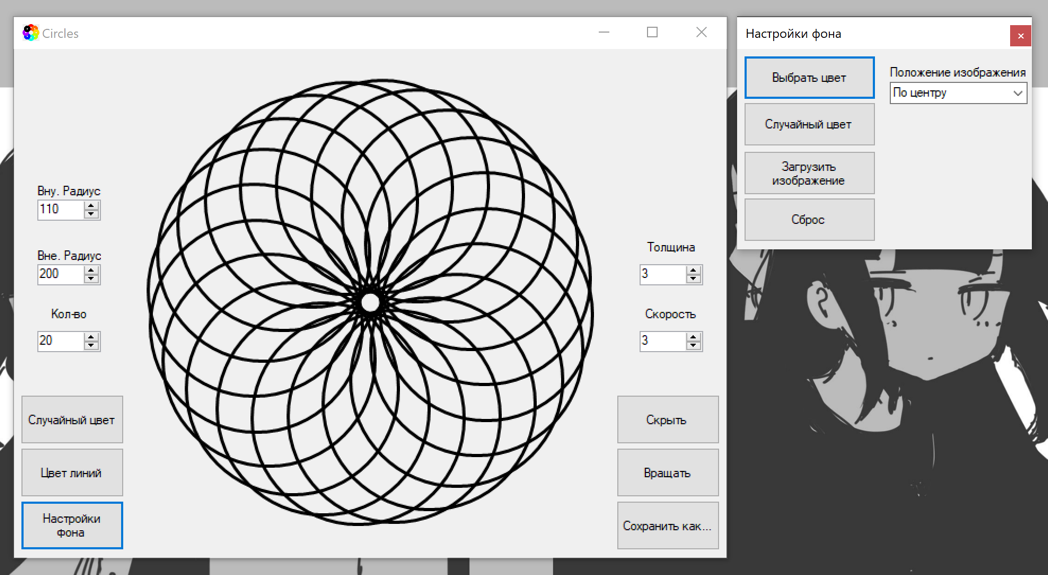


Рис. 4: Интерфейс программы

# Список литературы

1. Высокоуровневый язык программирования [электронный ресурс]. Режим доступа: https://blog.skillfactory.ru/glossary/vysokourovnevyy-yazyk-programmirovaniya, свободный – (5 июня 2024)
2. Кому и для чего нужен C++ [электронный ресурс]. Режим доступа: https://blog.skillfactory.ru/cplus-komu-i-dlya-chego-nuzhen, свободный – (5 июня 2024)
3. Класс Graphics [электронный ресурс]. Режим доступа: https://learn.microsoft.com/ru-ru/dotnet/api/system.drawing.graphics, свободный – (6 июня 2024)
4. Класс Pen [электронный ресурс]. Режим доступа: https://learn.microsoft.com/ru-ru/dotnet/api/system.drawing.pen, свободный – (6 июня 2024)
5. Класс Brush [электронный ресурс]. Режим доступа: https://learn.microsoft.com/ru-ru/dotnet/api/system.drawing.brush, свободный – (6 июня 2024)

# Приложение 1. Техническое задание

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

|  |
| --- |
| информационных технологий |
| наименование института (факультета) |
| математическое и программное обеспечение ЭВМ |
| наименование кафедры |
| Программная инженерия |
| наименование дисциплины в соответствии с учебным планом |

УТВЕРЖДАЮ

Зав. кафедрой \_\_\_\_\_\_\_\_\_\_\_,

д.т.н., профессор \_\_\_\_\_\_\_\_\_\_ Ершов Е.В.

«\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_ 2024 г.

ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ ВЫСОКОГО УРОВНЯ

Техническое задание на курсовую работу

Листов 5

|  |  |
| --- | --- |
|  | Пышницкий К. М. |
|  | Ф.И.О. Преподавателя |

Руководитель

|  |
| --- |
| 1ПИб-02-2оп-23 |
| |  | | --- | | Кринкин О. А. | | Фамилия, имя, отчество |   группа |

Исполнитель студент

Введение

Продукт предназначен для построения орнамента «Окружности по окружности», используя введённые пользователем данные, и последующего сохранения результата в качестве изображения.

Проект курсовой работы предназначен для формирования навыков по профильному предмету «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.

Дата утверждения: 21 февраля 2024 года.

Наименование темы разработки: программирование на языке высокого уровня.

2. Назначение разработки

Программа предоставляет возможности для построения разнообразных орнаментов типа «Окружности по окружности». Пользователь имеет возможность изменять различные параметры орнамента, устанавливать цвет как орнамента, так и фона, устанавливать фоновое изображение, вращать орнамент и сохранять отображаемый результат.

3. Требования к программе

3.1. Требования к функциональным характеристикам

Функциональность программы должна отвечать следующим характеристикам:

* Генерация орнамента: Программа должна позволять пользователю создавать орнаменты на основе заданных параметров, таких как радиусы, количество окружностей и цвет.
* Визуализация орнамента: Созданный орнамент должен быть наглядно для пользователя отображён на экране.
* Изменение параметров орнамента: Пользователь должен иметь возможность изменять параметры орнамента, такие как радиусы, количество окружностей и цвет, и получать визуальное представление внесённых изменений.
* Сохранение орнамента: Пользователь должен иметь возможность сохранить созданный орнамент в файл для последующего использования в собственных целях.
* Исключение ошибок: Программа должна ограничивать ввод пользователя для предотвращения предоставления неверных данных.
* Понятный интерфейс: Интерфейс программы должен быть интуитивно понятным и удобным в использовании, чтобы пользователь мог легко настраивать параметры орнамента и видеть результаты своих действий.
* Настройка фона орнамента: Программа должна предоставлять пользователю возможность гибко настраивать фон для орнамента: устанавливать его цвет или фоновое изображение.

3.2. Требования к надежности

Программа должна предоставлять результат в виде визуального представления орнамента «Окружности по окружности» при любых введённых в неё данных.

3.3. Условия эксплуатации

Условия эксплуатации наследуются от условий эксплуатации устройства, на котором запущена программа.

3.4. Требования к составу и параметрам технических средств

Минимальные требования к техническим характеристикам компьютера для работы программы:

процессор с тактовой частотой 1 ГГц;

оперативная память объемом не менее 2 ГБ;

жесткий диск с объемом не менее 10 ГБ;

монитор;

мышь;

клавиатура.

3.5. Требования к информационной и программной совместимости

Программа должна корректно работать на устройствах под управлением операционной системы Windows версии 7 и выше с установленным пакетом Visual C++ Redist 2019 и выше.

3.6. Требования к маркировке и упаковке

Носитель информации или его упаковка должны иметь обозначение, идентифицирующее нахождение на носителе разработанной программы, а также её документации.

3.7. Требования к транспортированию и хранению

Требования к транспортировке наследуются от носителя информации, на котором расположена программа. Условия транспортирования и хранения должны гарантировать целостность информации на носителе, на котором расположена программа.

3.8. Специальные требования

Ограничение доступа людям, подверженным эпилептическим расстройствам.

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 – алгоритм работы программы

3. Физическое проектирование – структуры данных и спецификация функций

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 – 1.5, 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 |
| Написание ТЗ | до 20.05.2024 | Расписано ТЗ | Выполнено |
| Разработка программы | 28.04.2024-07.06.2024 | Разработана программа | Выполнено |
| Написание руководства пользователя | 07.06.2024-08.06.2024 | Расписано руководство пользователя | Выполнено |
| Написание РПЗ | 20.05.2024-09.06.2024 | Расписано РПЗ | Выполнено |

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 | 20.05.2024 | Оформленное ТЗ | Выполнено |
| Разработка программы | 24.05.2024 | Неконечная версия программы | Выполнено |
| Доработка программы | 07.06.2024 | Конечная версия программы | Выполнено |
| Оформление руководство пользователя | 08.06.2024 | Оформленное руководство пользователя | Выполнено |
| Оформление РПЗ | 09.06.2024 | Оформленная РПЗ | Выполнено |
| Сдача РПЗ | 10.06.2024 | Оценка за курсовую работу | Выполнено |

# Приложение 2. Руководство пользователя

1. Общие сведения о программе

Программа представляет собой графическое приложение для построения орнамента «Окружности по окружности» (рис. П2.1).

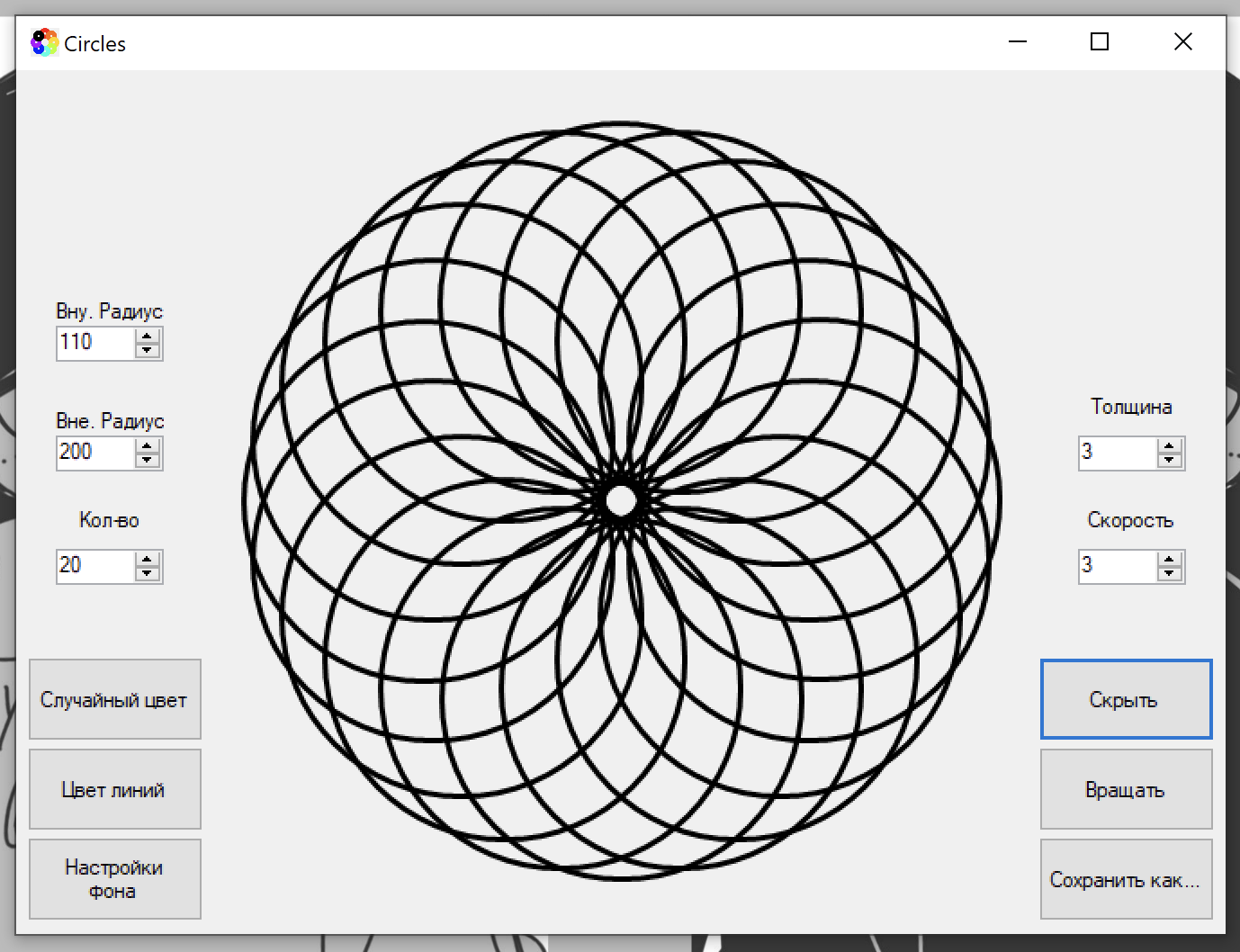


Рис. П2.1. Окно программы с орнаментом

Для создания орнамента необходимо задание таких параметров как: радиусы и количество окружностей, а также параметров толщины линии, скорости вращения, цвета линий и цвета фона. Параметры орнамента задаются при помощи цифирных блоков, а различные действия с орнаментом производятся при помощи соответствующих кнопок.

2. Описание установки

Описываемая программа не требует какой-либо установки на компьютер и является переносной. Для корректной работы необходимо установить Visual C++ Runtime 2019 64-разрядную версию и выше.

3. Описание запуска

Для начала работы с программой достаточно запустить её исполняемый файл, после чего на экране появится рабочее окно программы (рис. П2.2).

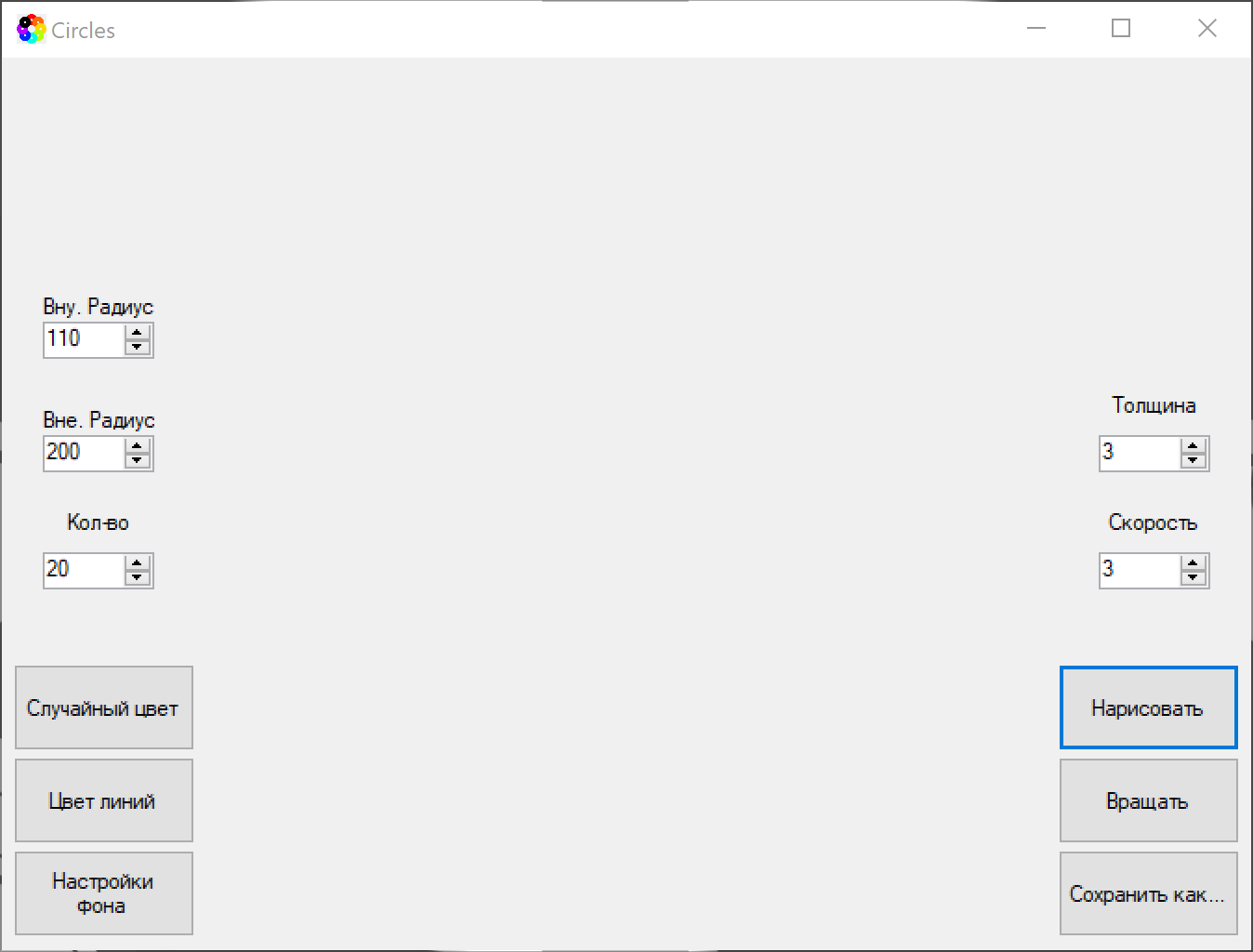


Рис. П2.2: Открытое окно программы

4. Описание работы с программой

После запуска программы пользователь имеет возможность установить параметры орнамента или оставить параметры по умолчанию. Пользователь может установить следующие параметры (рис. П2.3):

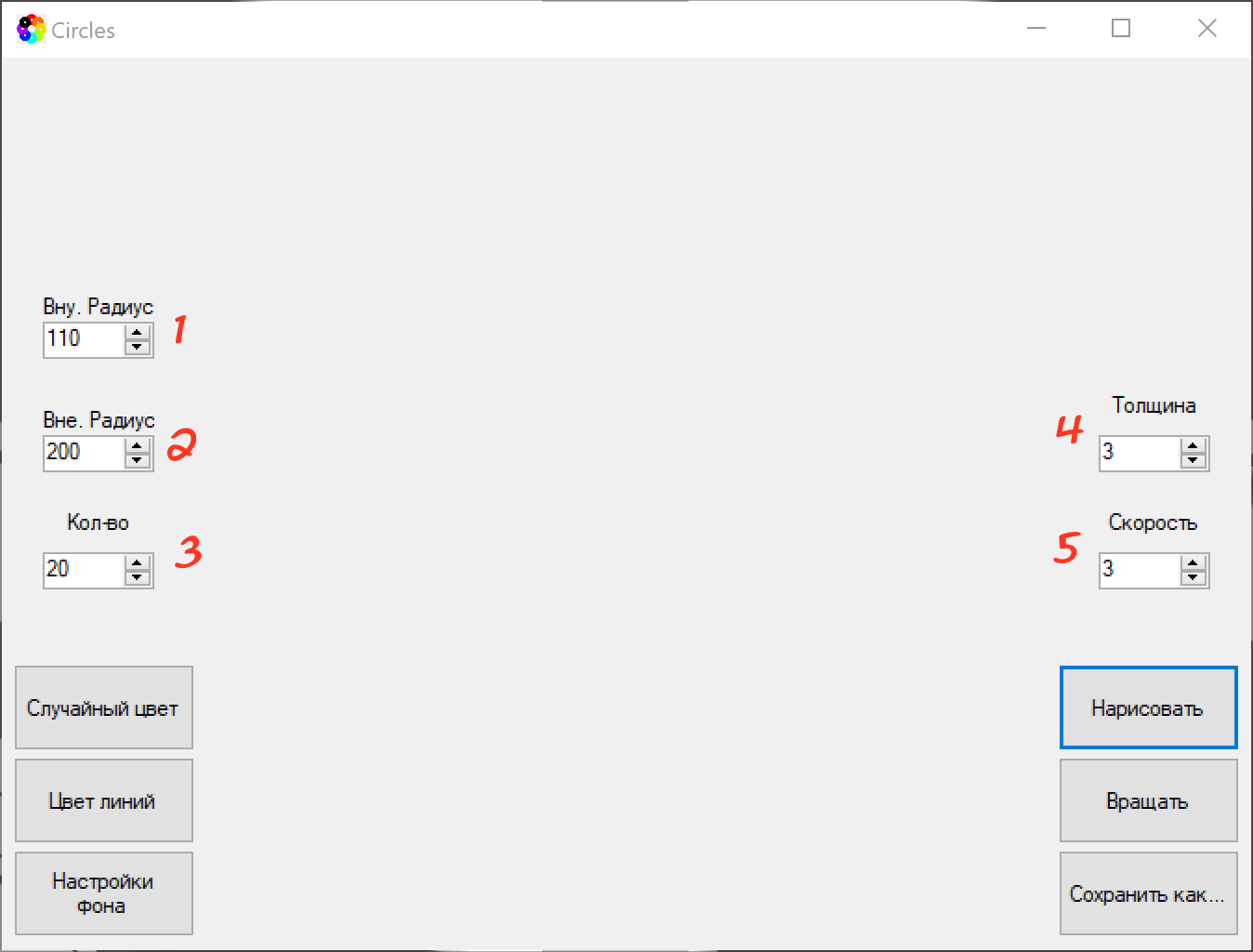


Рис. П2. 3. Параметры орнамента

1. Установка радиуса внутренней окружности, по которой устанавливаются остальные окружности. Задаётся положительными целыми числами максимальное значение – 9999. Минимальное значение – 1;
2. Установка радиуса внешних окружностей, которые будут вращаться по внутренней окружности. Задаётся положительными целыми числами максимальное значение – 9999. Минимальное значение – 1;
3. Установка количества вращающихся окружностей;
4. Установка толщины линий, которыми будут строится орнамент;
5. Установка скорости вращения орнамента.

Действия с орнаментом производятся при помощи кнопок справа (рис. П2.4).

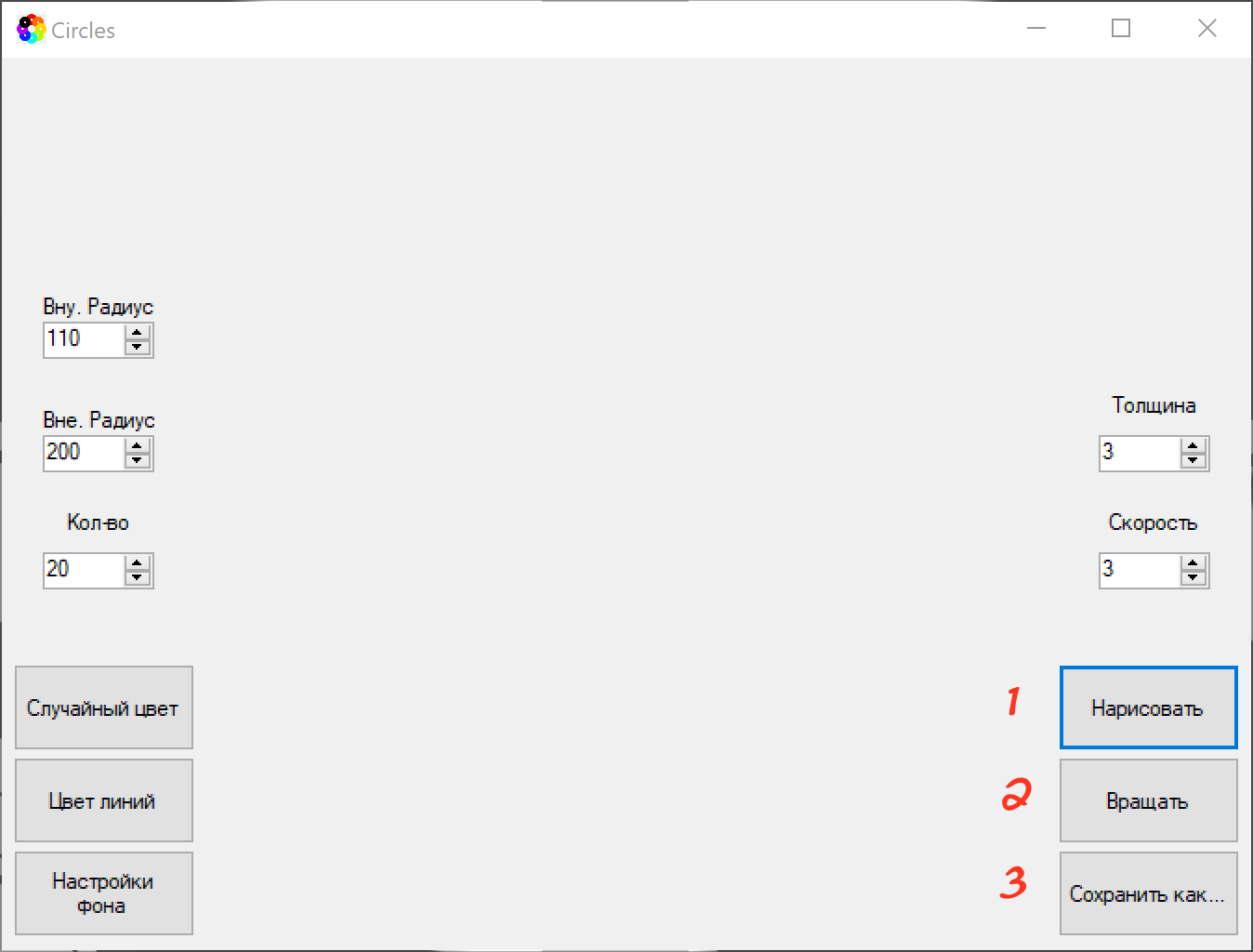


Рис. П2.4. Кнопки управления орнаментом

1. Кнопка запускающая отрисовку орнамента по заданным параметрам. При повторном нажатии на кнопку отрисовка орнамента прекращается;
2. Кнопка запускает вращение орнамента или останавливает его при повторном нажатии;
3. Кнопка отрывает диалог сохранения файла. Изображение орнамента сохраняется в файл, имя и расширение которого задаётся пользователем.

При помощи кнопок слева можно установить цвета линий орнамента, а также настроить фон (рис. П2.5).

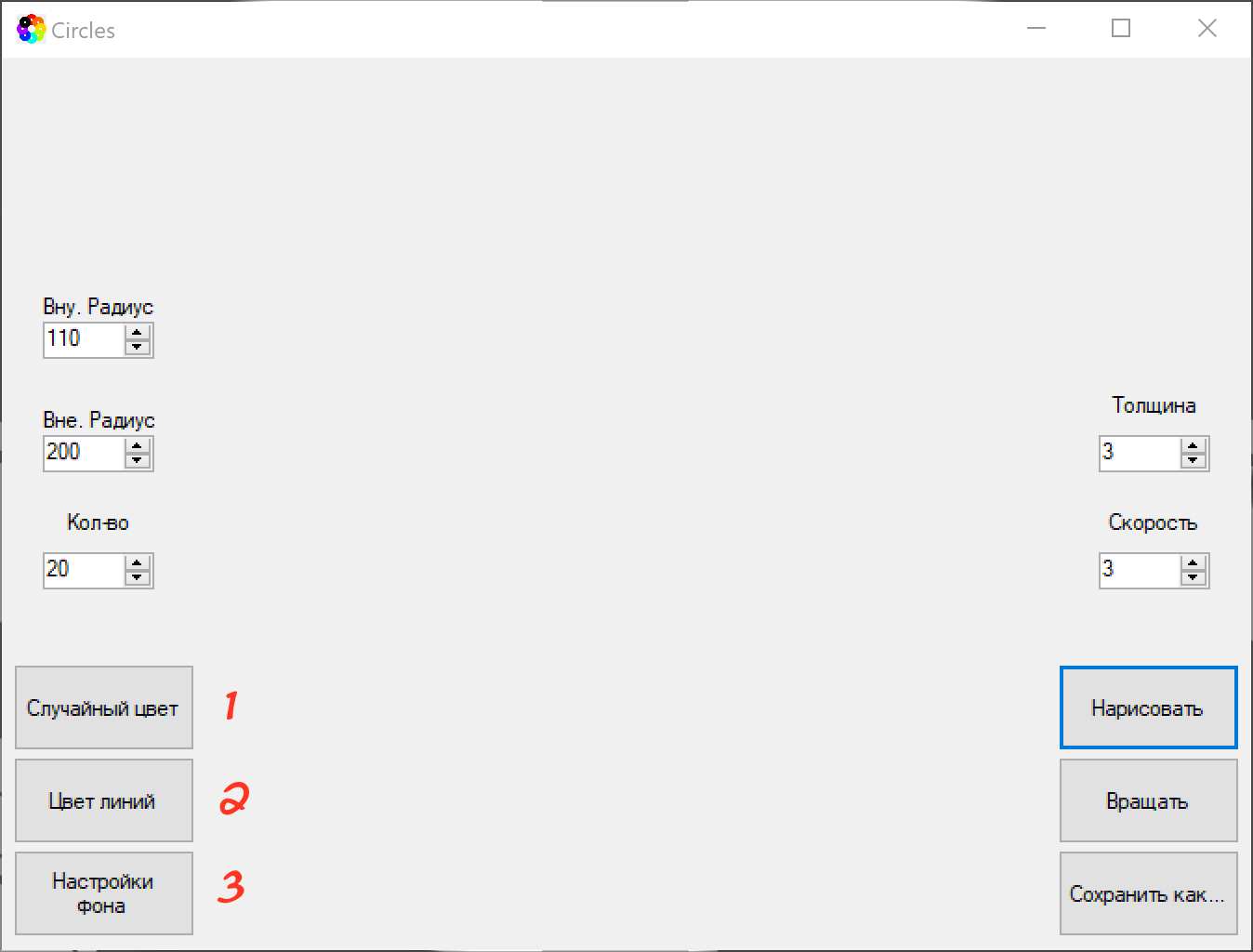


Рис. П2. 5. Кнопки настройки цвета

1. Устанавливает случайный цвет для каждой из окружностей;
2. Вызывает диалог выбора цвета для линий окружностей. При нажатии на диалоговую кнопку Ок цвет применяется, при нажатии на кнопку Отмена изменения не вносятся;
3. Открывает меню настройки фона для орнамента.

При нажатии на кнопку 3 открывается меню настройки фона для орнамента (рис. П2.6).

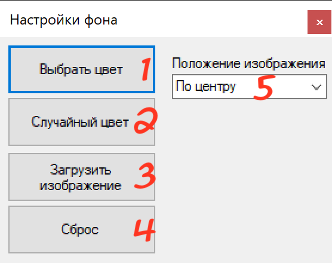


Рис. П2. 6: Меню настроек фона

1. Вызывает диалог выбора цвета для фона. При нажатии на диалоговую кнопку Ок цвет применяется, при нажатии на кнопку Отмена изменения не вносятся;
2. Устанавливает случайный цвет для фона;
3. Открывает диалог выбора файла изображения, которое затем будет установлено в качестве фонового изображения. Поддерживаются расширения .png, .jpg, .bmp, .gif;
4. Сбрасывает фон орнамента к стандартному, который был при запуске программы;
5. Устанавливает режим расположения фонового рисунка.
   1. По центру – устанавливает изображение в центре окна;
   2. Растянуть – растягивает изображение по размеру окна;
   3. Плитки – создаёт плиточный узор из изображений;
   4. Приближение – увеличивает изображение для заполнения окна по вертикали;
   5. По умолчанию – устанавливает изображение в левый верхний угол.

Пользователь в праве изменять размер окна программы, при этом орнамент всегда будет оставаться в центре. При изменении окна и сохранении изображения оно будет иметь размер такой же какое имеет окно приложения.

# Приложение 3. Программный код

Файл MyForm.cpp

|  |
| --- |
| #include "MyForm.h"  #include "BackSettings.h"  using namespace System;  using namespace System::Windows::Forms;  [STAThread]  int main(array<String^>^ args)  {  Application::EnableVisualStyles();  Application::SetCompatibleTextRenderingDefault(false);  Circles::MyForm form;  Application::Run(% form);  }  void showBackSettings() {  Circles::BackSettings^ form;  form = gcnew Circles::BackSettings();  form->Show();  } |

Файл MyForm.h

|  |
| --- |
| #pragma once  #include <ctime>  #include <cstdlib>  void showBackSettings();  namespace Circles {  using namespace System;  using namespace System::ComponentModel;  using namespace System::Collections;  using namespace System::Windows::Forms;  using namespace System::Data;  using namespace System::Drawing;  using namespace System::Drawing::Drawing2D;  using namespace System::Threading;  /// <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;  }  }  public: System::Windows::Forms::PictureBox^ canvas;  private: System::Windows::Forms::Button^ buttonDraw;  private: System::Windows::Forms::Button^ buttonColor;  private: System::Windows::Forms::ColorDialog^ colorDialogLine;  private: System::Windows::Forms::NumericUpDown^ radiusOut;  private: System::Windows::Forms::TextBox^ textBox1;  private: System::Windows::Forms::NumericUpDown^ lineThickness;  private: System::Windows::Forms::TextBox^ textBox2;  public: System::Windows::Forms::NumericUpDown^ circlesCount;  private: System::Windows::Forms::TextBox^ textBox3;  private: System::Windows::Forms::Button^ buttonRotate;  private: System::Windows::Forms::NumericUpDown^ rotationSpeed;  private: System::Windows::Forms::TextBox^ textBox4;  private: System::Windows::Forms::NumericUpDown^ radiusIn;  private: System::Windows::Forms::TextBox^ textBox5;  private: System::Windows::Forms::Button^ ButtonRandom;  private: System::Windows::Forms::Button^ buttonSave;  private: System::Windows::Forms::SaveFileDialog^ saveImageDialog;  private: System::Windows::Forms::Button^ buttonBackSettings;  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->canvas = (gcnew System::Windows::Forms::PictureBox());  this->buttonDraw = (gcnew System::Windows::Forms::Button());  this->buttonColor = (gcnew System::Windows::Forms::Button());  this->colorDialogLine = (gcnew System::Windows::Forms::ColorDialog());  this->radiusOut = (gcnew System::Windows::Forms::NumericUpDown());  this->textBox1 = (gcnew System::Windows::Forms::TextBox());  this->lineThickness = (gcnew System::Windows::Forms::NumericUpDown());  this->textBox2 = (gcnew System::Windows::Forms::TextBox());  this->circlesCount = (gcnew System::Windows::Forms::NumericUpDown());  this->textBox3 = (gcnew System::Windows::Forms::TextBox());  this->buttonRotate = (gcnew System::Windows::Forms::Button());  this->rotationSpeed = (gcnew System::Windows::Forms::NumericUpDown());  this->textBox4 = (gcnew System::Windows::Forms::TextBox());  this->radiusIn = (gcnew System::Windows::Forms::NumericUpDown());  this->textBox5 = (gcnew System::Windows::Forms::TextBox());  this->ButtonRandom = (gcnew System::Windows::Forms::Button());  this->buttonSave = (gcnew System::Windows::Forms::Button());  this->saveImageDialog = (gcnew System::Windows::Forms::SaveFileDialog());  this->buttonBackSettings = (gcnew System::Windows::Forms::Button());  (cli::safe\_cast<System::ComponentModel::ISupportInitialize^>(this->canvas))->BeginInit();  (cli::safe\_cast<System::ComponentModel::ISupportInitialize^>(this->radiusOut))->BeginInit();  (cli::safe\_cast<System::ComponentModel::ISupportInitialize^>(this->lineThickness))->BeginInit();  (cli::safe\_cast<System::ComponentModel::ISupportInitialize^>(this->circlesCount))->BeginInit();  (cli::safe\_cast<System::ComponentModel::ISupportInitialize^>(this->rotationSpeed))->BeginInit();  (cli::safe\_cast<System::ComponentModel::ISupportInitialize^>(this->radiusIn))->BeginInit();  this->SuspendLayout();  //  // canvas  //  this->canvas->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->canvas->Location = System::Drawing::Point(1, -1);  this->canvas->Name = L"canvas";  this->canvas->Size = System::Drawing::Size(1343, 925);  this->canvas->TabIndex = 0;  this->canvas->TabStop = false;  //  // buttonDraw  //  this->buttonDraw->Anchor = static\_cast<System::Windows::Forms::AnchorStyles>((System::Windows::Forms::AnchorStyles::Bottom | System::Windows::Forms::AnchorStyles::Right));  this->buttonDraw->Location = System::Drawing::Point(1135, 627);  this->buttonDraw->Name = L"buttonDraw";  this->buttonDraw->Size = System::Drawing::Size(195, 90);  this->buttonDraw->TabIndex = 1;  this->buttonDraw->Text = L"Нарисовать";  this->buttonDraw->UseVisualStyleBackColor = true;  this->buttonDraw->Click += gcnew System::EventHandler(this, &MyForm::buttonDraw\_Click);  //  // buttonColor  //  this->buttonColor->Anchor = static\_cast<System::Windows::Forms::AnchorStyles>((System::Windows::Forms::AnchorStyles::Bottom | System::Windows::Forms::AnchorStyles::Left));  this->buttonColor->Location = System::Drawing::Point(12, 723);  this->buttonColor->Name = L"buttonColor";  this->buttonColor->Size = System::Drawing::Size(195, 90);  this->buttonColor->TabIndex = 2;  this->buttonColor->Text = L"Цвет линий";  this->buttonColor->UseVisualStyleBackColor = true;  this->buttonColor->Click += gcnew System::EventHandler(this, &MyForm::buttonColor\_Click);  //  // radiusOut  //  this->radiusOut->Anchor = static\_cast<System::Windows::Forms::AnchorStyles>((System::Windows::Forms::AnchorStyles::Bottom | System::Windows::Forms::AnchorStyles::Left));  this->radiusOut->Location = System::Drawing::Point(43, 391);  this->radiusOut->Maximum = System::Decimal(gcnew cli::array< System::Int32 >(4) { 9999, 0, 0, 0 });  this->radiusOut->Minimum = System::Decimal(gcnew cli::array< System::Int32 >(4) { 1, 0, 0, 0 });  this->radiusOut->Name = L"radiusOut";  this->radiusOut->Size = System::Drawing::Size(120, 31);  this->radiusOut->TabIndex = 3;  this->radiusOut->Value = System::Decimal(gcnew cli::array< System::Int32 >(4) { 200, 0, 0, 0 });  this->radiusOut->ValueChanged += gcnew System::EventHandler(this, &MyForm::radiusOut\_ValueChanged);  //  // textBox1  //  this->textBox1->Anchor = static\_cast<System::Windows::Forms::AnchorStyles>((System::Windows::Forms::AnchorStyles::Bottom | System::Windows::Forms::AnchorStyles::Left));  this->textBox1->BackColor = System::Drawing::SystemColors::Control;  this->textBox1->BorderStyle = System::Windows::Forms::BorderStyle::None;  this->textBox1->Location = System::Drawing::Point(43, 361);  this->textBox1->Name = L"textBox1";  this->textBox1->ReadOnly = true;  this->textBox1->Size = System::Drawing::Size(120, 24);  this->textBox1->TabIndex = 4;  this->textBox1->Text = L"Вне. Радиус";  this->textBox1->TextAlign = System::Windows::Forms::HorizontalAlignment::Center;  //  // lineThickness  //  this->lineThickness->Anchor = static\_cast<System::Windows::Forms::AnchorStyles>((System::Windows::Forms::AnchorStyles::Bottom | System::Windows::Forms::AnchorStyles::Right));  this->lineThickness->Location = System::Drawing::Point(1179, 391);  this->lineThickness->Minimum = System::Decimal(gcnew cli::array< System::Int32 >(4) { 1, 0, 0, 0 });  this->lineThickness->Name = L"lineThickness";  this->lineThickness->Size = System::Drawing::Size(120, 31);  this->lineThickness->TabIndex = 5;  this->lineThickness->Value = System::Decimal(gcnew cli::array< System::Int32 >(4) { 3, 0, 0, 0 });  this->lineThickness->ValueChanged += gcnew System::EventHandler(this, &MyForm::lineThickness\_ValueChanged);  //  // textBox2  //  this->textBox2->Anchor = static\_cast<System::Windows::Forms::AnchorStyles>((System::Windows::Forms::AnchorStyles::Bottom | System::Windows::Forms::AnchorStyles::Right));  this->textBox2->BackColor = System::Drawing::SystemColors::Control;  this->textBox2->BorderStyle = System::Windows::Forms::BorderStyle::None;  this->textBox2->Location = System::Drawing::Point(1179, 347);  this->textBox2->Name = L"textBox2";  this->textBox2->ReadOnly = true;  this->textBox2->Size = System::Drawing::Size(120, 24);  this->textBox2->TabIndex = 6;  this->textBox2->Text = L"Толщина";  this->textBox2->TextAlign = System::Windows::Forms::HorizontalAlignment::Center;  //  // circlesCount  //  this->circlesCount->Anchor = static\_cast<System::Windows::Forms::AnchorStyles>((System::Windows::Forms::AnchorStyles::Bottom | System::Windows::Forms::AnchorStyles::Left));  this->circlesCount->Location = System::Drawing::Point(43, 512);  this->circlesCount->Minimum = System::Decimal(gcnew cli::array< System::Int32 >(4) { 1, 0, 0, 0 });  this->circlesCount->Name = L"circlesCount";  this->circlesCount->Size = System::Drawing::Size(120, 31);  this->circlesCount->TabIndex = 7;  this->circlesCount->Value = System::Decimal(gcnew cli::array< System::Int32 >(4) { 20, 0, 0, 0 });  this->circlesCount->ValueChanged += gcnew System::EventHandler(this, &MyForm::circlesCount\_ValueChanged);  //  // textBox3  //  this->textBox3->Anchor = static\_cast<System::Windows::Forms::AnchorStyles>((System::Windows::Forms::AnchorStyles::Bottom | System::Windows::Forms::AnchorStyles::Left));  this->textBox3->BackColor = System::Drawing::SystemColors::Control;  this->textBox3->BorderStyle = System::Windows::Forms::BorderStyle::None;  this->textBox3->Location = System::Drawing::Point(43, 468);  this->textBox3->Name = L"textBox3";  this->textBox3->ReadOnly = true;  this->textBox3->Size = System::Drawing::Size(120, 24);  this->textBox3->TabIndex = 8;  this->textBox3->Text = L"Кол-во";  this->textBox3->TextAlign = System::Windows::Forms::HorizontalAlignment::Center;  //  // buttonRotate  //  this->buttonRotate->Anchor = static\_cast<System::Windows::Forms::AnchorStyles>((System::Windows::Forms::AnchorStyles::Bottom | System::Windows::Forms::AnchorStyles::Right));  this->buttonRotate->Location = System::Drawing::Point(1135, 723);  this->buttonRotate->Name = L"buttonRotate";  this->buttonRotate->Size = System::Drawing::Size(195, 90);  this->buttonRotate->TabIndex = 9;  this->buttonRotate->Text = L"Вращать";  this->buttonRotate->UseVisualStyleBackColor = true;  this->buttonRotate->Click += gcnew System::EventHandler(this, &MyForm::buttonRotate\_Click);  //  // rotationSpeed  //  this->rotationSpeed->Anchor = static\_cast<System::Windows::Forms::AnchorStyles>((System::Windows::Forms::AnchorStyles::Bottom | System::Windows::Forms::AnchorStyles::Right));  this->rotationSpeed->Location = System::Drawing::Point(1179, 512);  this->rotationSpeed->Minimum = System::Decimal(gcnew cli::array< System::Int32 >(4) { 100, 0, 0, System::Int32::MinValue });  this->rotationSpeed->Name = L"rotationSpeed";  this->rotationSpeed->Size = System::Drawing::Size(120, 31);  this->rotationSpeed->TabIndex = 10;  this->rotationSpeed->Value = System::Decimal(gcnew cli::array< System::Int32 >(4) { 3, 0, 0, 0 });  this->rotationSpeed->ValueChanged += gcnew System::EventHandler(this, &MyForm::rotationSpeed\_ValueChanged);  //  // textBox4  //  this->textBox4->Anchor = static\_cast<System::Windows::Forms::AnchorStyles>((System::Windows::Forms::AnchorStyles::Bottom | System::Windows::Forms::AnchorStyles::Right));  this->textBox4->BackColor = System::Drawing::SystemColors::Control;  this->textBox4->BorderStyle = System::Windows::Forms::BorderStyle::None;  this->textBox4->Location = System::Drawing::Point(1179, 468);  this->textBox4->Name = L"textBox4";  this->textBox4->ReadOnly = true;  this->textBox4->Size = System::Drawing::Size(120, 24);  this->textBox4->TabIndex = 11;  this->textBox4->Text = L"Cкорость";  this->textBox4->TextAlign = System::Windows::Forms::HorizontalAlignment::Center;  //  // radiusIn  //  this->radiusIn->Anchor = static\_cast<System::Windows::Forms::AnchorStyles>((System::Windows::Forms::AnchorStyles::Bottom | System::Windows::Forms::AnchorStyles::Left));  this->radiusIn->Location = System::Drawing::Point(43, 274);  this->radiusIn->Maximum = System::Decimal(gcnew cli::array< System::Int32 >(4) { 9999, 0, 0, 0 });  this->radiusIn->Minimum = System::Decimal(gcnew cli::array< System::Int32 >(4) { 1, 0, 0, 0 });  this->radiusIn->Name = L"radiusIn";  this->radiusIn->Size = System::Drawing::Size(120, 31);  this->radiusIn->TabIndex = 12;  this->radiusIn->Value = System::Decimal(gcnew cli::array< System::Int32 >(4) { 110, 0, 0, 0 });  this->radiusIn->ValueChanged += gcnew System::EventHandler(this, &MyForm::radiusIn\_ValueChanged);  //  // textBox5  //  this->textBox5->Anchor = static\_cast<System::Windows::Forms::AnchorStyles>((System::Windows::Forms::AnchorStyles::Bottom | System::Windows::Forms::AnchorStyles::Left));  this->textBox5->BackColor = System::Drawing::SystemColors::Control;  this->textBox5->BorderStyle = System::Windows::Forms::BorderStyle::None;  this->textBox5->Location = System::Drawing::Point(43, 244);  this->textBox5->Name = L"textBox5";  this->textBox5->Size = System::Drawing::Size(120, 24);  this->textBox5->TabIndex = 13;  this->textBox5->Text = L"Вну. Радиус";  this->textBox5->TextAlign = System::Windows::Forms::HorizontalAlignment::Center;  //  // ButtonRandom  //  this->ButtonRandom->Anchor = static\_cast<System::Windows::Forms::AnchorStyles>((System::Windows::Forms::AnchorStyles::Bottom | System::Windows::Forms::AnchorStyles::Left));  this->ButtonRandom->Location = System::Drawing::Point(12, 627);  this->ButtonRandom->Name = L"ButtonRandom";  this->ButtonRandom->Size = System::Drawing::Size(195, 90);  this->ButtonRandom->TabIndex = 15;  this->ButtonRandom->Text = L"Случайный цвет";  this->ButtonRandom->UseVisualStyleBackColor = true;  this->ButtonRandom->Click += gcnew System::EventHandler(this, &MyForm::ButtonRainbow\_Click);  //  // buttonSave  //  this->buttonSave->Anchor = static\_cast<System::Windows::Forms::AnchorStyles>((System::Windows::Forms::AnchorStyles::Bottom | System::Windows::Forms::AnchorStyles::Right));  this->buttonSave->Location = System::Drawing::Point(1135, 820);  this->buttonSave->Name = L"buttonSave";  this->buttonSave->Size = System::Drawing::Size(195, 90);  this->buttonSave->TabIndex = 16;  this->buttonSave->Text = L"Сохранить как...";  this->buttonSave->UseVisualStyleBackColor = true;  this->buttonSave->Click += gcnew System::EventHandler(this, &MyForm::buttonSave\_Click);  //  // buttonBackSettings  //  this->buttonBackSettings->Anchor = static\_cast<System::Windows::Forms::AnchorStyles>((System::Windows::Forms::AnchorStyles::Bottom | System::Windows::Forms::AnchorStyles::Left));  this->buttonBackSettings->Location = System::Drawing::Point(12, 820);  this->buttonBackSettings->Name = L"buttonBackSettings";  this->buttonBackSettings->Size = System::Drawing::Size(195, 90);  this->buttonBackSettings->TabIndex = 17;  this->buttonBackSettings->Text = L"Настройки фона";  this->buttonBackSettings->UseVisualStyleBackColor = true;  this->buttonBackSettings->Click += gcnew System::EventHandler(this, &MyForm::buttonBackSettings\_Click);  //  // MyForm  //  this->AutoScaleDimensions = System::Drawing::SizeF(12, 25);  this->AutoScaleMode = System::Windows::Forms::AutoScaleMode::Font;  this->ClientSize = System::Drawing::Size(1344, 924);  this->Controls->Add(this->buttonBackSettings);  this->Controls->Add(this->buttonSave);  this->Controls->Add(this->ButtonRandom);  this->Controls->Add(this->textBox5);  this->Controls->Add(this->radiusIn);  this->Controls->Add(this->textBox4);  this->Controls->Add(this->rotationSpeed);  this->Controls->Add(this->buttonRotate);  this->Controls->Add(this->textBox3);  this->Controls->Add(this->circlesCount);  this->Controls->Add(this->textBox2);  this->Controls->Add(this->lineThickness);  this->Controls->Add(this->textBox1);  this->Controls->Add(this->radiusOut);  this->Controls->Add(this->buttonColor);  this->Controls->Add(this->buttonDraw);  this->Controls->Add(this->canvas);  this->Icon = (cli::safe\_cast<System::Drawing::Icon^>(resources->GetObject(L"$this.Icon")));  this->MinimumSize = System::Drawing::Size(800, 800);  this->Name = L"MyForm";  this->Text = L"Circles";  this->Resize += gcnew System::EventHandler(this, &MyForm::MyForm\_Resize);  (cli::safe\_cast<System::ComponentModel::ISupportInitialize^>(this->canvas))->EndInit();  (cli::safe\_cast<System::ComponentModel::ISupportInitialize^>(this->radiusOut))->EndInit();  (cli::safe\_cast<System::ComponentModel::ISupportInitialize^>(this->lineThickness))->EndInit();  (cli::safe\_cast<System::ComponentModel::ISupportInitialize^>(this->circlesCount))->EndInit();  (cli::safe\_cast<System::ComponentModel::ISupportInitialize^>(this->rotationSpeed))->EndInit();  (cli::safe\_cast<System::ComponentModel::ISupportInitialize^>(this->radiusIn))->EndInit();  this->ResumeLayout(false);  this->PerformLayout();  }  #pragma endregion  private: System::Void radiusIn\_ValueChanged(System::Object^ sender, System::EventArgs^ e) {  canvas->Refresh();  }  private: System::Void radiusOut\_ValueChanged(System::Object^ sender, System::EventArgs^ e) {  canvas->Refresh();  }  private: System::Void lineThickness\_ValueChanged(System::Object^ sender, System::EventArgs^ e) {  canvas->Refresh();  }  private: System::Void circlesCount\_ValueChanged(System::Object^ sender, System::EventArgs^ e) {  canvas->Refresh();  }  private: System::Void rotationSpeed\_ValueChanged(System::Object^ sender, System::EventArgs^ e) {  canvas->Refresh();  }  private:  bool drawed = false;  bool rotation = false;  bool randcolor = false;  float rotation\_angle = 90.0F;  System::Void buttonDraw\_Click(System::Object^ sender, System::EventArgs^ e) {  if (!drawed) {  canvas->Paint += gcnew System::Windows::Forms::PaintEventHandler(this, &MyForm::Draw);  canvas->Refresh();  buttonDraw->Text = "Скрыть";  drawed = true;  }  else {  canvas->Invalidate();  canvas->Paint -= gcnew System::Windows::Forms::PaintEventHandler(this, &MyForm::Draw);  buttonDraw->Text = "Нарисовать";  drawed = false;  }  }  static System::Windows::Forms::Timer^ rotTimer = gcnew System::Windows::Forms::Timer;  System::Void buttonRotate\_Click(System::Object^ sender, System::EventArgs^ e) {  rotTimer->Interval = 1;  if (!rotation) {  rotTimer->Tick += gcnew EventHandler(this, &MyForm::Rotate);  rotTimer->Start();  buttonRotate->Text = "Остановить";  rotation = true;  }  else {  rotTimer->Tick -= gcnew EventHandler(this, &MyForm::Rotate);  rotTimer->Stop();  buttonRotate->Text = "Вращать";  rotation = false;  }  }  System::Void buttonColor\_Click(System::Object^ sender, System::EventArgs^ e) {  colorDialogLine->ShowDialog();  canvas->Refresh();  }  array <System::Drawing::Color>^ colorArr;  System::Drawing::Color GetRandColor(int index) {  if (!colorArr) {  srand(time(0) + index);  colorArr = gcnew array<System::Drawing::Color>((int)circlesCount->Maximum);  for (int i = 0; i < circlesCount->Maximum; i++) {  colorArr[i] = System::Drawing::Color::FromArgb(rand() % 255, rand() % 255, rand() % 255);  }  }  return colorArr[index];  }  System::Void ButtonRainbow\_Click(System::Object^ sender, System::EventArgs^ e) {  if (randcolor) {  ButtonRandom->Text = "Случайный цвет";  randcolor = false;  delete colorArr;  }  else {  ButtonRandom->Text = "Сплошной цвет";  randcolor = true;  }  canvas->Refresh();  }  void Draw(System::Object^ sender, System::Windows::Forms::PaintEventArgs^ event)  {  event->Graphics->SmoothingMode = SmoothingMode::AntiAlias;  int count = (int)circlesCount->Value;  for (int i = 0; i < count; i++) {  Pen^ drawingPen = gcnew Pen(Brushes::Black);  if (randcolor) drawingPen->Color = GetRandColor(i);  else drawingPen->Color = colorDialogLine->Color;  drawingPen->Width = (float)lineThickness->Value;  drawingPen->LineJoin = System::Drawing::Drawing2D::LineJoin::Bevel;  DrawCircle(drawingPen, event, i);  delete drawingPen;  }  }  void DrawCircle(Pen^ drawer, System::Windows::Forms::PaintEventArgs^ event, int index)  {  int count = (int)circlesCount->Value;  int loc\_radius = (int)radiusIn->Value;  int radius = (int)radiusOut->Value;  double radians = System::Math::PI / 180;  double angle = 360.0F \* radians / count \* index + rotation\_angle \* radians;  int x = (canvas->Width / 2 - radius / 2) + (loc\_radius \* Math::Cos(angle));  int y = (canvas->Height / 2 - radius / 2) + (loc\_radius \* Math::Sin(angle));  event->Graphics->DrawEllipse(drawer, Rectangle(x, y, radius, radius));  }  System::Void Rotate(System::Object^ sender, System::EventArgs^ e) {  if (rotation\_angle >= 360.0F) rotation\_angle = 0.0F;  rotation\_angle += 1.0F \* (float)rotationSpeed->Value;  canvas->Refresh();  }  System::Void buttonSave\_Click(System::Object^ sender, System::EventArgs^ e) {  saveImageDialog = gcnew SaveFileDialog();  saveImageDialog->Filter = "Файл \"PNG\"|\*.png|Файл \"JPG\"|\*.jpg|Файл \"BMP\"|\*.bmp|Файл \"GIF\"|\*.gif";  saveImageDialog->Title = "Сохранить как изображение";  if (saveImageDialog->ShowDialog() == System::Windows::Forms::DialogResult::OK) {  Bitmap^ bmp = gcnew Bitmap(canvas->Width, canvas->Height);  Rectangle rect = \*gcnew Rectangle(0, 0, bmp->Width, bmp->Height);  canvas->DrawToBitmap(bmp, rect);  bmp->Save(saveImageDialog->FileName);  }  }    System::Void MyForm\_Resize(System::Object^ sender, System::EventArgs^ e) {  canvas->Refresh();  }    System::Void buttonBackSettings\_Click(System::Object^ sender, System::EventArgs^ e) {  showBackSettings();  }  };  } |

Файл BackSettings.h

|  |
| --- |
| #pragma once  #include "MyForm.h"  namespace Circles {  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 BackSettings  /// </summary>  public ref class BackSettings : public System::Windows::Forms::Form  {  public:  BackSettings(void)  {  InitializeComponent();  //  //TODO: добавьте код конструктора  //  }  protected:  /// <summary>  /// Освободить все используемые ресурсы.  /// </summary>  ~BackSettings()  {  if (components)  {  delete components;  }  }  private: System::Windows::Forms::Button^ buttonColor;  private: System::Windows::Forms::Button^ buttonRandom;  private: System::Windows::Forms::Button^ buttonImage;  private: System::Windows::Forms::ColorDialog^ colorDialogBack;  private: System::Windows::Forms::OpenFileDialog^ openFileDialog;  private: System::Windows::Forms::Button^ buttonReset;  private: System::Windows::Forms::ComboBox^ comboBox;  private: System::Windows::Forms::TextBox^ textBox1;  protected:  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)  {  this->buttonColor = (gcnew System::Windows::Forms::Button());  this->buttonRandom = (gcnew System::Windows::Forms::Button());  this->buttonImage = (gcnew System::Windows::Forms::Button());  this->colorDialogBack = (gcnew System::Windows::Forms::ColorDialog());  this->openFileDialog = (gcnew System::Windows::Forms::OpenFileDialog());  this->buttonReset = (gcnew System::Windows::Forms::Button());  this->comboBox = (gcnew System::Windows::Forms::ComboBox());  this->textBox1 = (gcnew System::Windows::Forms::TextBox());  this->SuspendLayout();  //  // buttonColor  //  this->buttonColor->Location = System::Drawing::Point(12, 12);  this->buttonColor->Name = L"buttonColor";  this->buttonColor->Size = System::Drawing::Size(250, 80);  this->buttonColor->TabIndex = 0;  this->buttonColor->Text = L"Выбрать цвет";  this->buttonColor->UseVisualStyleBackColor = true;  this->buttonColor->Click += gcnew System::EventHandler(this, &BackSettings::buttonBackColor\_Click);  //  // buttonRandom  //  this->buttonRandom->Location = System::Drawing::Point(12, 97);  this->buttonRandom->Name = L"buttonRandom";  this->buttonRandom->Size = System::Drawing::Size(250, 80);  this->buttonRandom->TabIndex = 1;  this->buttonRandom->Text = L"Случайный цвет";  this->buttonRandom->UseVisualStyleBackColor = true;  this->buttonRandom->Click += gcnew System::EventHandler(this, &BackSettings::buttonBackRandom\_Click);  //  // buttonImage  //  this->buttonImage->Location = System::Drawing::Point(12, 184);  this->buttonImage->Name = L"buttonImage";  this->buttonImage->Size = System::Drawing::Size(250, 80);  this->buttonImage->TabIndex = 2;  this->buttonImage->Text = L"Загрузить изображение";  this->buttonImage->UseVisualStyleBackColor = true;  this->buttonImage->Click += gcnew System::EventHandler(this, &BackSettings::buttonBackImage\_Click);  //  // openFileDialog  //  this->openFileDialog->FileName = L"openFileDialog";  //  // buttonReset  //  this->buttonReset->Location = System::Drawing::Point(12, 270);  this->buttonReset->Name = L"buttonReset";  this->buttonReset->Size = System::Drawing::Size(250, 80);  this->buttonReset->TabIndex = 3;  this->buttonReset->Text = L"Сброс";  this->buttonReset->UseVisualStyleBackColor = true;  this->buttonReset->Click += gcnew System::EventHandler(this, &BackSettings::buttonReset\_Click);  //  // comboBox  //  this->comboBox->FormattingEnabled = true;  this->comboBox->Items->AddRange(gcnew cli::array< System::Object^ >(5) {  L"По центру", L"Растянуть", L"Плитки", L"Приближение",  L"По умолчанию"  });  this->comboBox->Location = System::Drawing::Point(289, 59);  this->comboBox->Name = L"comboBox";  this->comboBox->Size = System::Drawing::Size(255, 33);  this->comboBox->TabIndex = 4;  this->comboBox->Text = L"По центру";  this->comboBox->SelectedIndexChanged += gcnew System::EventHandler(this, &BackSettings::comboBox\_SelectedIndexChanged);  //  // textBox1  //  this->textBox1->BackColor = System::Drawing::SystemColors::Control;  this->textBox1->BorderStyle = System::Windows::Forms::BorderStyle::None;  this->textBox1->Location = System::Drawing::Point(289, 29);  this->textBox1->Name = L"textBox1";  this->textBox1->Size = System::Drawing::Size(255, 24);  this->textBox1->TabIndex = 5;  this->textBox1->Text = L"Положение изображения";  this->textBox1->TextAlign = System::Windows::Forms::HorizontalAlignment::Center;  //  // BackSettings  //  this->AutoScaleDimensions = System::Drawing::SizeF(12, 25);  this->AutoScaleMode = System::Windows::Forms::AutoScaleMode::Font;  this->ClientSize = System::Drawing::Size(556, 363);  this->Controls->Add(this->textBox1);  this->Controls->Add(this->comboBox);  this->Controls->Add(this->buttonReset);  this->Controls->Add(this->buttonImage);  this->Controls->Add(this->buttonRandom);  this->Controls->Add(this->buttonColor);  this->FormBorderStyle = System::Windows::Forms::FormBorderStyle::FixedToolWindow;  this->Name = L"BackSettings";  this->Text = L"Настройки фона";  this->ResumeLayout(false);  this->PerformLayout();  }  #pragma endregion    private: MyForm^ ownerForm = (MyForm^)Application::OpenForms["MyForm"];  private: System::Void buttonBackColor\_Click(System::Object^ sender, System::EventArgs^ e) {  if (colorDialogBack->ShowDialog() == System::Windows::Forms::DialogResult::OK) {  ownerForm->canvas->BackColor = colorDialogBack->Color;  }  }  private: System::Void buttonBackRandom\_Click(System::Object^ sender, System::EventArgs^ e) {  ownerForm->canvas->BackColor = System::Drawing::Color::FromArgb(rand() % 255, rand() % 255, rand() % 255);  }  private: System::Void buttonBackImage\_Click(System::Object^ sender, System::EventArgs^ e) {  openFileDialog = gcnew OpenFileDialog();  openFileDialog->Filter = "Файл \"PNG\"|\*.png|Файл \"JPG\"|\*.jpg|Файл \"BMP\"|\*.bmp|Файл \"GIF\"|\*.gif|Все файлы|\*";  openFileDialog->Title = "Открыть фоновое изображение";  if (openFileDialog->ShowDialog() == System::Windows::Forms::DialogResult::OK) {  ownerForm->canvas->BackgroundImage = gcnew Bitmap(openFileDialog->FileName);  ownerForm->canvas->BackgroundImageLayout = ImageLayout::Center;  }  }  private: System::Void buttonReset\_Click(System::Object^ sender, System::EventArgs^ e) {  ownerForm->canvas->BackColor = System::Drawing::SystemColors::Control;  ownerForm->canvas->BackgroundImage = nullptr;  }  private: System::Void comboBox\_SelectedIndexChanged(System::Object^ sender, System::EventArgs^ e) {  switch (comboBox->SelectedIndex) {  case 0: {  ownerForm->canvas->BackgroundImageLayout = ImageLayout::Center;  break;  }  case 1: {  ownerForm->canvas->BackgroundImageLayout = ImageLayout::Stretch;  break;  }  case 2: {  ownerForm->canvas->BackgroundImageLayout = ImageLayout::Tile;  break;  }  case 3: {  ownerForm->canvas->BackgroundImageLayout = ImageLayout::Zoom;  break;  }  case 4: {  ownerForm->canvas->BackgroundImageLayout = ImageLayout::None;  }  }  }  };  } |