ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«РОССИЙСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ НЕФТИ И ГАЗА
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)
ИМЕНИ И.М. ГУБКИНА»

ФАКУЛЬТЕТ КОМПЛЕКСНОЙ БЕЗОПАСНОСТИ ТЭК
КАФЕДРА БЕЗОПАСНОСТИ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

Лабораторная работа №4

по дисциплине «Специализированные языки и технологии программирования»

на тему «СОЗДАНИЕ КАСТОМНОГО ВИДЖЕТА С ПЕРЕОПРЕДЕЛЕНИЕМ СОБЫТИЙ ОТРИСОВКИ»

Выполнил студент:
группы КА-22-06
Воронин Алексей Дмитриевич

Преподаватель:
Греков Владимир Сергеевич

Москва, 2025

Оглавление

Цель работы 3

Задание 3

ЧАСТЬ 1 - Детальные инструкции к выполнению 4

ЧАСТЬ 2 - Интеграция кастомного виджета в приложение 10

Самостоятельная работа 13

ЗАКЛЮЧЕНИЕ 14

Контрольные вопросы 15

Цель работы

Научиться создавать кастомные виджеты в Qt, используя механизмы переопределения событий отрисовки. Разработать приложение, которое демонстрирует использование созданного кастомного виджета.

Задание

Разработка кастомного виджета "Индикатор выполнения".

ЧАСТЬ 1 - Детальные инструкции к выполнению

Создайте новый класс C++. В созданном проекте нажмите правой кнопкой мыши по папке Source Files и выберите Add New.

Назовите класс ProgressIndicator.

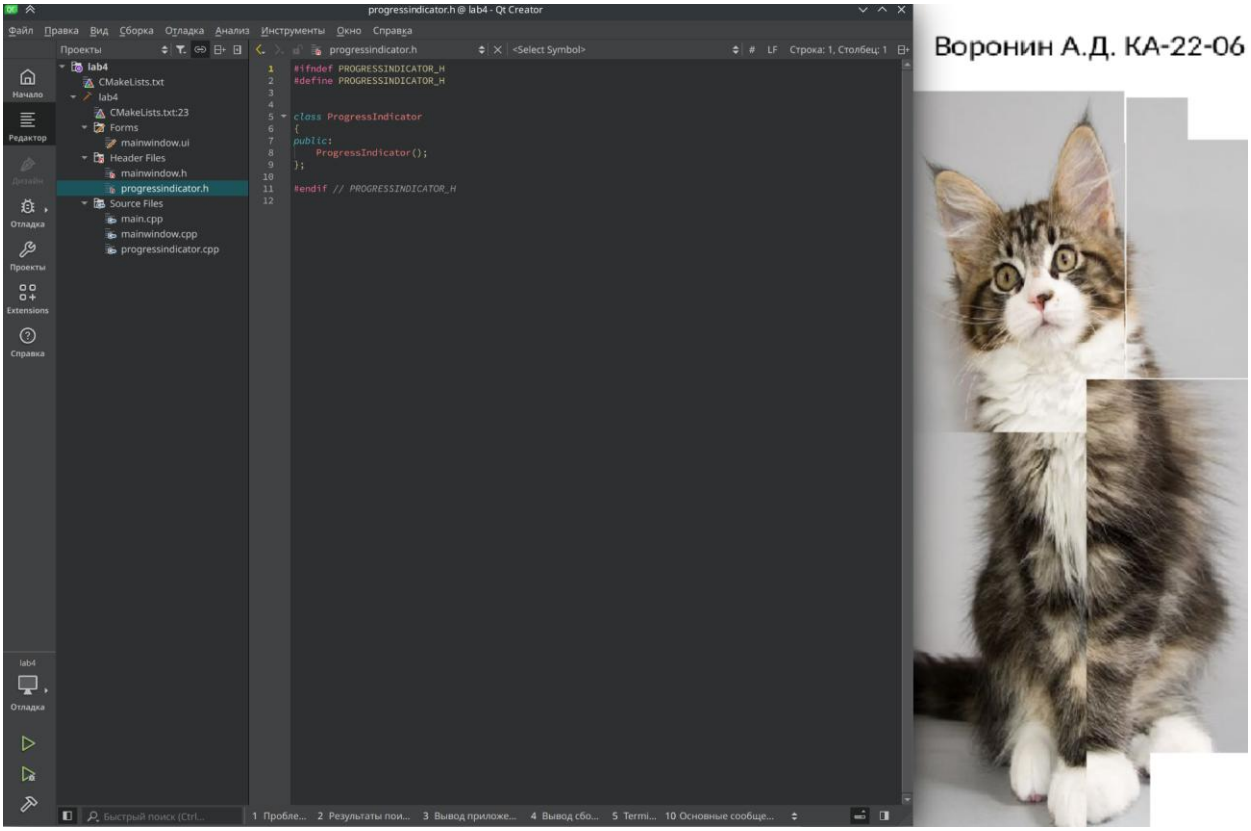

Рисунок 1 – созданный класс

Измените код в файле progressindicator.h. Наследуйте его от QWidget.

```
#ifndef PROGRESSINDICATOR_H
#define PROGRESSINDICATOR_H


#include <QWidget>
#include <QPainter>
#include <QWheelEvent>


class ProgressIndicator : public QWidget
{
```

```
    Q_OBJECT

public:
    explicit ProgressIndicator(QWidget *parent = nullptr);
};

#endif // PROGRESSINDICATOR_H
```

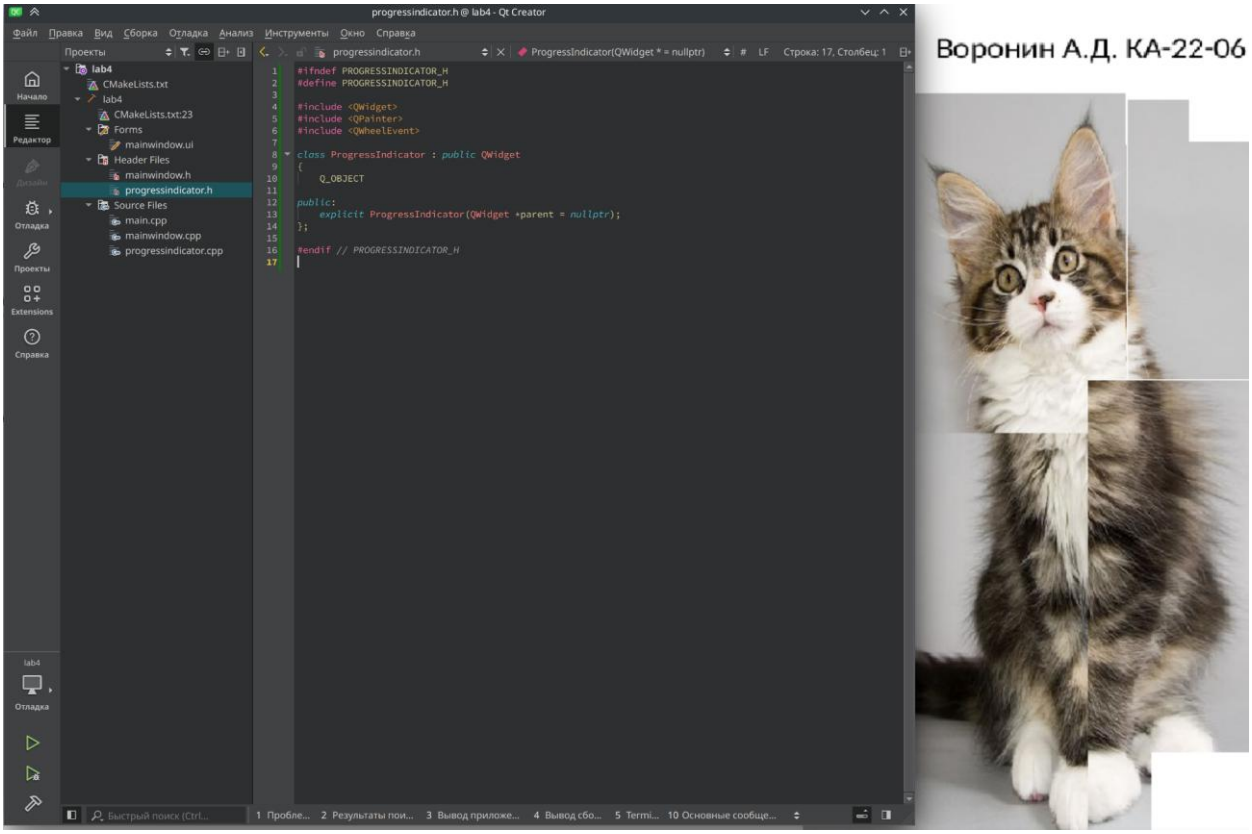

Рисунок 2 – Изменение ProgressIndicator.h

В классе ProgressIndicator переопределите метод paintEvent(QPaintEvent *event)для отрисовки индикатора выполнения. Используйте QPainter для рисования индикатора.

Добавьте в progressindicator.h следующее:

```
protected:
    void paintEvent(QPaintEvent *event) override;
```

Добавьте в progressindicator.cpp следующее:

```
void ProgressIndicator::paintEvent(QPaintEvent *event)
{
    Q_UNUSED(event);

    QPainter painter(this);
    painter.setRenderHint(QPainter::Antialiasing);

    int side = qMin(width(), height());
    QRectF outerRect(0, 0, side, side);
    outerRect.moveCenter(rect().center());

    painter.setPen(Qt::NoPen);
    painter.setBrush(Qt::gray);
    painter.drawEllipse(outerRect);

    painter.setBrush(Qt::blue);
    int spanAngle = static_cast<int>(360 * (static_cast<double>(m_progressValue - m_minimumValue) / (m_maximumValue - m_minimumValue)));
    painter.drawPie(outerRect, 90 * 16, -spanAngle * 16);

    painter.setPen(Qt::white);
    QFont font = painter.font();
    int fontSize = static_cast<int>(side * 0.15);
    font.setPointSize(fontSize);
    painter.setFont(font);

    QString progressText = QString("%1%").arg((static_cast<double>(m_progressValue - m_minimumValue) / (m_maximumValue - m_minimumValue)) * 100, 0, 'f', 1);

    QRectF textRect = outerRect;
    textRect.adjust(10, 10, -10, -10);
    painter.drawText(textRect, Qt::AlignCenter, progressText);
}
```

Добавьте в класс ProgressIndicator свойства, такие как progressValue (текущее значение прогресса), maximumValue (максимальное значение) и minimumValue (минимальное значение).

Измените файл progressindicator.h на следующее содержание:

```
#ifndef PROGRESSINDICATOR_H
#define PROGRESSINDICATOR_H

#include <QWidget>
#include <QPainter>
#include <QWheelEvent>

class ProgressIndicator : public QWidget
{
   Q_OBJECT

public:
   explicit ProgressIndicator(QWidget *parent = nullptr);

   void setProgressValue(int value);
   int progressValue() const;

   void setMaximumValue(int value);
   int maximumValue() const;

   void setMinimumValue(int value);
   int minimumValue() const;

protected:
   void paintEvent(QPaintEvent *event) override;

private:
   int m_progressValue;
   int m_maximumValue;
   int m_minimumValue;
};
#endif // PROGRESSINDICATOR_H
```

Реализуйте методы для работы со свойствами.

Добавьте в progressindicator.cpp следующее:

```
ProgressIndicator::ProgressIndicator(QWidget *parent)
    : QWidget(parent), m_progressValue(0), m_maximumValue(100),
m_minimumValue(0)
{
    setSizePolicy(QSizePolicy::Expanding, QSizePolicy::Expanding);
}

void ProgressIndicator::setProgressValue(int value)
{
    if (value < m_minimumValue)
        value = m_minimumValue;
    if (value > m_maximumValue)
        value = m_maximumValue;

    m_progressValue = value;
    update();
}

int ProgressIndicator::progressValue() const
{
    return m_progressValue;
}

void ProgressIndicator::setMaximumValue(int value)
{
    m_maximumValue = value;
    update();
}

int ProgressIndicator::maximumValue() const
{
    return m_maximumValue;
}

void ProgressIndicator::setMinimumValue(int value)
```

```
{
    m_minimumValue = value;
    update();
}

int ProgressIndicator::minimumValue() const
{
    return m_minimumValue;
}
```

Добавьте возможность изменять значение прогресса через колесо мыши.

Добавьте в progressindicator.h следующее:

```
protected:
    void wheelEvent(QWheelEvent *event) override;
```

Добавьте в progressindicator.cpp следующее:

```
void ProgressIndicator::wheelEvent(QWheelEvent *event)
{
    int delta = event->angleDelta().y() / 120;
    setProgressValue(m_progressValue + delta);
}
```

ЧАСТЬ 2 - Интеграция кастомного виджета в приложение

Используйте QMainWindow или QWidget в качестве контейнера для вашего кастомного виджета. Создайте новые файлы mainwindow.h и mainwindow.cpp для реализации главного окна.

Добавьте в mainwindow.h следующее:

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include <QSlider>
#include <QVBoxLayout>
#include "progressindicator.h"

class MainWindow : public QMainWindow
{
   Q_OBJECT

public:
   MainWindow(QWidget *parent = nullptr);
   ~MainWindow();

private:
   ProgressIndicator *progressIndicator;
   QSlider *slider;
};

#endif // MAINWINDOW_H
```

Реализуйте mainwindow.cpp:

```
#include "mainwindow.h"
MainWindow::MainWindow(QWidget *parent)
   : QMainWindow(parent)
{
   progressIndicator = new ProgressIndicator(this);
```

```
    slider = new QSlider(Qt::Horizontal, this);
    slider->setRange(0, 100);
    slider->setValue(0);

    QVBoxLayout *layout = new QVBoxLayout();
    layout->addWidget(progressIndicator);
    layout->addWidget(slider);

    QWidget *centralWidget = new QWidget(this);
    centralWidget->setLayout(layout);
    setCentralWidget(centralWidget);

    connect(slider, &QSlider::valueChanged, progressIndicator,
&ProgressIndicator::setProgressValue);
}

MainWindow::~MainWindow()
{
}
```

Создадим файл main.cpp для запуска приложения и тестирования виджета.

```
#include <QApplication>
#include "mainwindow.h"

int main(int argc, char *argv[])
{
    QApplication app(argc, argv);

    MainWindow mainWindow;
    mainWindow.show();

    return app.exec();
}
```

Реализуйте изменение значения индикатора выполнения через предоставленный интерфейс и убедитесь, что отрисовка индикатора корректно обновляется.

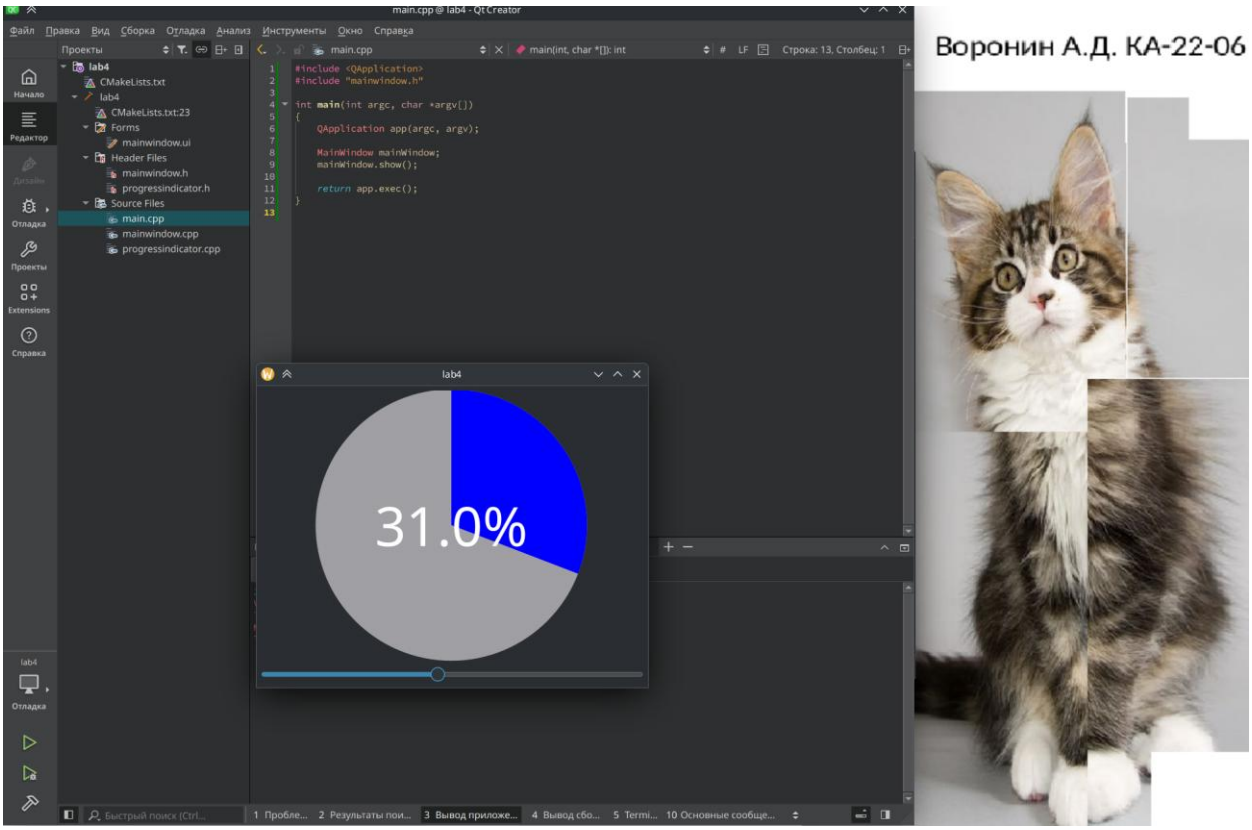

Рисунок 3 – Тестирование

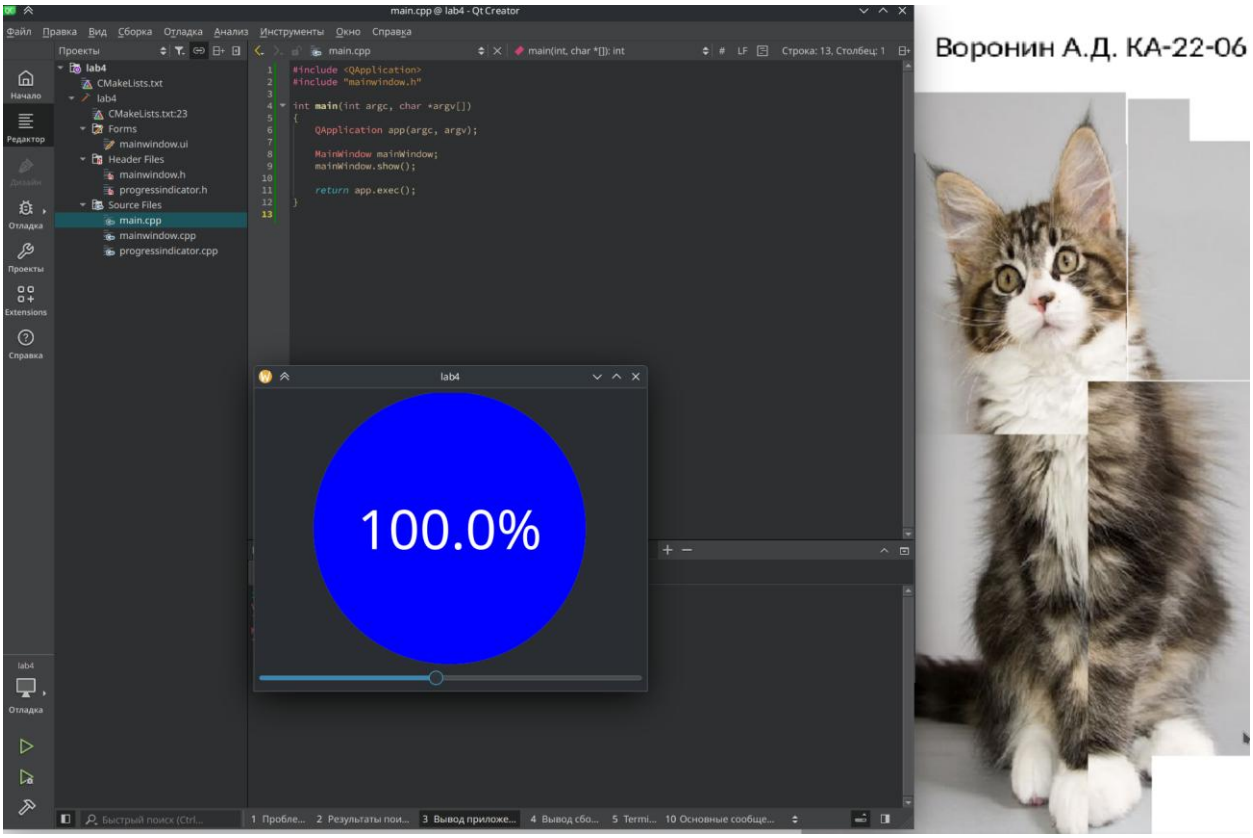

Рисунок 4 – Тестирование

Самостоятельная работа

- Создать класс SpiralProgressIndicator, наследующийся от QWidget.
- Реализовать отрисовку спирального индикатора прогресса с использованием QPainter.
- Добавить свойства для управления минимальным, максимальным и текущим значениями прогресса.
- Реализовать анимацию изменения значения прогресса.
- Добавить изменение значения с помощью колесика мыши и слайдера.
- Отображать текущее значение прогресса в виде текста в центре спирали.

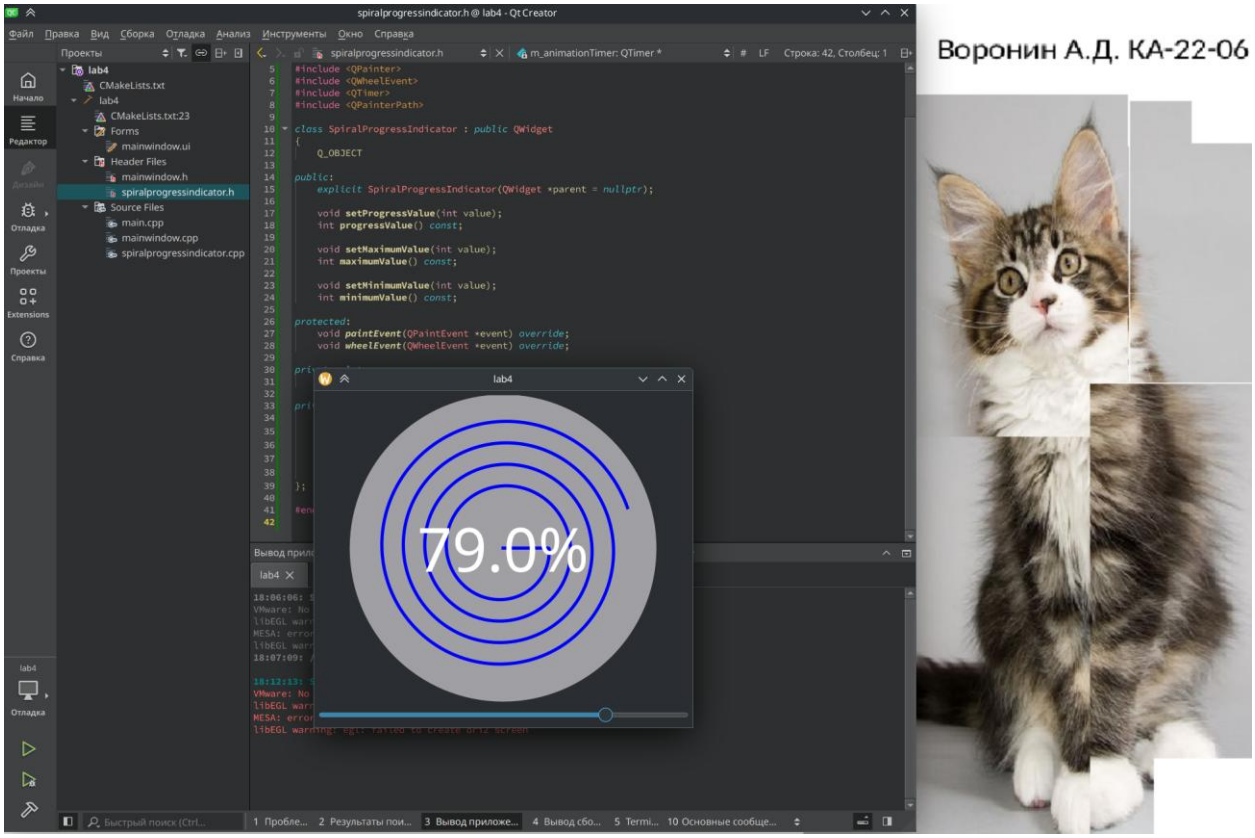

Рисунок 5 – Результат выполнения самостоятельной работы

ЗАКЛЮЧЕНИЕ

В заключение, подводя итог проделанной работе, все поставленные в начале цели и задания данной лабораторной работы были достигнуты в полном объеме.

Контрольные вопросы

1. Что такое кастомный виджет в Qt?

Кастомный виджет в Qt — это пользовательский виджет, который расширяет или улучшает функциональность встроенных виджетов Qt. Такие виджеты создаются путём подкласса существующих классов виджетов Qt или путём объединения нескольких существующих виджетов для создания нового.

2. Какие преимущества предоставляет использование QPainter для отрисовки виджетов?

Использование QPainter для отрисовки виджетов даёт полную гибкость в создании визуальных элементов: можно рисовать любые формы (линии, кривые, текст, изображения), использовать сглаживание (Antialiasing) для красивых границ, применять градиенты и сложные стили оформления, а также оптимизировать производительность за счёт прямого контроля над процессом рисования, что позволяет создавать уникальные и высокоэффективные пользовательские интерфейсы, выходящие за рамки стандартных виджетов Qt.

3. Как реализовать изменение значения индикатора через колесико мыши?

Изменение значения индикатора через колесико мыши реализуется переопределением метода wheelEvent, где считывается направление прокрутки с помощью event->angleDelta().y(), делится на 120 для получения количества «шагов» прокрутки, и затем вызывается setProgressValue, увеличивая или уменьшая текущее значение.

```
void ProgressIndicator::wheelEvent(QWheelEvent *event)
{
   int delta = event->angleDelta().y() / 120;
   setProgressValue(m_progressValue + delta);
}
```

4. Как можно оптимизировать производительность кастомного виджета?

Для оптимизации производительности кастомного виджета можно минимизировать количество и область перерисовок, вызывать update() только при реальных изменениях, использовать кэширование часто рисуемых элементов через QPixmap, избегать тяжёлых вычислений внутри paintEvent, заранее готовить объекты типа QPen, QBrush и QFont, аккуратно применять антиалиасинг только там, где это нужно, и правильно настраивать частоту обновления анимаций, чтобы избежать лишней нагрузки на систему.