ЛАБОРАТОРНАЯ РАБОТА №2

«ИССЛЕДОВАНИЕ СПОСОБОВ ИСПОЛЬЗОВАНИЯ СИГНАЛОВ

И СЛОТОВ В QT-ПРИЛОЖЕНИЯХ»

1. Цель

Исследование технологии подготовки и выполнения программ в интегрированной среде Qt Creator. Получение базовых навыков работы с фреймворком Qt. Исследовать принцип работы механизма сигналов и слотов фреймворка Qt. Приобрести практические навыки применения сигналов и слотов при разработке Qt-приложений.

2. Постановка задачи

2.1. Изучить принципы работы механизма сигналов и слотов в Qt, способы соединения сигналов и слотов (выполняется в ходе домашней подготовки к лабораторной работе).

2.2. Создать проект Qt Gui Application.

2.3. Создать класс-наследник класса QLabel, добавив собственный сигнал, который будет посылаться, когда значение QLabel равно числу, большему десяти.

2.4. Разместить на форме виджеты QLineEdit, QPushButton, два виджета QPlainTextEdit и виджет созданного на шаге 3.3 наследника QLabel.

2.5. Обеспечить изменение названия заголовка окна приложения на значение, введенное в QLineEdit при нажатии на кнопку.

2.6. Создать собственный слот для MainWindow, который будет копировать текст из первого QPlainTextEdit во второй, заменяя все символы ‘a’ на ‘\*’.

2.7. Подключить слот, созданный на предыдущем этапе к textChanged сигналу первого QPlainTextEdit, таким образом обеспечив автоматическое копирование.

2.8. Создать собственный слот для MainWindow, который будет выводить количество ‘\*’ во втором QPlainTextEdit в QLabel.

2.9. Подключить слот, созданный на предыдущем этапе к textChanged сигналу второго QPlainTextEdit, таким образом обеспечив автоматическое подсчет количества символов ‘\*’.

2.10. Подключить слот setDisabled первого QPlainTextEdit к сигналу, созданному на шаге 3.3, тем самым обеспечив запрет на дальнейший ввод (setDisabled cлот) при вводе более десяти символов ‘a’.

2.11. Выполнить экспериментальное исследование полученного приложения, выполняя ввод тестовых последовательностей с различным количеством символов ‘a’ и различным их положением во вводимой строке: в начале, в середине и в конце.

3. Ход работы

Был создан проект – приложение Qt Widgets. На рисунке 1 изображено его создание.

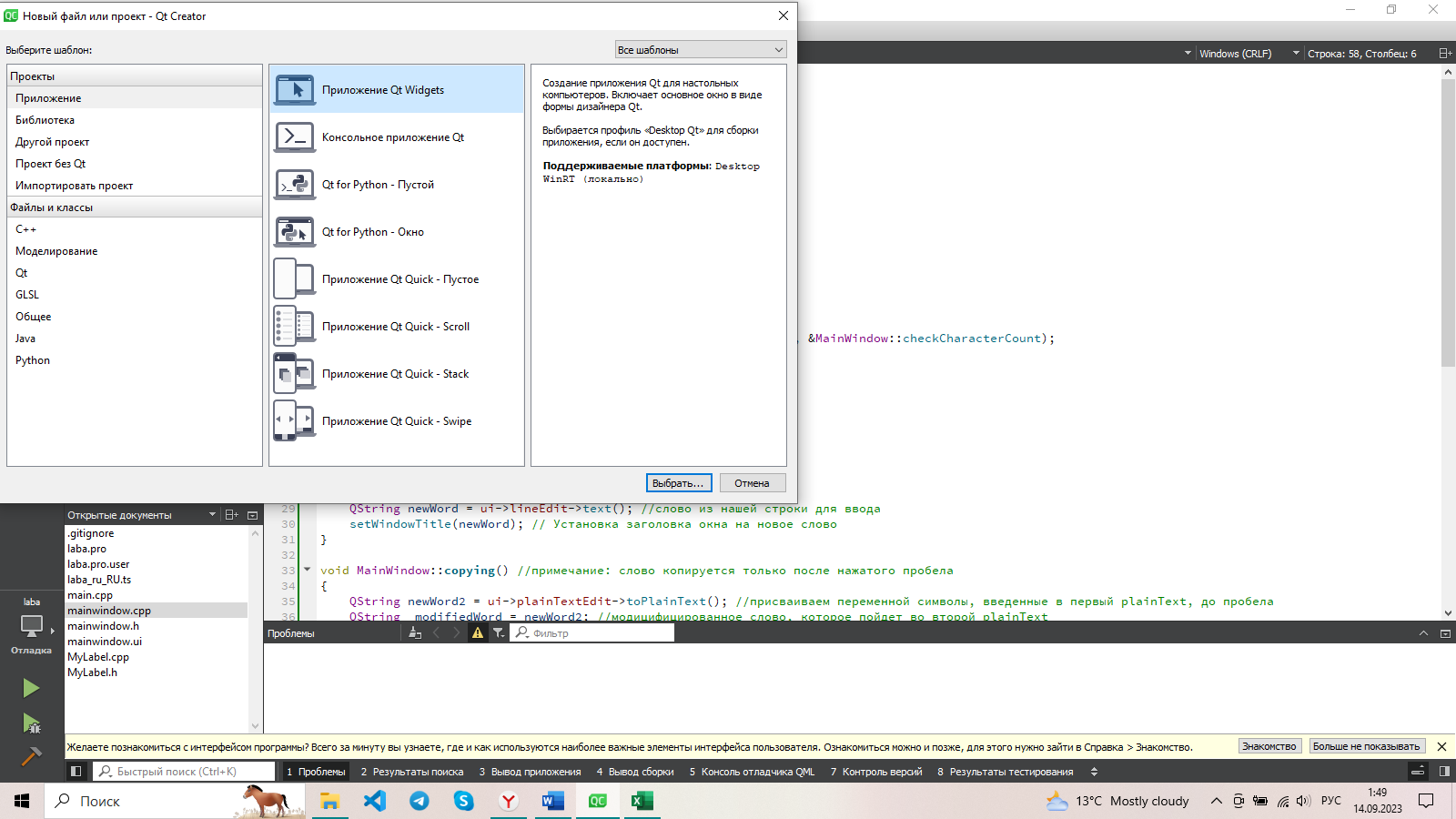


Рисунок 1 – Создание нового проекта

Требовалось создать класс-наследник класса QLabel, добавить собственный сигнал, который будет посылаться, когда значение QLabel равно числу, большему десяти. Для этого был создан класс MyLabel. Код представлен в листинге 1.

Полный код проекта находится в приложении А.

Листинг 1 – Класс MyLabel

MyLabel.h:

#ifndef MYLABEL\_H

#define MYLABEL\_H

#include <QLabel>

class **MyLabel** : public QLabel //наследуем класс

{

Q\_OBJECT

private:

int value = 0; //текущее значение баллов

const int banLimit = 10; //максимальное количество штрафов

public:

**MyLabel**(QWidget \*parent=0);

public slots:

void **incValue**();

signals:

void **banUser**();

};

#endif // MYLABEL\_H

MyLabel.cpp:

#include "MyLabel.h"

MyLabel::**MyLabel**(QWidget \*parent)

: QLabel(parent)

{

/\*\*/

}

void MyLabel::**incValue**()

{

if (++value > banLimit)

{

emit banUser();

}

this->setText(QString::number(value));

}

Далее требовалось разместить на форме виджеты QLineEdit, QPushButton, два виджета QPlainTextEdit и виджет созданного на шаге 3.3 наследника QLabel. На рисунке 2 изображена форма после изменений. В правом углу можно видеть названия каждого виджета: label – надпись «Мое приложение», label\_2 – надпись «Штраф», label\_3 – пустая надпись возле надписи «Штраф», lineEdit – строка ввода, plainTextEdit – левое поле для ввода, plainTextEdit\_2 – правое поле для ввода, pushButton – кнопка «Добавить».

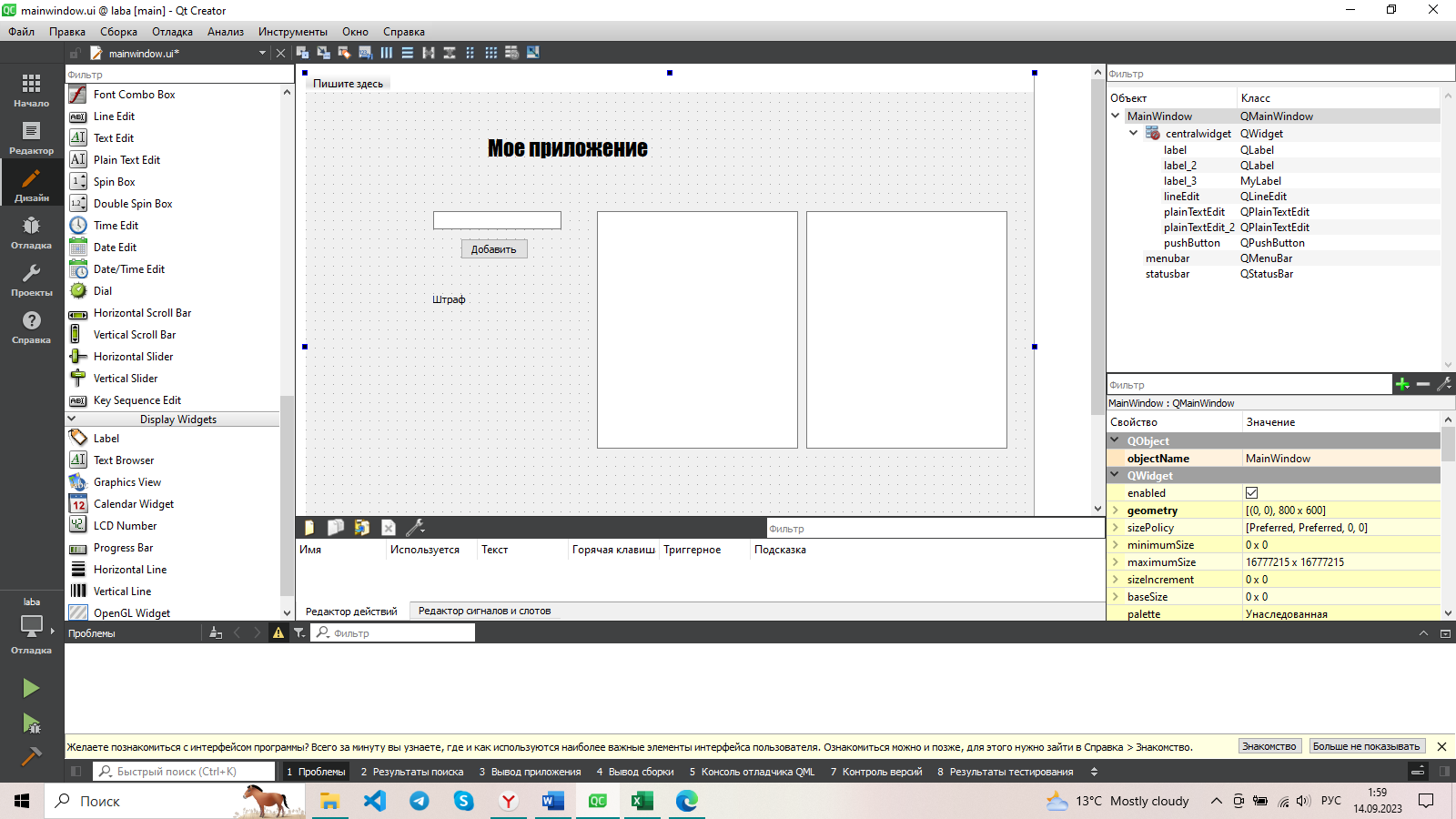


Рисунок 2 – Готовая форма

Необходимо было обеспечить изменение названия заголовка окна приложения на значение, введенное в QLineEdit при нажатии на кнопку. Для этого в файле mainwindow.h был объявлен приватный слот addWord(), релизован в файле mainwindow.cpp (код реализации представлен в листинге 2), а также соединен сигнал нажатия клавиши «Добавить» с новым слотом addWord.

Листинг 2 – Реализация слота addWord()

MainWindow::**MainWindow**(QWidget \*parent)

: QmainWindow(parent)

, ui(new Ui::MainWindow)

{

/\* \*/

connect(ui->pushButton, SIGNAL(clicked()), this,

SLOT(addWord()));

/\* \*/

}

void MainWindow::**addWord**()

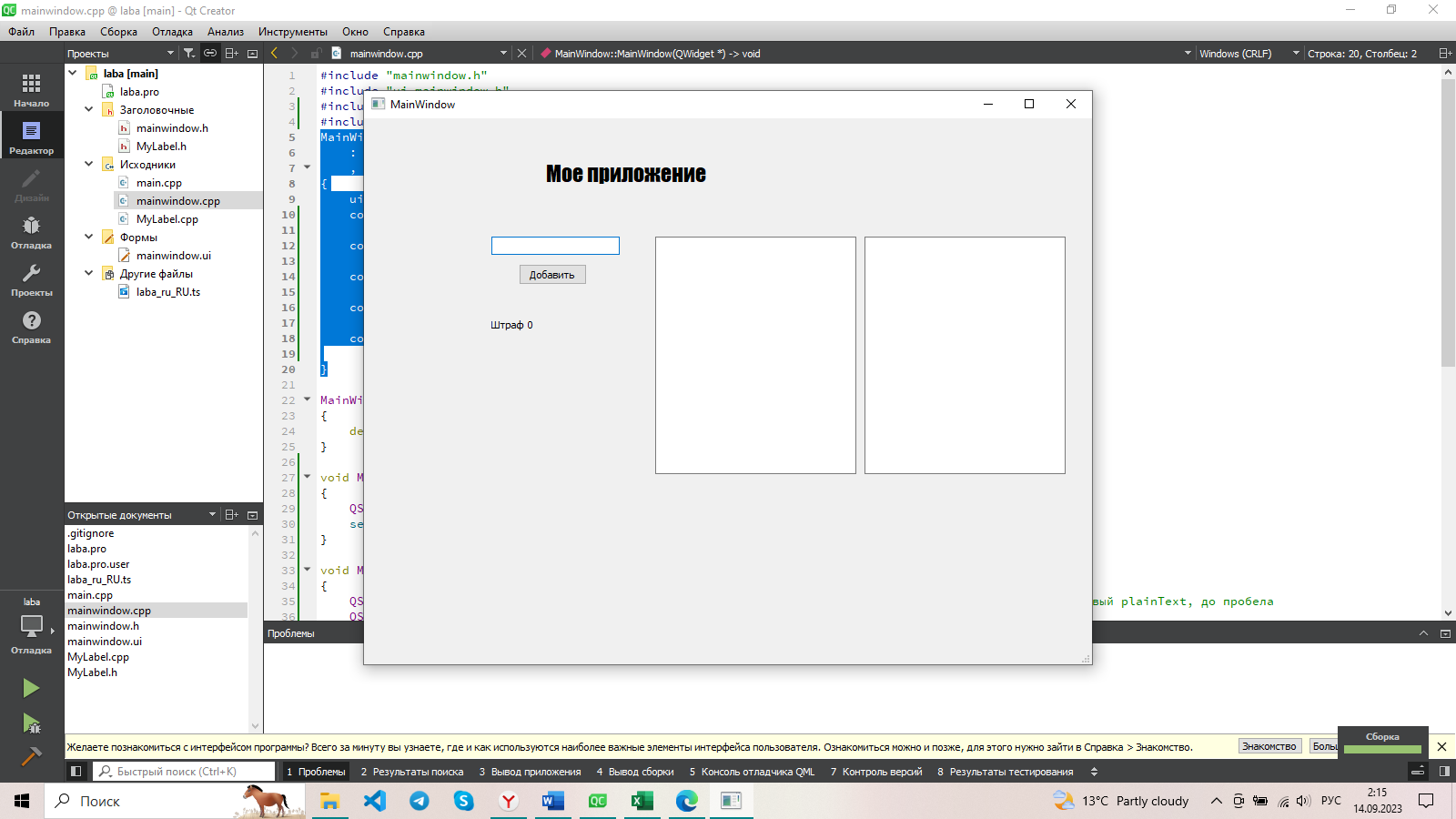
{

Qstring newWord = ui->lineEdit->text(); //слово для ввода

setWindowTitle(newWord); // Установка заголовка окна на новое слово

}

На рисунке 3 показана работа данной функции.



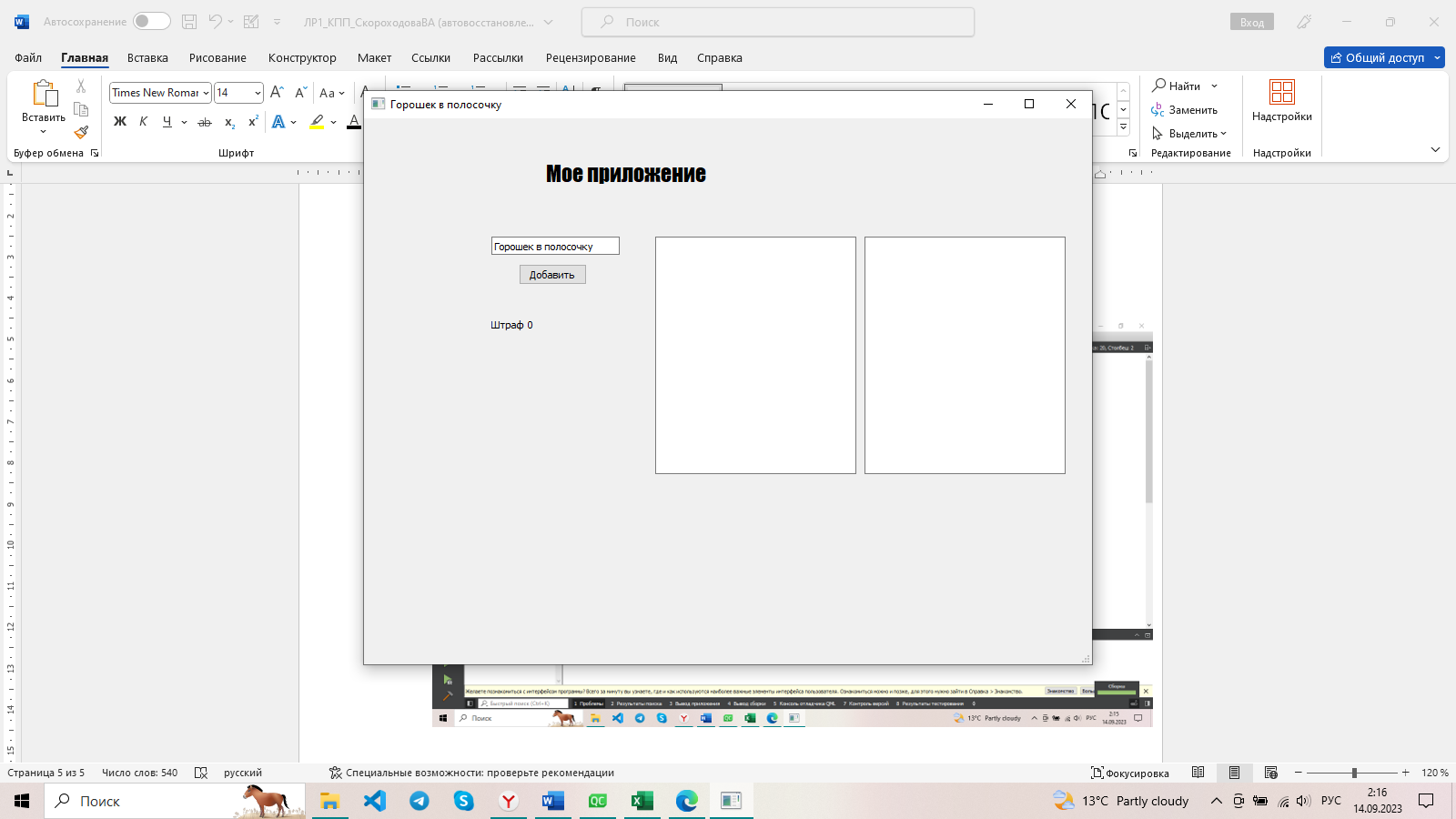


Рисунок 3 – До и после работы функции

Далее требовалось создать собственный слот для MainWindow, который будет копировать текст из первого QPlainTextEdit во второй, заменяя все символы ‘a’ на ‘\*’. Подключить слот, созданный на предыдущем этапе к textChanged сигналу первого QPlainTextEdit, таким образом обеспечив автоматическое копирование.

Листинг 3 – Реализация слота copying()

MainWindow::**MainWindow**(QWidget \*parent)

: QMainWindow(parent)

, ui(new Ui::MainWindow)

{

/\* \*/

connect(ui->plainTextEdit, SIGNAL(textChanged()), this,

SLOT(copying()));

/\* \*/

}

void MainWindow::**copying**() //примечание: слово копируется только после нажатого пробела

{

QString newWord2 = ui->plainTextEdit->toPlainText(); //присваиваем переменной символы, введенные в первый plainText, до пробела

QString modifiedWord = newWord2; //модицифицированное слово, которое пойдет во второй plainText

QString lastchar = newWord2.right(1).at(0); //присваиваем lastchar'y последний символ вводимой строки, ищем пробел

modifiedWord.replace("a", "\*"); //замена а на \*

if(lastchar == " ") //если нашли пробел, т.е. конец слова, то пишем модифицированное слово во второй plainText

{

ui->plainTextEdit\_2->setPlainText(modifiedWord);

}

}

На рисунке 4 изображена работа приведенного выше кода.

Далее было необходимо создать собственный слот для MainWindow, который будет выводить количество ‘\*’ во втором QPlainTextEdit в QLabel. Подключить слот, созданный на предыдущем этапе к textChanged сигналу второго QPlainTextEdit, таким образом обеспечив автоматическое подсчет количества символов ‘\*’. На рисунке 4 показана работа функции.

В листинге 4 представлена реализация слота и подключение его к сигналу.

Листинг 4 – Реализация слота и подключение

MainWindow::**MainWindow**(QWidget \*parent)

: QMainWindow(parent)

, ui(new Ui::MainWindow)

{

/\* \*/

connect(ui->plainTextEdit\_2, SIGNAL(textChanged()), this,

SLOT(counting()));

/\* \*/

}

void MainWindow::**counting**(){

QString text = ui->plainTextEdit\_2->toPlainText();

int starCount = text.count('\*');

ui->label\_3->setText(QString("Count: %1").arg(starCount));

}

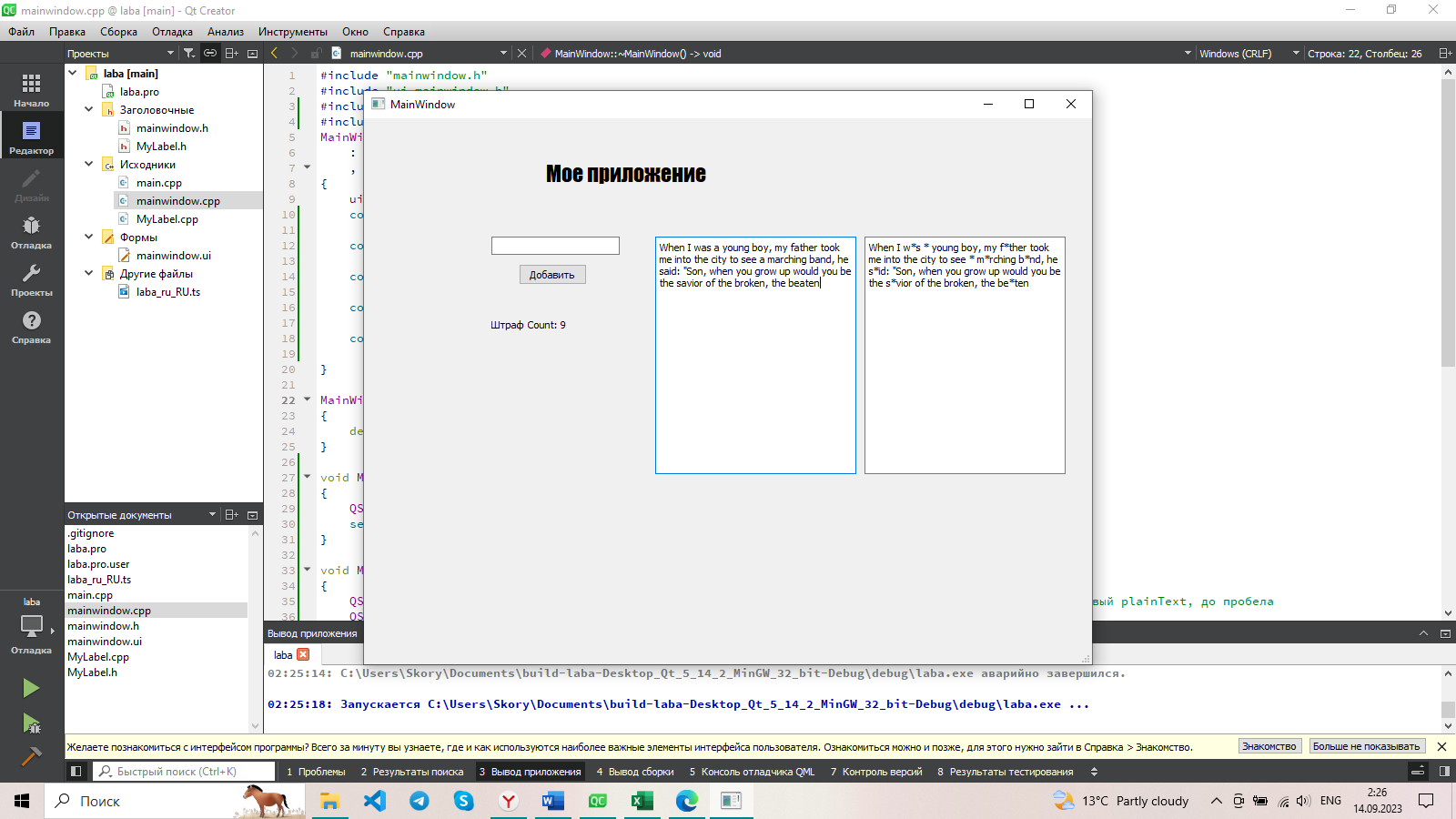


Рисунок 4 – Работа слотов copying() и counting()

Наконец, требовалось подключить слот setDisabled первого QPlainTextEdit к сигналу, созданному на шаге 3.3, тем самым обеспечив запрет на дальнейший ввод (setDisabled cлот) при вводе более десяти символов ‘a’. В листинге 5 представлен код подключения

Листинг 5 – Реализация слота и подключение

MainWindow::**MainWindow**(QWidget \*parent)

: QMainWindow(parent)

, ui(new Ui::MainWindow)

{

/\* \*/

connect(ui->plainTextEdit, SIGNAL(textChanged()), this,

SLOT(checkCharacterCount()));

/\* \*/

}

void MainWindow::**checkCharacterCount**()

{

QString text = ui->plainTextEdit->toPlainText();

int starCount = text.count('a');

if (starCount > 10)

{

ui->plainTextEdit->setDisabled(true);

}

else

{

ui->plainTextEdit->setDisabled(false);

}

}

На рисунке 5 изображена работа выше приведенного кода.

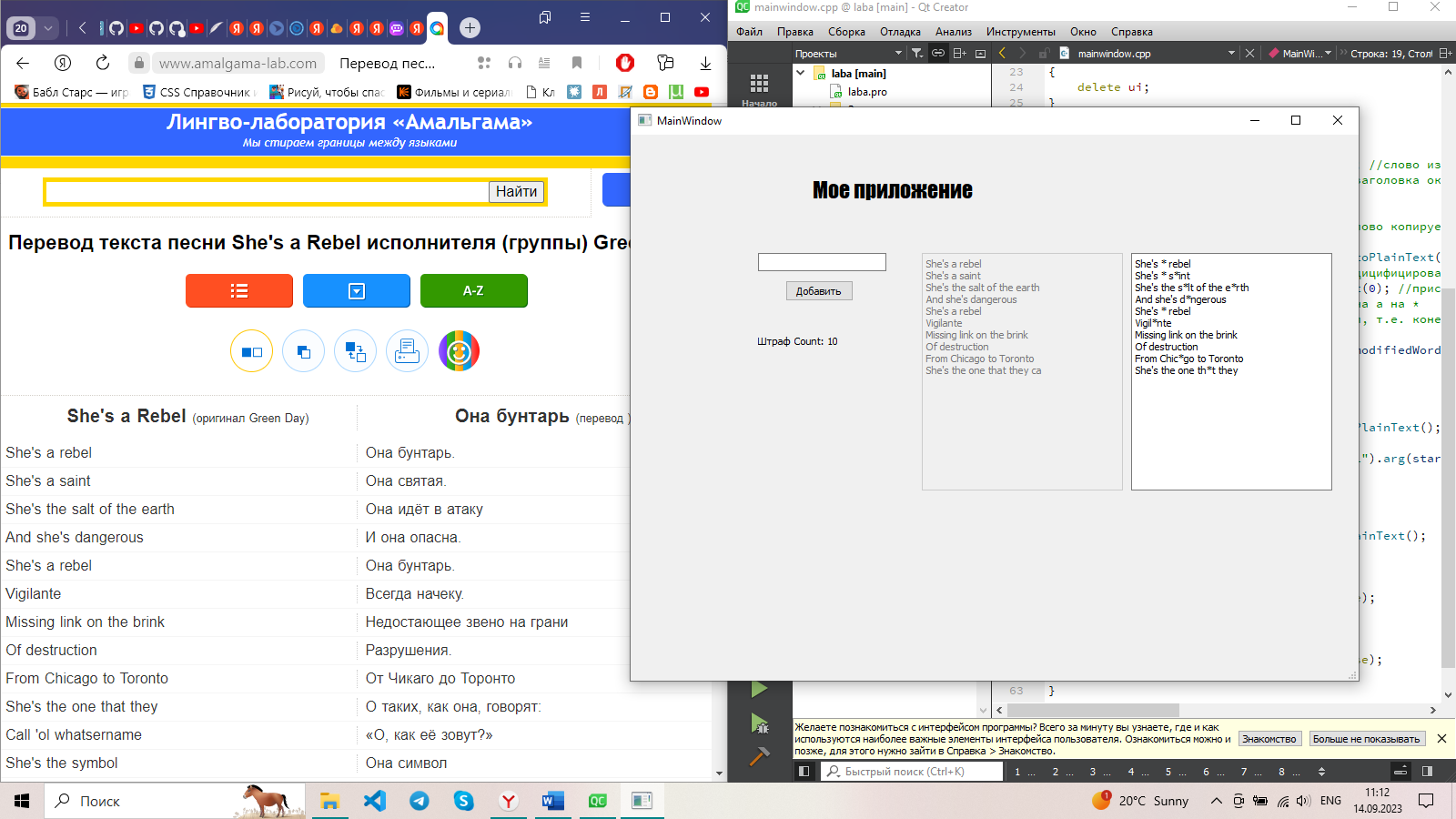


Рисунок 5 – Работа слота checkCharacterCount()

Как можно видеть из тестовых последовательностей, все функции работают корректно.

4. Выводы

В ходе лабораторной работы было проведено исследование технологии подготовки и выполнения программ в интегрированной среде Qt Creator, были получены базовые навыки работы с фреймворком Qt, исследован принцип работы механизма сигналов и слотов фреймворка Qt и приобретены практические навыки применения сигналов и слотов при разработке Qt-приложения.

5. Контрольные вопросы

5.1. Какие существуют подходы к организации взаимодействия элементов графического интерфейса?

* Прямое связывание (Direct Connection): Элементы графического интерфейса взаимодействуют напрямую друг с другом через вызовы функций или методов. Например, один виджет может вызывать метод другого виджета для передачи данных или выполнения действия.
* Использование сигналов и слотов (Signals and Slots): Это механизм, предоставляемый Qt, который позволяет объектам отправлять сигналы и принимать их с помощью слотов. Сигналы и слоты могут быть соединены между собой, чтобы установить связь и реагировать на события или передавать данные.
* Шаблон Наблюдатель (Observer Pattern): Этот шаблон проектирования позволяет объектам подписываться и получать уведомления об изменениях в других объектах. В Qt этот шаблон может быть реализован с помощью сигналов и слотов или с использованием других механизмов, таких как события.

5.2. Каковы плюсы и минусы каждого подхода к организации взаимодействия элементов графического интерфейса?

* Прямое связывание:

Плюсы: Простота реализации, прямой доступ к функциональности другого объекта.

Минусы: Жесткая связь между объектами, что может привести к сложностям при изменении кода и его поддержке.

* Использование сигналов и слотов:

Плюсы: Гибкость и расширяемость, слабая связь между объектами, возможность подключения нескольких слотов к одному сигналу.

Минусы: Небольшое снижение производительности из-за использования механизма сигналов и слотов.

* Шаблон Наблюдатель:

Плюсы: Расширяемость, возможность подписки на события от нескольких объектов, слабая связь между объектами.

Минусы: Больше кода для реализации, возможность возникновения проблем с управлением подписками и уведомлениями.

5.3. В чем состоит особенность описания класса, использующего механизм сигналов и слотов?

объявление класса должно содержать специальный макрос Q\_OBJECT на следующей строке после ключевого слова class: + signals и slots

5.4. Что происходит при компиляции класса, использующего механизм сигналов и слотов?

Перед выполнением компиляции Meta Object Compiler (MOC) анализирует Q\_Object классы и автоматически внедряет в них всю необходимую информацию (используются отдельные файлы, разработчик может их игнорировать)

5.5. Каким образом создаются собственные сигналы и слоты?

Через класс Q\_Object и ключевые слова slots и signals.

5.6. Каким образом вызвать созданный сигнал?

void MyClass ::sendMySignal() { **emit** mySignal(); }

5.7. Как соединить сигналы и слоты?

Для соединения сигналов и слотов можно использовать статический метод connect, определенный в классе QObject. В общем виде соединение выглядит следующим образом:

**connect(sender, SIGNAL(signal), receiver, SLOT(slot));**

sender и receiver – это указатели на Qobject, signal и slot – сигнатуры сигнала и слота.

5.8. Возможно ли соединение двух сигналов?

**connect(sender1, SIGNAL(function1()),receiver, SIGNAL(function2()));**

В этом случае при возникновении первого сигнала, автоматически генерируются все связанные сигналы.

5.9. Возможно ли соединение сигнала с несколькими слотами?

Да, в Qt возможно соединение сигнала с несколькими слотами. Вы можете использовать несколько вызовов функции QObject::connect(), чтобы соединить один и тот же сигнал с разными слотами.

**connect(slider, SIGNAL(valueChanged(int)),spinBox, SLOT(setValue(int)));**

**connect(slider, SIGNAL(valueChanged(int)),this, SLOT(updateStatusBarIndicator(int)));**

5.10. Как можно разъединить сигнал и слот?

**disconnect(sender0, SIGNAL(overflow()),receiver1, SLOT(handleMathError()));**

5.11. Каким образом возможно осуществить передачу параметра из сигнала в слот?

При соединении сигналов и слотов можно передавать параметры от сигнала к слоту, если существует соответствующая пара сигнал/слот:

- без параметров:

QPushButton \*btn\_quit = new QPushButton("Quit",wgt); ... QObject::connect(btn\_quit, SIGNAL(clicked()), qApp, SLOT(quit()));

В этом примере сигнал clicked() от кнопки btn\_quit соеденяется со слотом quit() приложения (qApp - глобальный указатель на экземпляр приложения).

- с параметрами. При соединении сигналов и слотов с передачей параметров следует помнить замечание указанное в описании к QObject::connect(). То есть имена аргументов должны быть опущены.

QObject::connect(scrollBar, SIGNAL(valueChanged(int)), label, SLOT(setNum(int)));

ПРИЛОЖЕНИЕ А – ПОЛНЫЙ КОД ПРОЕКТА

Листинг 6 – Заголовочный файл mainwindow.h

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include <QMainWindow>

#include <QMessageBox>

#include <MyLabel.h>

QT\_BEGIN\_NAMESPACE

namespace **Ui** { class **MainWindow**; }

QT\_END\_NAMESPACE

class **MainWindow** : public QMainWindow

{

Q\_OBJECT

public:

explicit **MainWindow**(QWidget \*parent = nullptr);

~***MainWindow***();

private:

Ui::MainWindow \*ui;

private slots:

void **addWord**();

void **copying**();

void **counting**();

void **checkCharacterCount**();

};

#endif // MAINWINDOW\_H

Листинг 7 – Заголовочный файл MyLabel.h

#ifndef MYLABEL\_H

#define MYLABEL\_H

#include <QLabel>

class **MyLabel** : public QLabel //наследуем класс

{

Q\_OBJECT

private:

int value = 0; //текущее значение баллов

const int banLimit = 10; //максимальное количество штрафов

public:

**MyLabel**(QWidget \*parent=0);

public slots:

void **incValue**();

signals:

void **banUser**();

};

#endif // MYLABEL\_H

Листинг 8 – Исходный файл main.cpp

#include "mainwindow.h"

#include <QMainWindow>

#include <QLineEdit>

#include <QPushButton>

#include <QVBoxLayout>

#include <QApplication>

int main(int argc, char \*argv[])

{

QApplication a(*argc*, argv);

MainWindow window;

window.show();

return a.exec();

}

Листинг 9 – Исходный файл mainwindow.cpp

#include "mainwindow.h"

#include "ui\_mainwindow.h"

#include <QLineEdit>

#include <QPushButton>

MainWindow::**MainWindow**(QWidget \*parent)

: QMainWindow(parent)

, ui(new Ui::MainWindow)

{

ui->setupUi(this);

connect(ui->pushButton, SIGNAL(clicked()), this,

SLOT(addWord()));

connect(ui->label\_3, SIGNAL(banUser()), ui->pushButton,

SLOT(hide()));

connect(ui->plainTextEdit, SIGNAL(textChanged()), this,

SLOT(copying()));

connect(ui->plainTextEdit\_2, SIGNAL(textChanged()), this,

SLOT(counting()));

connect(ui->plainTextEdit, SIGNAL(textChanged()), this, SLOT(checkCharacterCount()));

}

MainWindow::~***MainWindow***()

{

delete ui;

}

void MainWindow::**addWord**()

{

QString newWord = ui->lineEdit->text(); //слово из нашей строки для ввода

setWindowTitle(newWord); // Установка заголовка окна на новое слово

}

void MainWindow::**copying**() //примечание: слово копируется только после нажатого пробела

{

QString newWord2 = ui->plainTextEdit->toPlainText(); //присваиваем переменной символы, введенные в первый plainText, до пробела

QString modifiedWord = newWord2; //модицифицированное слово, которое пойдет во второй plainText

QString lastchar = newWord2.right(1).at(0); //присваиваем lastchar'y последний символ вводимой строки, ищем пробел

modifiedWord.replace("a", "\*"); //замена а на \*

if(lastchar == " ") //если нашли пробел, т.е. конец слова, то пишем модифицированное слово во второй plainText

{

ui->plainTextEdit\_2->setPlainText(modifiedWord);

}

}

void MainWindow::**counting**(){

QString text = ui->plainTextEdit\_2->toPlainText();

int starCount = text.count('\*');

ui->label\_3->setText(QString("Count: %1").arg(starCount));

}

void MainWindow::**checkCharacterCount**()

{

QString text = ui->plainTextEdit->toPlainText();

int starCount = text.count('a');

if (starCount > 10)

{

ui->plainTextEdit->setDisabled(true);

}

else

{

ui->plainTextEdit->setDisabled(false);

}

}

Листинг 10 – Исходный файл MyLabel.cpp

#include "MyLabel.h"

MyLabel::**MyLabel**(QWidget \*parent)

: QLabel(parent)

{

/\*\*/

}

void MyLabel::**incValue**()

{

if (++value > banLimit)

{

emit banUser();

}

this->setText(QString::number(value));

}