

К.В. ВАЛЬШТЕЙН, А.А. БАРМИНА, И.Н. МАГОМЕДОВ

# ВИЗУАЛЬНОЕ ПРОГРАММИРОВАНИЕ

## Практикум



Санкт-Петербург  
Издательство БГТУ «ВОЕНМЕХ» им. Д.Ф. Устинова  
2023

УДК

**Вальштейн К. В.**

Визуальное программирование: практикум/ К.В. Вальштейн, А.А. Бармина, И.Н. Магомедов; Балт. гос. техн. ун-т. – СПб., Изд-во БГТУ «ВОЕНМЕХ» им. Д.Ф. Устинова, 2023. – 138 с.

Целью учебного пособия является формирование у студентов практических навыков разработки приложений с использованием методов визуального и объектно-ориентированного подхода к программированию, а также использование специализированных инструментов разработки, таких как фреймворк Qt или технология WPF.

Содержит 5 практических работ, выполняемых на компьютере и расположенные в порядке изучения лекционного материала. Каждая работа содержит порядок выполнения, варианты индивидуальных заданий и контрольные вопросы. Также приведены теоретические сведения, требуемые для выполнения работы и особенности использования различных средств визуального программирования для решения поставленной задачи.

Предназначено для студентов по направлениям 09.03.01, 09.03.02, 09.03.04 (очная и заочная формы обучения), для студентов, чьей сферой деятельности также является программирование

Рецензент: канд. техн. наук, доц., И.К. Ракова

*Утверждено  
редакционно-издательским  
советом университета*

© Изд-во БГТУ «ВОЕНМЕХ»

им. Д.Ф. Устинова, 2023

© Авторы 2023

## **ПРЕДИСЛОВИЕ**

На сегодняшний день важным подходом к разработке программного обеспечения является использование объектно-ориентированного программирования. Вместе с методами визуального программирования это позволяет упростить разработку программного обеспечения.

Предлагаемые для выполнения практические работы знакомят с основными особенностями, механизмами работы и техниками, которые необходимы для написания любого приложения с использованием данных подходов к разработке. В ходе работ на выбор студента предполагается использование либо фреймворка Qt в сочетании с поддерживаемым им языком программирования, либо языка программирования C# вместе с технологиями WPF и Windows Forms. В пособии приведена постановка заданий для обоих случаев.

Знания, полученные студентами в результате выполнения заданий, позволит студентам применять полученные знания при разработке программных средств, как для образовательных целей, так и в профессиональной деятельности.

После выполнения работы предполагается формирование отчета. Содержимое отчета: цель работы, текст задания согласно варианту, демонстрация результатов работы (изображениями или набором тестов), исходные тексты программ (при объёме текста более двух печатных листов, допускается использовать в тексте отчете только наиболее важные фрагменты).

После оформления отчёта, студент может приступить к защите практической работы. В процессе защиты студенты отвечает на некоторые вопросы из списка контрольных вопросов. В случае неполного ответа преподаватель вправе задать дополнительные вопросы, не входящие в список контрольных вопросов, для того чтобы удостовериться в освоении знаний и получении навыков в процессе выполнения практической работы.

# ПРАКТИЧЕСКАЯ РАБОТА 1

## «Создание пользовательского интерфейса приложения с использованием инструментов визуального программирования»

### **Теоретические сведения**

В рамках данного курса, при выполнении практических работ возможно использовать один из двух подходов:

1. Фреймворк Qt в сочетании с любым поддерживаемым им языком программирования, при условии выполнения требований задания;
2. Язык программирования C# в сочетании с технологиями WPF или Windows Forms в зависимости от задания.

В качестве теоретических сведений в первую очередь будет рассмотрено использование в качестве языка программирования C++11 совместно с фреймворком Qt5, а также использование языка программирования C#, так как обзор всех возможных технологий визуального программирования излишне усложнит изложение теоретического материала. Основные моменты, приведённые в теоретических сведениях, возможно использовать в сочетании с любым, поддерживающим концепцию визуального программирования инструментом.

При выполнении работ, помимо кратких сведений, изложенных в данном пособии, рекомендуется использовать источники из списка рекомендованной литературы и конспект лекций.

### ***Использование фреймворка Qt для создания пользовательского интерфейса***

Фреймворк Qt поддерживает три основных подхода к созданию пользовательских интерфейсов:

1. Создание интерфейса на этапе проектирования программы (design time) - интерфейс полностью создаётся при помощи средств IDE, например с помощью Qt Designer.

2. Создание интерфейса во время работы программы (run time) - интерфейс или его часть создаётся после запуска программы путём выполнения заданного программистом алгоритма.

3. Использование QML для описания структуры и функциональности интерфейса.

Несмотря на подобное разделение, данные подходы часто применяются в совокупности - когда части интерфейса создаются с использованием одного из подходов, а остальной другой. Также стоит заметить, что с программной точки зрения все они сводятся к созданию в режиме run time, однако для разработчика каждый из них имеет свои особенности.

### ***Создание интерфейса пользователя с помощью Qt Designer и подхода design time***

При использовании Qt Designer формы создаются в виде отдельных XML файлов с расширением \*.ui. При сборке проекта эти файлы используются для создания стандартных заголовочных файлов C++, описывающих алгоритм создания формы и размещения на ней виджетов. Стоит заметить, что изменение созданных подобным образом заголовочных файлов нежелательно и, как правило, не требуется.

Базовый процесс размещения виджетов на форме и связи слотов и сигналов с использованием Qt Designer рассмотрен в курсе лекций, поэтому не будет освещаться здесь. Однако некоторые моменты стоит затронуть отдельно.

### ***Создание главного меню формы с использованием Qt Designer***

При создании главного меню формы, изначально следует добавить разделы меню и пункты в них, просто описав действия, как показано на рисунке 1.1

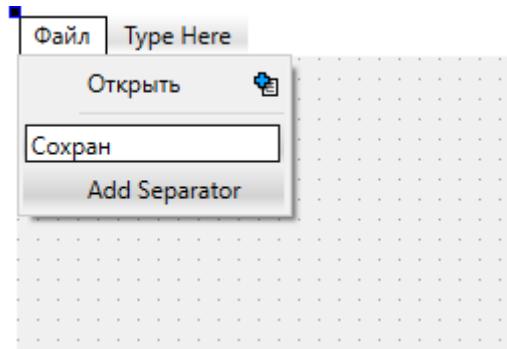


Рис. 1.1. Описание пунктов меню

При этом для каждого пункта меню будет создано своё действие, представленное в окне Action Editor. Двойным нажатием левой кнопкой мыши на действие в этом окне можно открыть окно дополнительных настроек, в котором есть возможность установить иконку, сочетание горячих клавиш, подсказку и др.

Для связи пунктов меню со слотами следует переключить Action Editor в режим Signals Slots Editor и добавить там новую связь, нажав на зелёный значок плюс. Далее следует задать параметры, например так, как показано на рисунке 1.2.

| Sender | Signal      | Receiver   | Slot    |
|--------|-------------|------------|---------|
| action | triggered() | MainWindow | close() |

Рис. 1.2. Связь пункта меню со слотом главного окна программы

В данном случае слот close формы будет связан с сигналом triggered действия, который испускается при нажатии на связанный с этим действием пункт меню.

### ***Использование файлов ресурсов и изображений***

Во время работы над интерфейсом может возникнуть потребность разместить на форме изображение. Для этого следует расположить на форме виджет QLabel и в его свойстве pixmap указать путь к файлу изображения, как показано на рисунке 1.3.

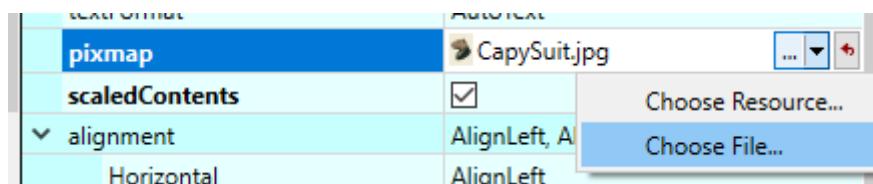


Рис. 1.3. Размещение на форме изображения

Для удобства, можно добавить в проект файл ресурсов, хранящий пути ко всем используемым в приложении изображениям и иным дополнительным файлам. Для этого необходимо нажать правой кнопкой мыши на проект в иерархии проектов, выбрать пункт меню Add New... и выбрать файл ресурсов Qt. После этого следует добавить новый префикс и добавить в него необходимые для работы приложения файлы, которые будет предложено при необходимости сохранить в директорию с исходными файлами программы.

После этого будет возможно выбирать файлы из числа сохранённых в ресурсе, как показано на рисунке 1.4.

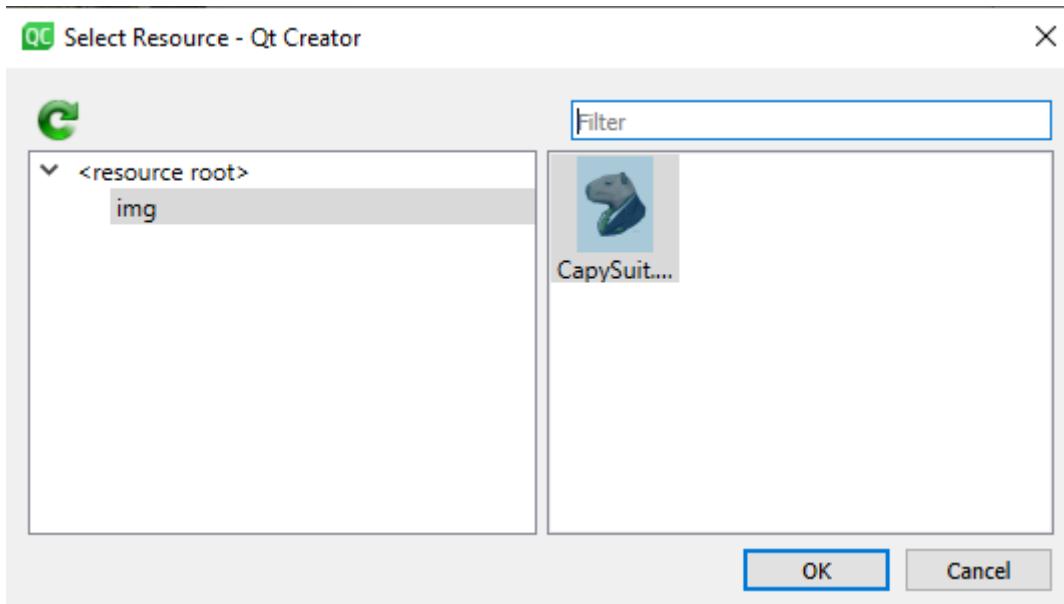


Рис. 1.4. Выбор изображения из ресурсов

### ***Создание интерфейса пользователя с помощью подхода run time***

Данный подход подразумевает создание всего или части интерфейса непосредственно при запуске программы. Следует отметить, что в рамках данной работы использование данного подхода подразумевает отказ от применения Qt Designer и использования файлов \*.ui. Весь интерфейс должен быть описан в конструкторе основной формы. Первым делом создаются все требуемые виджеты и производится настройка их свойств. Следом виджеты помещаются в компоновщики, при этом основной компоновщик назначается форме или центральному виджету. Затем с помощью функции connect устанавливается связь слотов и сигналов.

Для создания главного меню формы следует создать объект класса QMenu для каждого раздела главного меню, указав в качестве родительского основное меню (menubar) формы. Затем следует создать и добавить в него объекты класса QAction для каждого действия в меню.

Возможно использование файлов ресурсов для подключения изображений. Для использования изображений в программе следует использовать следующую конструкцию:

```
QPixmap("ПУТЬ_К_ИЗОБРАЖЕНИЮ")
```

Это создаст новый объект класса QPixmap из указанного изображения. После этого его можно присвоить требуемому виджету с помощью метода setPixmap. Следует учесть скалирование изображения при необходимости.

В остальном данный подход разобран в ходе лекций.

### ***Использование QML для описания структуры и функциональности интерфейса***

Данный подход подразумевает создание формы с помощью QML. При этом использование языка C++ сведено к минимуму. При этом подходе интерфейс представляют собой иерархию из элементов, описанных на QML.

Подробное описание синтаксиса QML было рассмотрено на лекциях, а также может быть найдено в источниках из списка рекомендованных.

При необходимости, в элементах может потребоваться объявление собственных свойств для хранения некой информации (например, текущего времени). Для этого используется конструкция, подобная следующей:

```
property int time: 42
```

В данном примере создано целочисленное свойство time. В дальнейшем можно преобразовывать данные из этого свойства в формат строки и помещать их в текстовое поле, например вот так:

```
item.text = time.toString()
```

Также стоит учитывать, что QML подразумевает использование различных модулей и для ряда элементов, таких как кнопки требуется подключить отдельный модуль. Важно при этом учитывать версию модуля.

Отдельно стоит заметить, что модуль QWebEngine и связанные с ним элементы для просмотра веб-страниц в QML не работает на компиляторах MinGW и требует использования компилятора MSVC

### ***Использование языка программирования C# и технологии WPF для создания пользовательского интерфейса***

Язык C# предполагает использование методов визуального программирования при построении пользовательских интерфейсов, и предлагает различные методы для этого.

Для этого платформа .NET предоставляет следующие технологии построения приложений с графическим интерфейсом пользователя — это Windows Forms и Windows Presentation Foundation (WPF).

В рамках данной работы предполагается использование следующих трёх методов:

1. Создание интерфейса с помощью технологии Windows Forms и подхода *design time*;
2. Создание интерфейса с помощью технологии Windows Forms и подхода *run time*;
3. Создание интерфейса с помощью технологии WPF и подхода *design time*.

Основы первых двух методов были рассмотрены на предыдущем курсе и потому не будут освещены в данном пособии, требуемые для выполнения работы теоретические сведения могут быть найдены в рекомендуемой литературе и конспекте лекций, поэтому далее будет рассмотрен только третий из них.

Технология WPF предполагает создание экранной формы, разделённой с программной точки зрения на две компоненты: исходный текст класса формы, описывающий поведение её компонентов и файл формата XAML, описывающий свойства формы и её компонентов.

Файл XAML представляет собой файл формата XML, содержащий все, размещённые на форме компоненты и их свойства. При компиляции на основе

этого файла происходит построение формы и связь её с соответствующим классом. Подробно структура данного файла рассмотрена в лекциях, здесь же будут затронуты только основные моменты.

Работа с файлом XAML возможно как в визуальном режиме с использованием встроенного в Visual Studio дизайнера, так и путём прямого редактирования непосредственно текста файла. В последнем случае важно внимательно следить за сохранением иерархии компонентов и соблюдением синтаксиса XAML, поскольку в случае ошибок в XAML файле Visual Studio может просто не показать форму в дизайнере, потому что внутренние средства среды не распознают структуру.

Пример XAML разметки простого WPF приложения:

```
<Window x:Class="WPFEExampleNew.MainWindow"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    xmlns:local="clr-namespace:WPFEExampleNew"
    mc:Ignorable="d"
    Title="WPF Example" Height="234" Width="264">
    <Grid>
        <Label Content="Hello from capybara" HorizontalAlignment="Center" Margin="0,41,0,0"
            VerticalAlignment="Top" FontSize="22" FontWeight="Bold"/>
        <Button Content="Button" HorizontalAlignment="Center" Margin="0,115,0,0" VerticalAlignment="Top"
            Click="Button_Click" Height="74" Width="194"/>
    </Grid>
</Window>
```

Данный пример создаёт окно с надписью и кнопкой, при нажатии на которую вызывается обработчик события из подключённого класса.

Следует заметить, что использование WPF предполагает активное редактирование файла XAML, так часть компонентов, например, таймер (DispatcherTimer), недоступны для размещения при помощи дизайнера.

## **Общая постановка задачи**

Выполнение данной работы состоит из трёх частей, в каждой из которых требуется создать простое приложение с графическим пользовательским интерфейсом.

В случае выполнения работы с использованием фреймворка Qt, следует учесть следующие требования:

1. Первая часть работы должна быть выполнена с применением инструмента Qt Designer для создания файлов форм формата \*.ui.

2. При выполнении задания из второй части работы, не допускается использование файлов форм, созданных средствами Qt Designer или аналогичными ему, интерфейс должен создаваться в момент запуска программы.

3. Третья часть работы должна быть выполнена с использованием технологии Qt Quick для создания интерфейса.

При выполнении работы с использованием языка C#, к поставленным задачам предъявляются следующие требования:

1. Первая часть работы должна быть выполнена с применением технологии Windows Forms и встроенного в Visual Studio инструмента для создания интерфейсов.

2. При выполнении задания из второй части работы, требуется также использовать технологию Windows Forms, однако не допускается использование встроенного в Visual Studio инструмента для создания интерфейсов или аналогичных ему, файл с описанием дизайна формы должен оставаться стандартным, весь интерфейс должен создаваться в момент запуска программы непосредственно в конструкторе класса формы после вызова метода InitializeComponent.

3. Третья часть работы должна быть выполнена с использованием технологии WPF для создания интерфейса.

По согласованию с преподавателем, возможно использование фреймворка Xamarin или технологии MAUI в третьей части работы.

Варианты представлены в виде изображений с примерным видом требуемого интерфейса. Возможно отклонение от представленного интерфейса, с сохранением основных элементов. Все кнопки должны демонстрировать свою работу, однако полноценная реализация возможностей приложения не требуется - кнопки могут просто выводить сообщение о своём срабатывании. Должна быть предусмотрена возможность изменения размеров окна приложения с сохранением взаимного расположения элементов интерфейса.

## **Варианты заданий**

### **Вариант 1**

#### **Задание для первой и второй части**

Требуется создать макет текстового редактора (рис. 1.5) с панелью инструментов и меню (рис. 1.6).

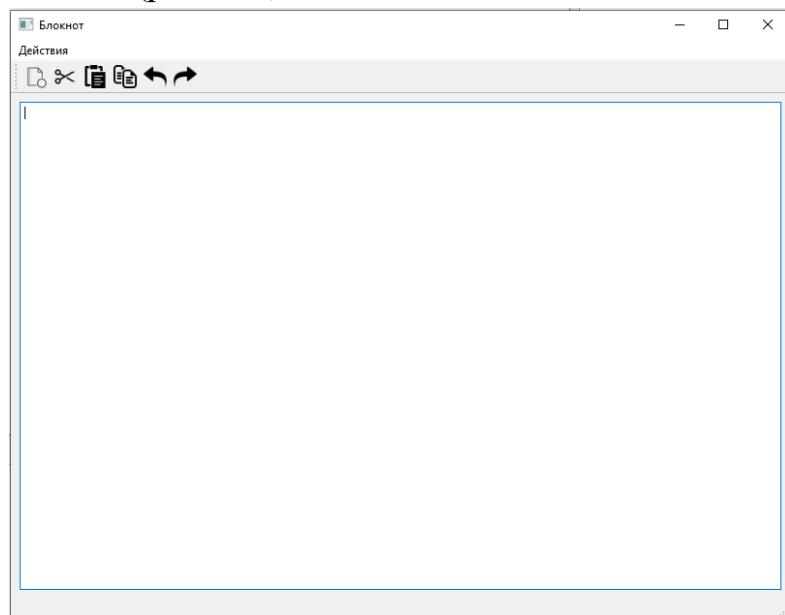


Рис. 1.5. Общий вид окна программы

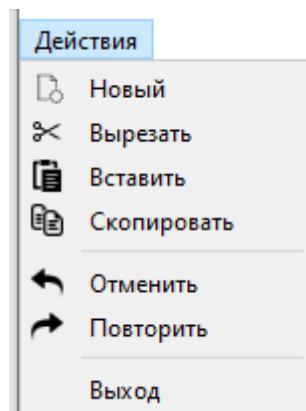


Рис. 1.6. Пункты меню

Команда “Новый” очищает содержимое текстового поля.

### Задание к третьей части

Требуется создать макет текстового редактора с кнопкой “закрыть окно”. Все элементы (поле для ввода текста, кнопка выхода, элементы оформления) должны быть созданы с использованием QML или WPF.

### *Вариант 2*

#### Задание для первой и второй части

Требуется создать макет электронного циферблата (рис. 1.7). Значение в текстовом поле задаётся либо при помощи “диска”, либо кнопками  $\pm 10$ .

В случае использования C#, ввиду отсутствия среди стандартных элементов «диска» можно заменить его обычным слайдером.

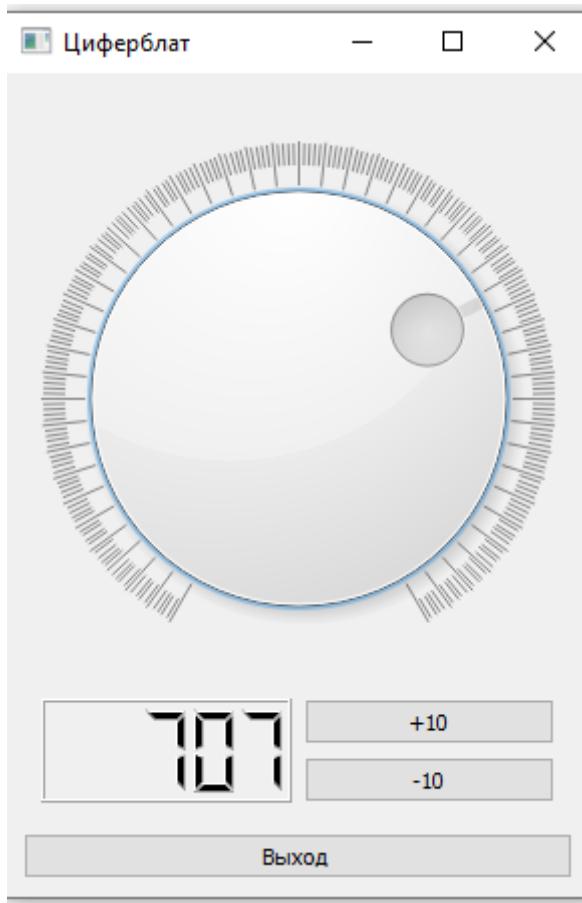


Рис. 1.7. Общий вид окна программы

Стоит заметить, что ввиду ограничений Qt Designer для первого задания может потребоваться использование stacked widget и spin box для корректной работы кнопок +10.

### **Задание для третьей части**

Требуется создать с помощью QML или WPF макет электронных часов, показывающие текущее время.

### ***Вариант 3***

#### **Задание для первой и второй части**

Требуется создать макет просмотровщика заранее заданных изображений (рис. 1.8).

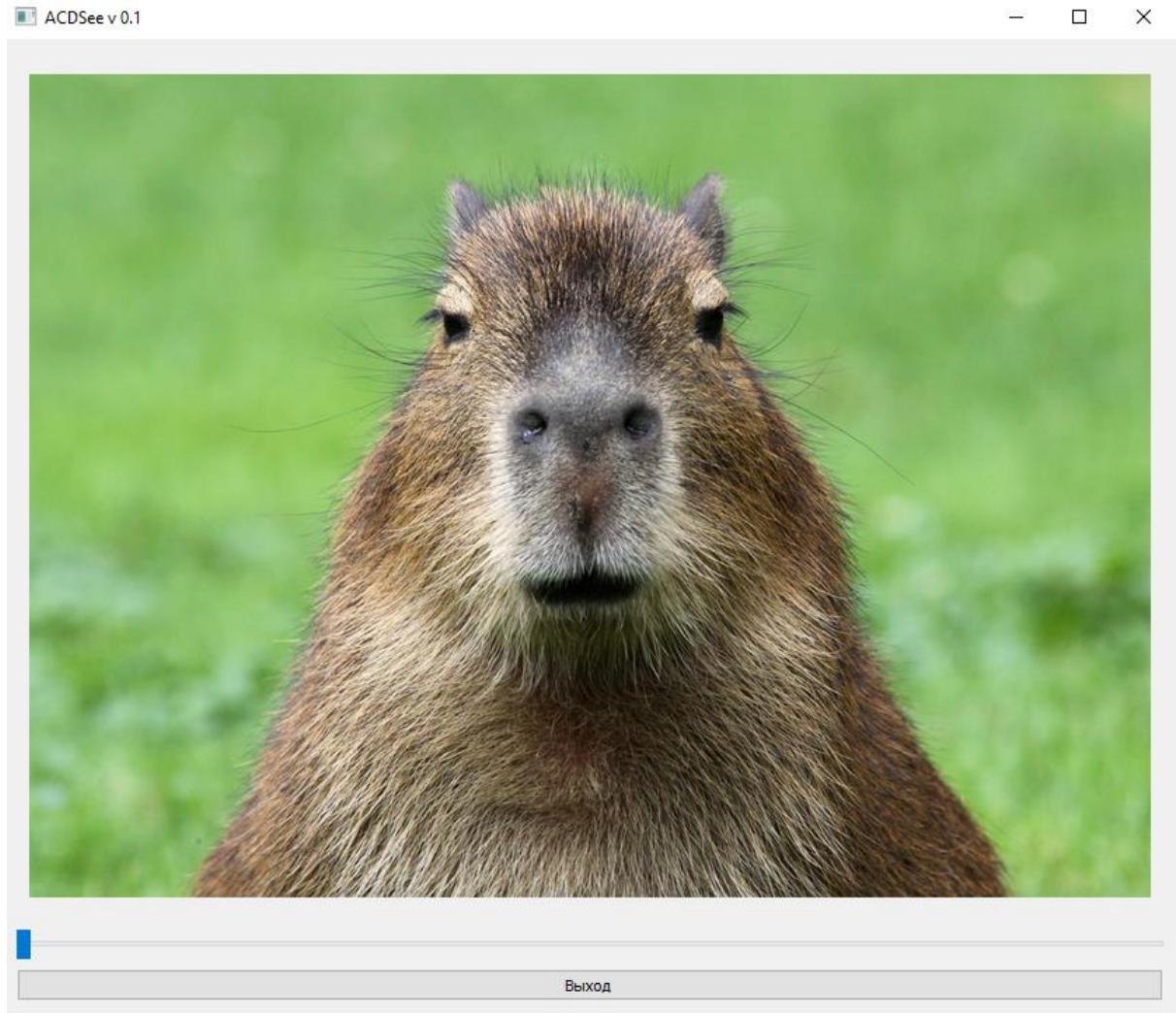


Рис. 1.8. Общий вид окна программы

Изображения расположены на разных страницах. Переключение между страницами выполняется при помощи слайдера или SpinBox.

### **Задание для третьей части**

Требуется создать с использованием QML или WPF простую анимацию перемещения изображения.

### ***Вариант 4***

#### **Задание для первой и второй части**

Требуется создать макет калькулятора (рис. 1.9).



Рис. 1.9. Общий вид окна программы

Кнопки должны нажиматься, но могут не выполнять никаких действий (за исключением кнопок С и CE, которые очищают поле для ввода). Прямой ввод в поле для ввода недопустим. Справа необходимо добавить аналог изображённого на рисунке переключателя память/журнал.

Основное меню можно не реализовывать.

### Задание для третьей части

Требуется создать с использованием QML или WPF макет интерфейса панели набора номера телефона. Номер может не набираться, но необходимо продемонстрировать что нажатие каждой кнопки приводит к какому-то результату.

### *Вариант 5*

Задание для первой и второй части

Требуется создать макет планировщика задач (рис. 1.10).

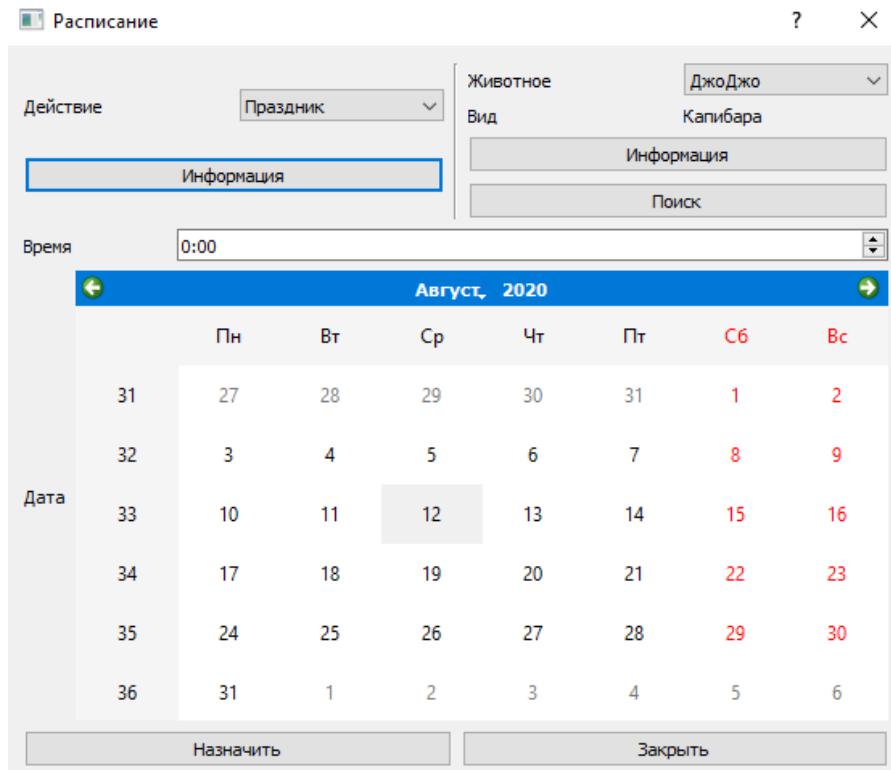


Рис. 1.10. Общий вид окна программы

Все кнопки должны нажиматься, но выполнять действие должна только кнопка “закрыть”.

### Задание для третьей части

Требуется создать макет аналогичного приложения с использованием QML или WPF.

### ***Вариант 6***

#### **Задание для первой и второй части**

Требуется создать макет IDE Qt Creator для режима Edit (рис. 1.11).

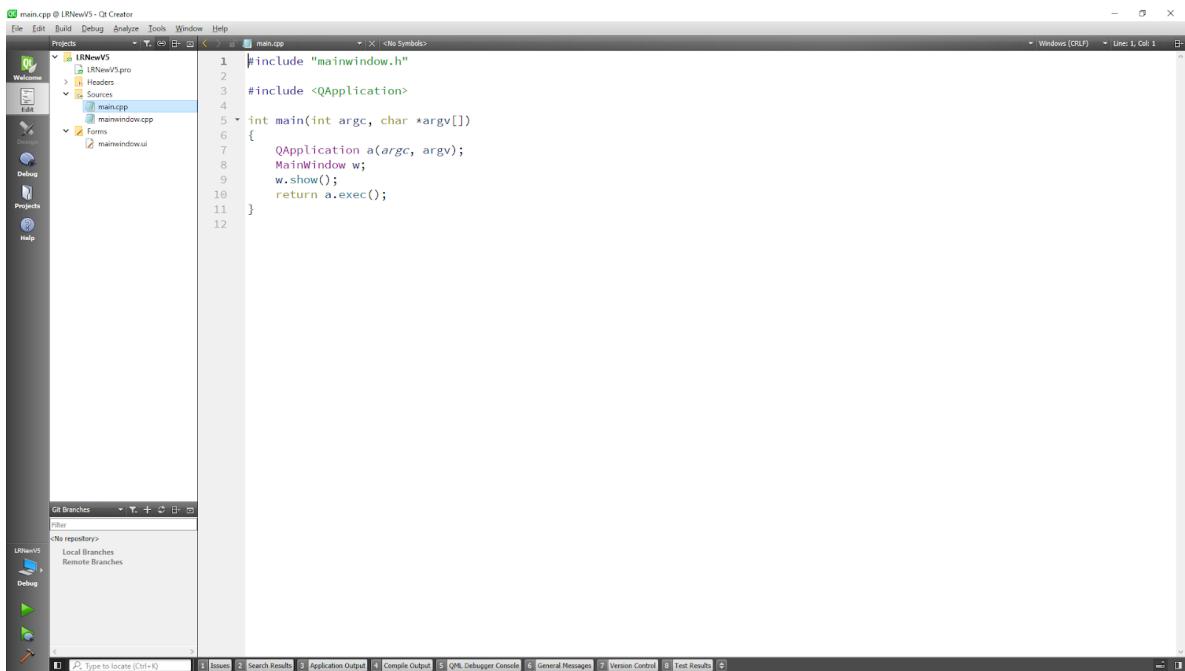


Рис. 1.11. Общий вид окна программы

Допустима замена части элементов на статичные изображения. Необходимо обеспечить возможность ввода текста. Подсветка синтаксиса не требуется. В основном меню необходимо обеспечить работу кнопки “закрыть”

#### **Задание для третьей части**

Требуется модифицировать программу из второй части, реализовав часть элементов интерфейса (например поле ввода текста) с помощью встраивания виджета, созданного с помощью QML или создать аналогичный интерфейс с помощью WPF

### ***Вариант 7***

#### **Задание для первой и второй части**

Требуется создать макет почтового клиента (рис. 1.12).

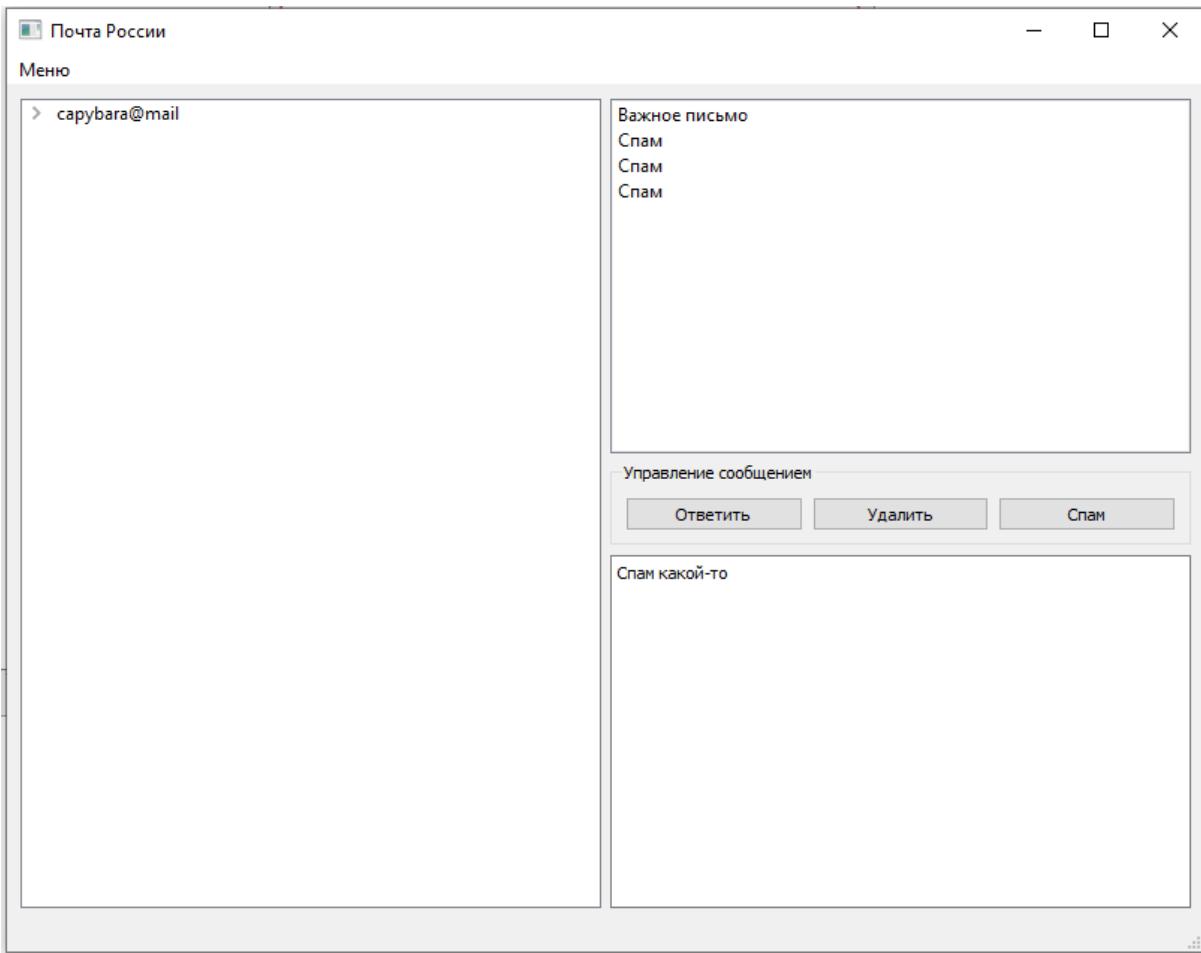


Рис. 1.12. Общий вид окна программы

В меню должны быть все необходимые для работы с почтой команды. Реализовывать работу кнопок и пунктов меню (кроме закрыть) не требуется. Слева должен быть раскрывающийся древовидный список почтовых ящиков. Изменять содержимое списков не требуется.

### **Задание для третьей части**

Требуется создать макет электронного секундомера с использованием QML или WPF.

### ***Вариант 8***

#### **Задание для первой и второй части**

Требуется создать макет видеопроигрывателя (рис. 1.13).

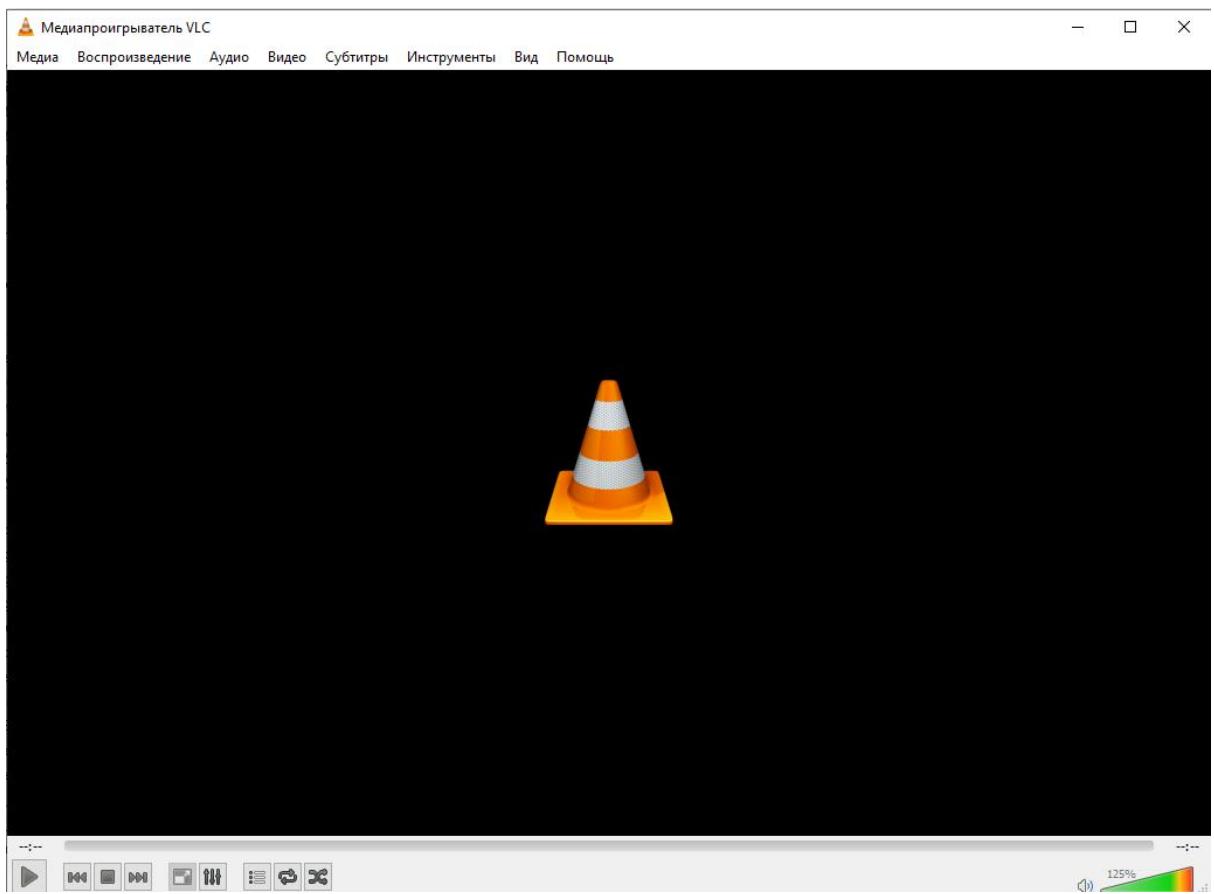


Рис. 1.13. Общий вид окна программы

В меню должны быть все необходимые для работы команды. Реализовывать работу кнопок и пунктов меню (кроме закрыть) не требуется. Для управления громкостью можно разместить обычный слайдер. Реализовывать воспроизведение видео не требуется.

### **Задание для третьей части**

Требуется создать приложение, воспроизводящее заранее указанное видео, с использованием QML или WPF.

### ***Вариант 9***

#### **Задание для первой и второй части**

В качестве первого и второго задания требуется создать макет приложения для записи экрана (рис. 1.14).

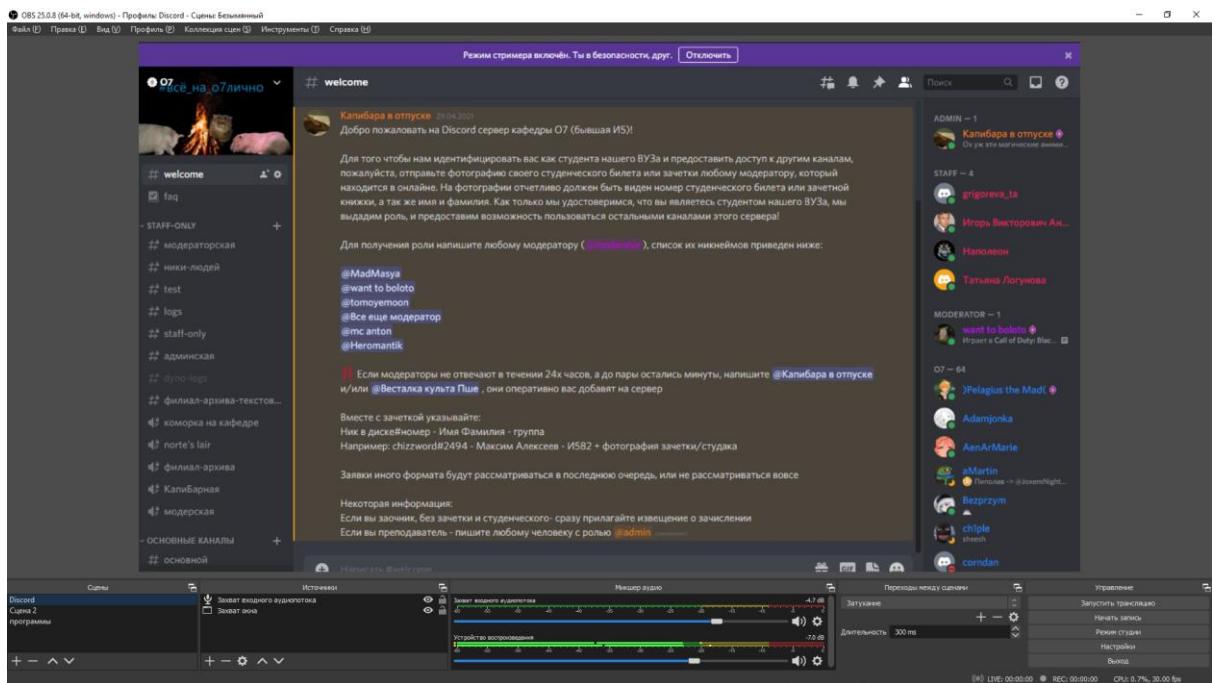


Рис.14. Общий вид окна программы

В меню должны быть все необходимые для работы команды. Реализовывать работу кнопок и пунктов меню (кроме закрыть) не требуется. Реализовывать запись экрана не требуется.

### **Задание для третьей части**

Требуется создать с использованием QML или WPF приложение, состоящее из нескольких экранов и реализующую переход между ними при помощи кнопок. Число экранов должно быть не менее семи,

### ***Вариант 10***

#### **Задание для первой и второй части**

Требуется создать прототип интерфейса приложения для редактирования таблиц (рис 1.15).

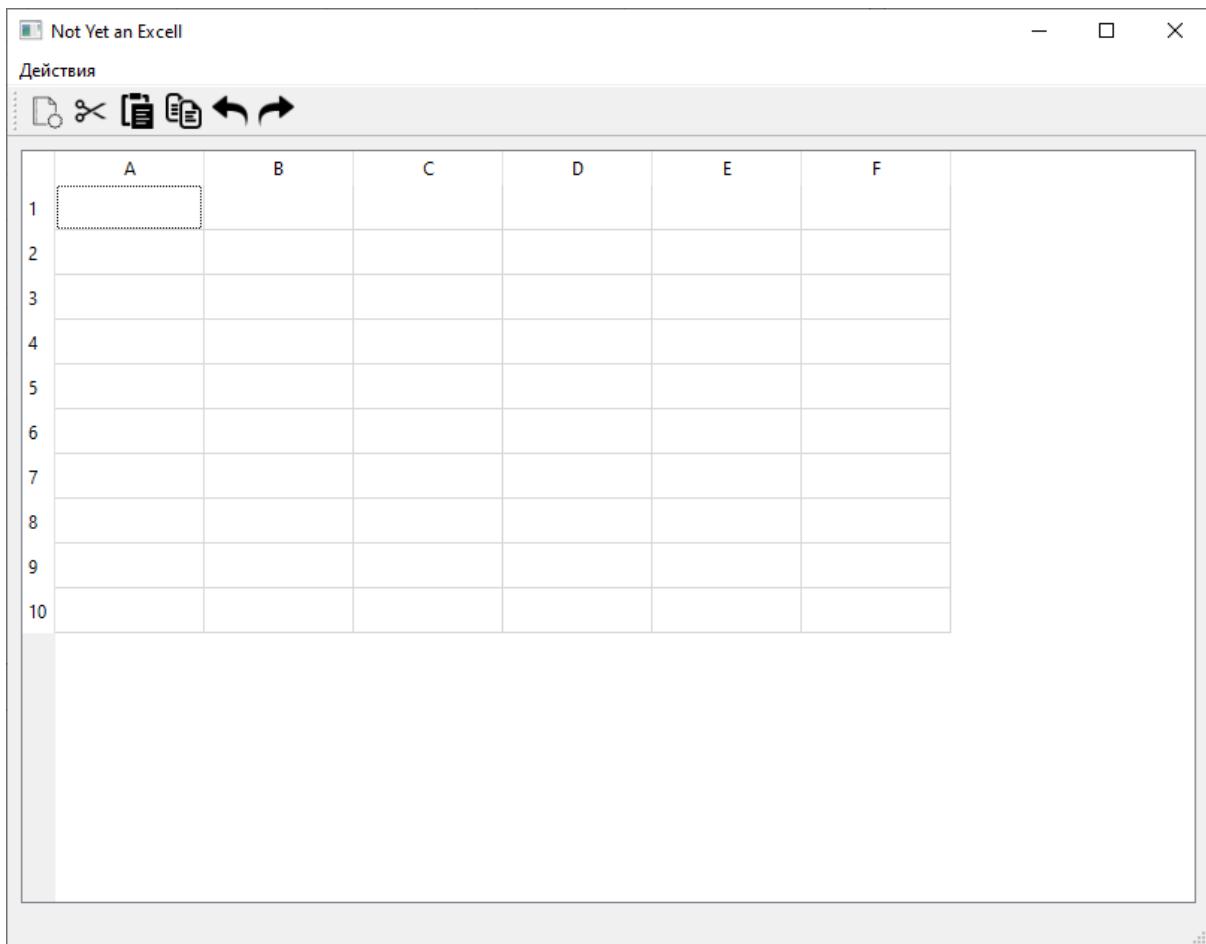


Рис. 1.15. Общий вид окна программы

Обязательно реализация только действий “Очистить” и “Выход”

### **Задание для третьей части**

Требуется создать с использованием QML или WPF программу для просмотра веб страниц по заранее указанному адресу.

### ***Вариант 11***

#### **Задание для первой и второй части**

Требуется создать макет интерфейса для интернет браузера с поддержкой вкладок (рис. 1.16).

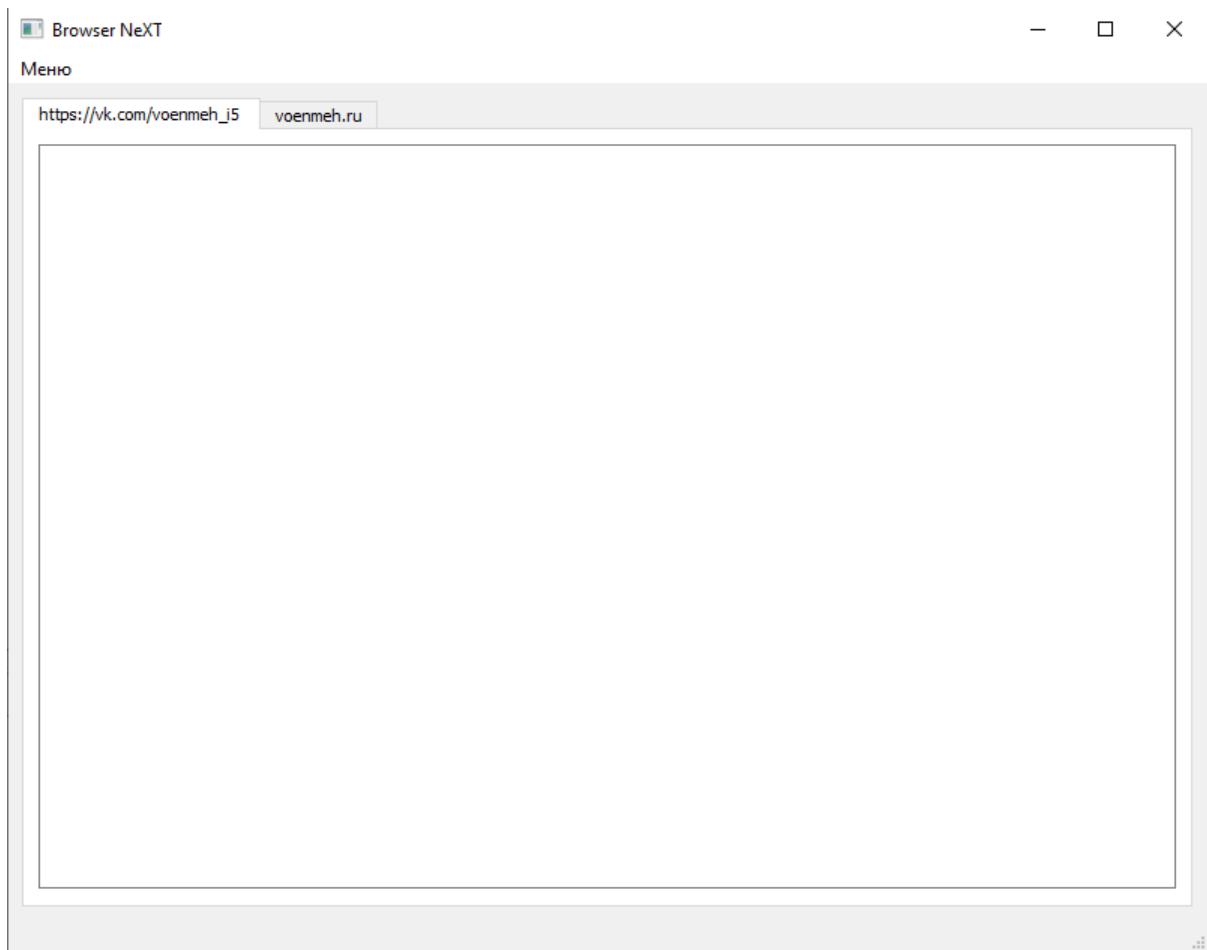


Рис. 1.16. Общий вид окна программы

На каждой вкладке должен быть свой набор виджетов (как минимум виджет для ввода/демонстрации текста). Также необходимо наполнить меню различными действиями (реализовывать требуется только действие “закрыть”).

### **Задание для третьей части**

Требуется создать с использованием QML или WPF простую анимацию изменения цвета хотя бы четырёх различных элементов.

### ***Вариант 12***

#### **Задание для первой и второй части**

Требуется создать макет интерфейса для файлового менеджера (рис. 1.17).

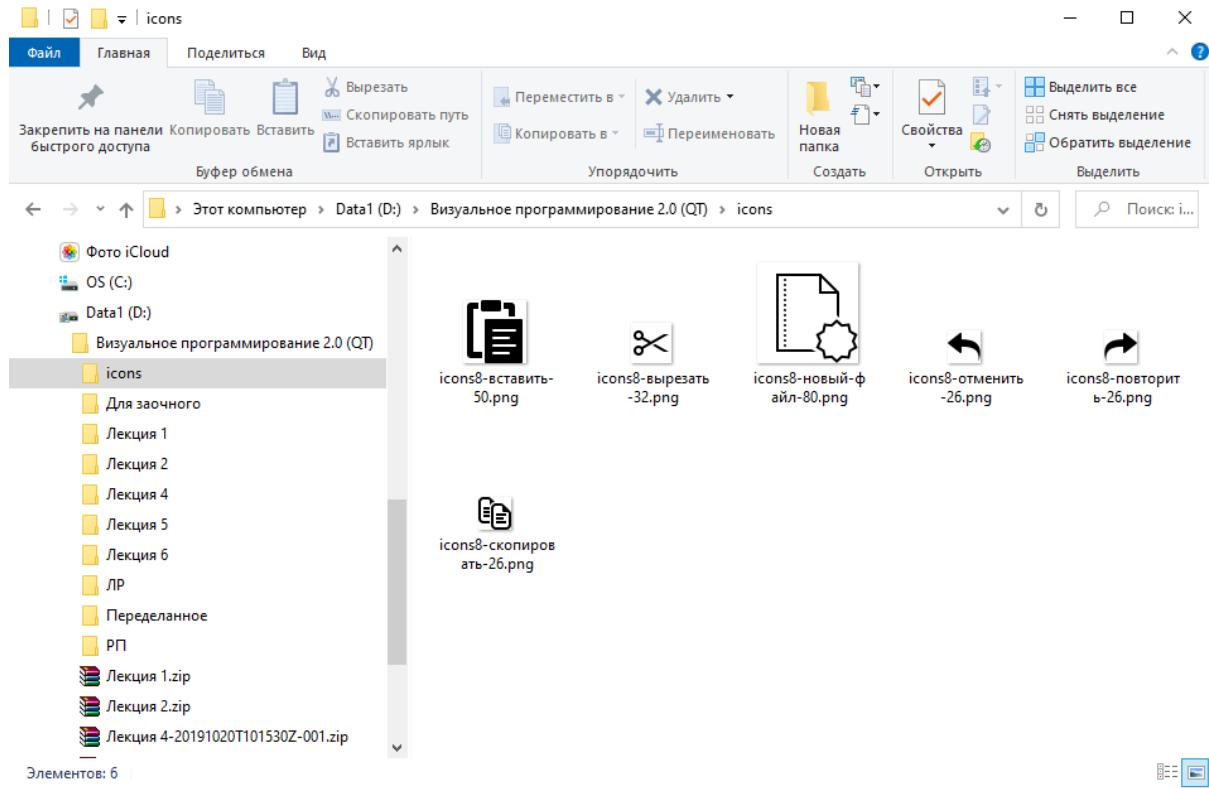


Рис. 1.17. Общий вид окна программы

Панель инструментов можно не реализовывать. В левой части должен быть расположен Tree Widget с перечнем директорий (можно без иконок). В правой либо список файлов, либо иконки. Синхронизация с реальной файловой системой не требуется.

### Задание для третьей части

Требуется создать с использованием QML или WPF прототип интерфейса калькулятора, нажатие на каждую кнопку должно что-то делать, но не обязательно производить вычисления.

### ***Вариант 13***

#### **Задание для первой и второй части**

Требуется создать макет интерфейса для графического редактора (рис. 1.18).

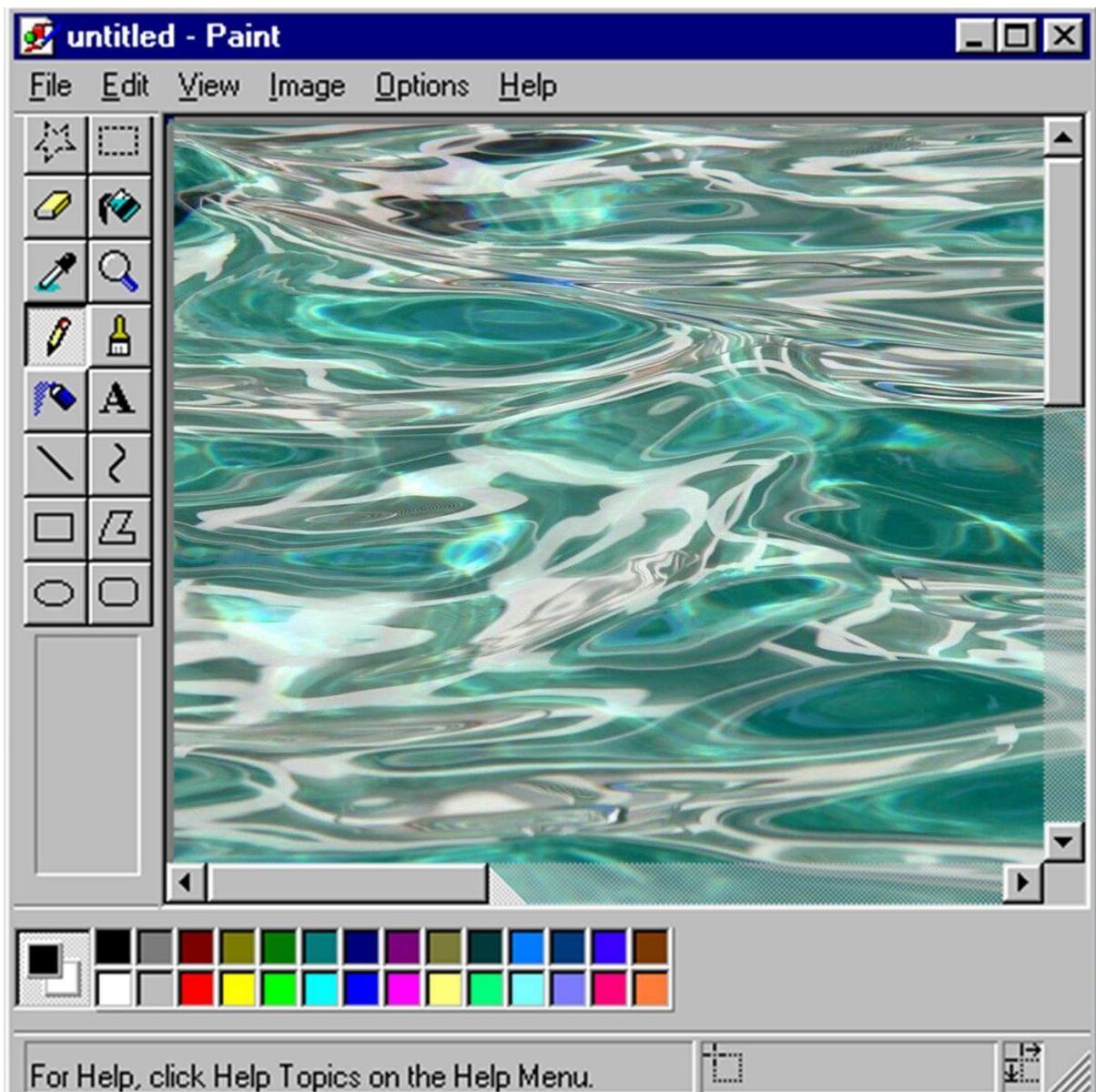


Рис. 1.18. Общий вид окна программы

Реализовывать работу с изображением не требуется, однако общая структура интерфейса должна примерно соответствовать изображённой на рисунке.

### Задание к третьей части

Требуется создать с использованием QML или WPF прототип интерфейса стратегической игры (рис. 1.19).



Рис. 1.19. Вариант интерфейса стратегической игры

## *Вариант 14*

### **Задание для первой и второй части**

Требуется создать макет интерфейса для аудио проигрывателя (рис. 1.20).



Рис. 1.20. Общий вид окна программы

Реализовывать работу с музыкой не требуется, ряд элементов интерфейса могут быть заменены изображениями либо стандартными виджетами, похожими по внешнему виду.

### Задание к третьей части

Требуется создать с использованием QML или WPF прототип интерфейса программы для управления умным домом (рис. 1.21).

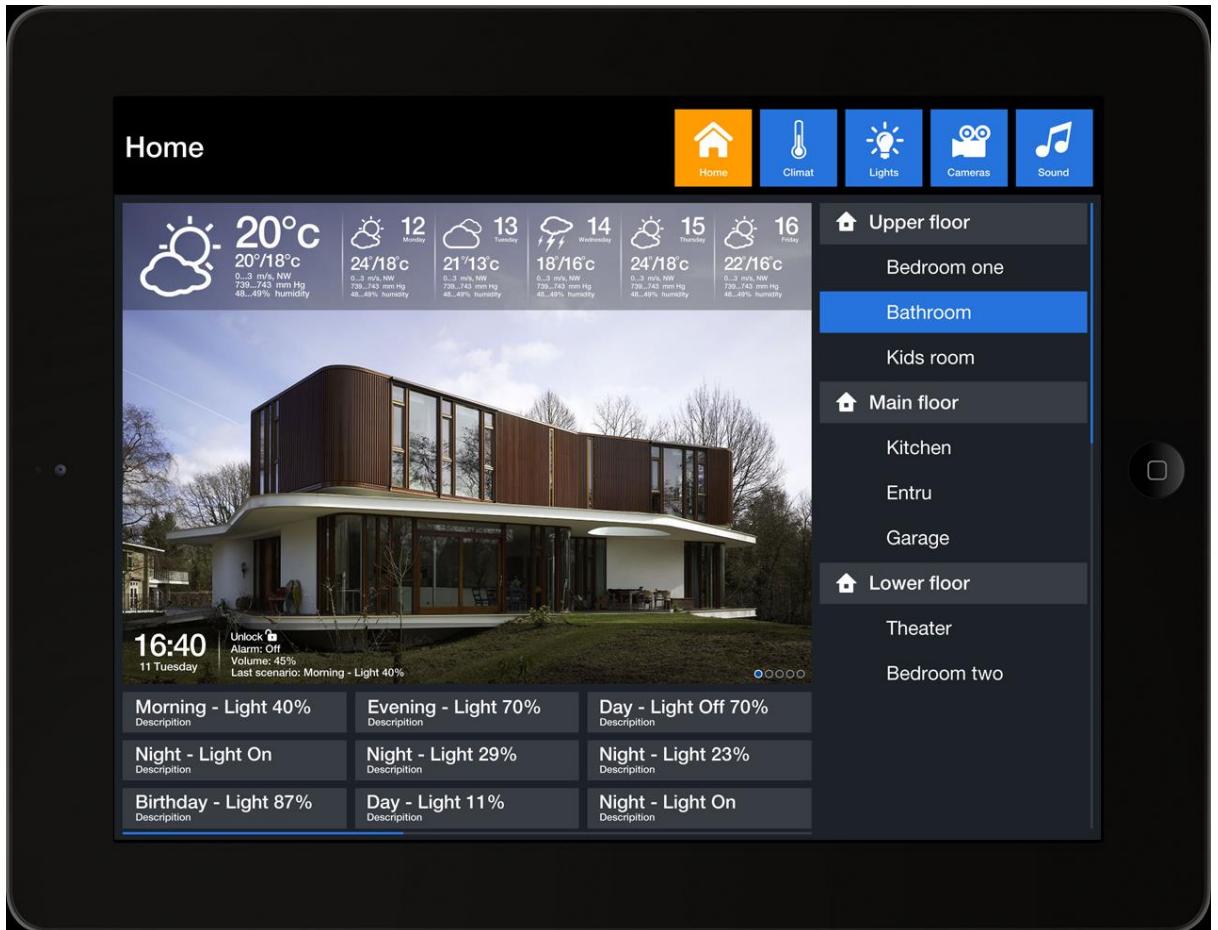


Рис. 1.21. Вариант интерфейса программы управления системами умного дома

### Вариант 15

#### Задание для первой и второй части

Требуется создать макет интерфейса для ftp-клиента (рис. 1.22).

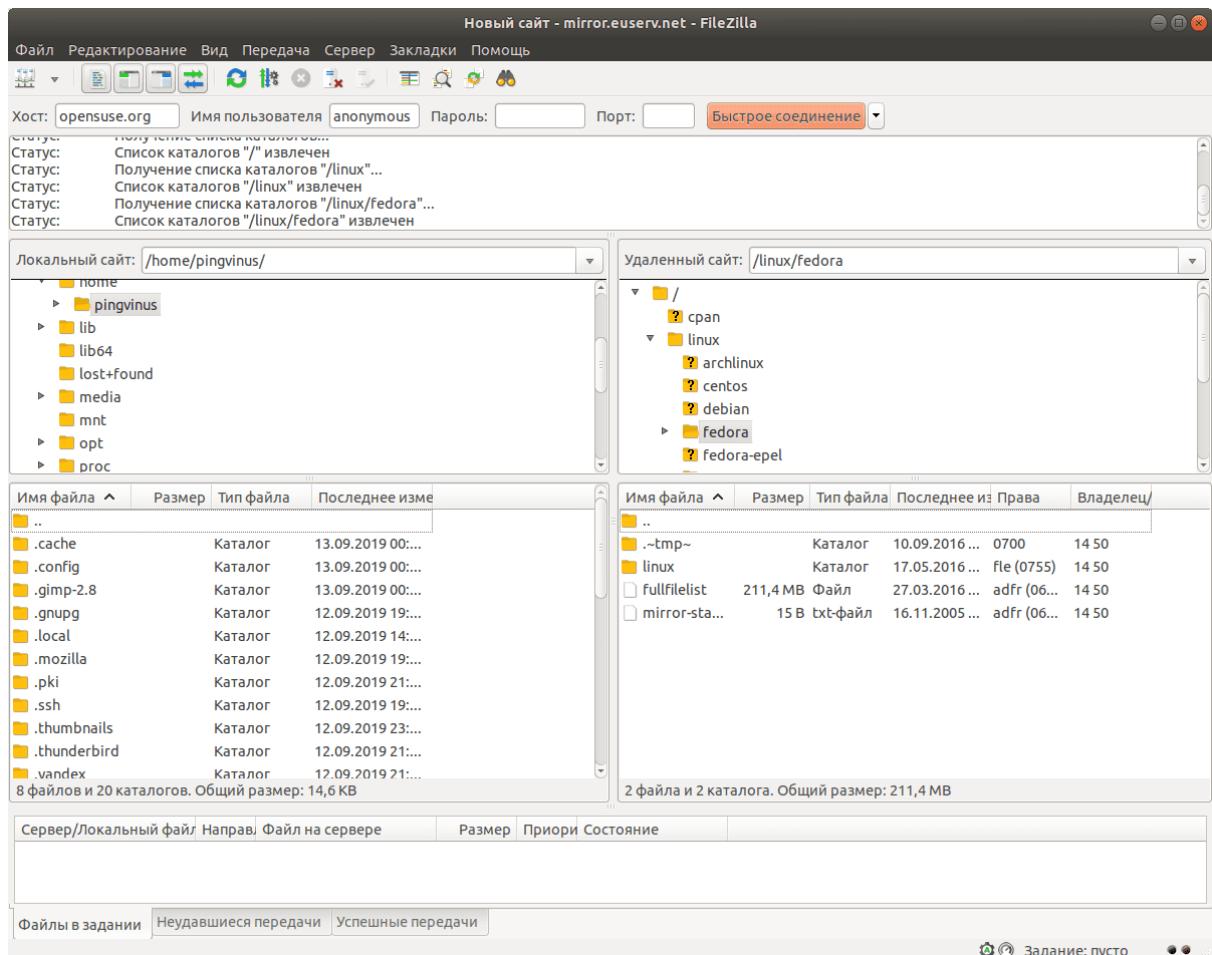


Рис. 1.22. Общий вид окна программы

Реализовывать работу не требуется. Обязательно использование списков для отображения условной файловой структуры. Допускается не использовать иконки папок.

### Задание к третьей части

Требуется создать с использованием QML или WPF прототип интерфейса интерактивной(умной) доски (рис. 1.23).



Рис. 1.23. Вариант интерфейса интерактивной доски

### ***Вариант 16***

#### **Задание для первой и второй части**

Требуется создать макет интерфейса для p2p-клиента (рис. 1.24).

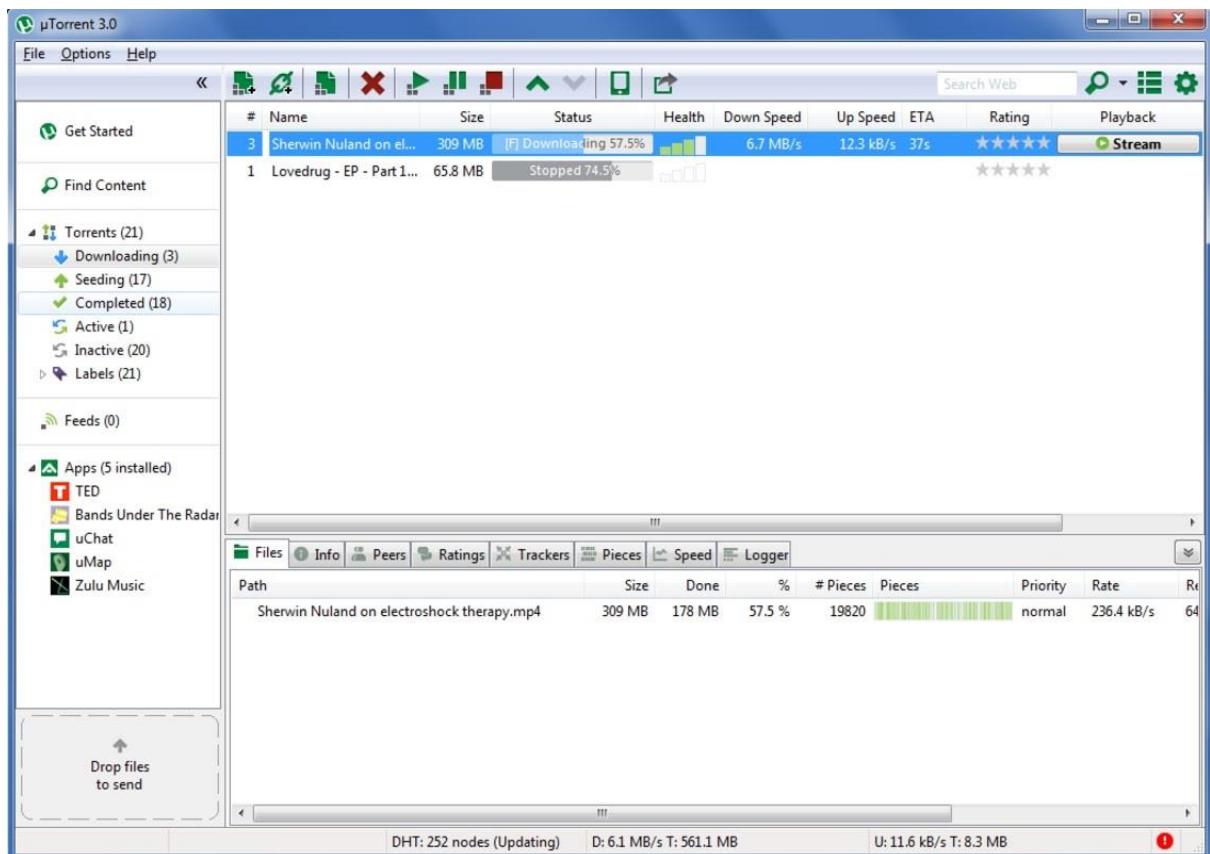


Рис. 1.24. Общий вид окна программы

Реализовывать работу не требуется.

### Задание к третьей части

Требуется создать с использованием QML или WPF прототип интерфейса операционной системы для мобильных устройств (смартфона или планшета). Учсть основное окно, окно приложений, окно настроек, контактов и звонков (рис. 1.25).



Рис. 1.25. Интерфейс трёх окон мобильной операционной системы

### Вариант 17

#### Задание для первой и второй части

Требуется создать макет интерфейса администратора для СУБД (рис. 1.26).

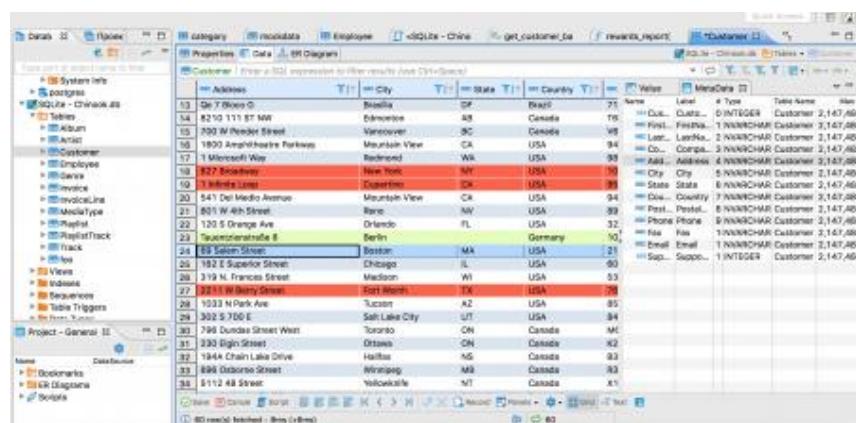


Рис. 26. Общий вид окна программы

Реализовывать работу не требуется.

## Задание к третьей части

Требуется создать с использованием QML или WPF прототип интерфейса бортовой системы автомобиля. Учесть одно окно настроек, информационное окно и основное (рис. 1.27).

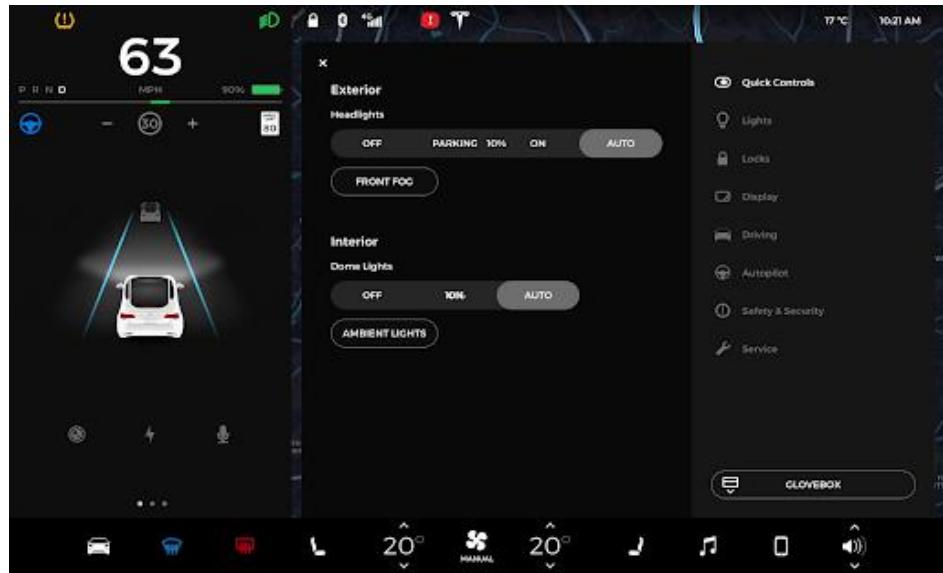


Рис. 1.27. Вариант интерфейса бортовой системы автомобиля

## Вариант 18

### Задание для первой и второй части

Требуется создать макет интерфейса клиентской программы магазина компьютерных игр (например Steam) (рис. 1.28).

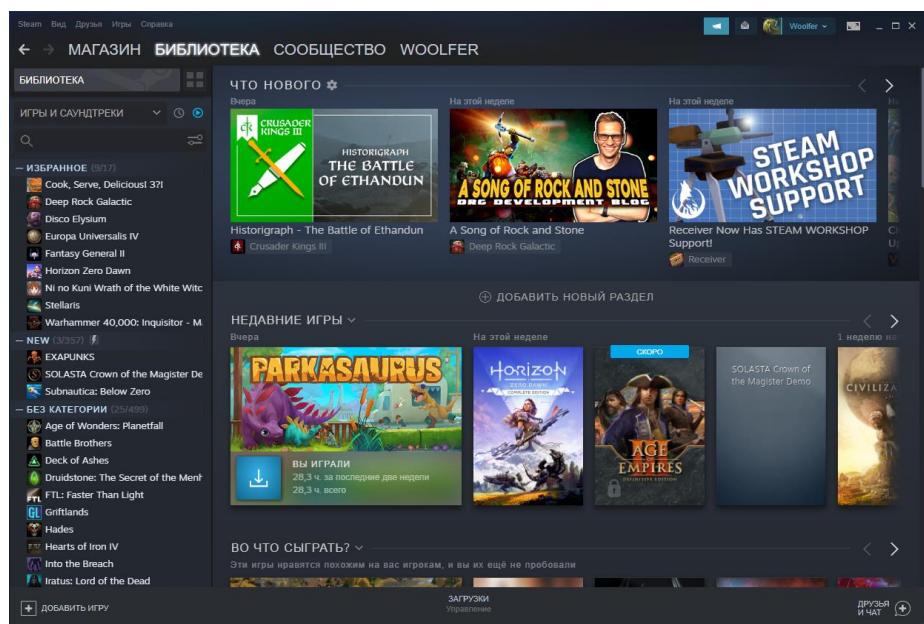


Рис. 28. Общий вид окна программы

Реализовывать работу не требуется.

### Задание к третьей части

Требуется создать с использованием QML или WPF прототип интерфейса операционной системы для консоли. Учесть библиотеку игр, настройки и друзей (рис. 1.29).

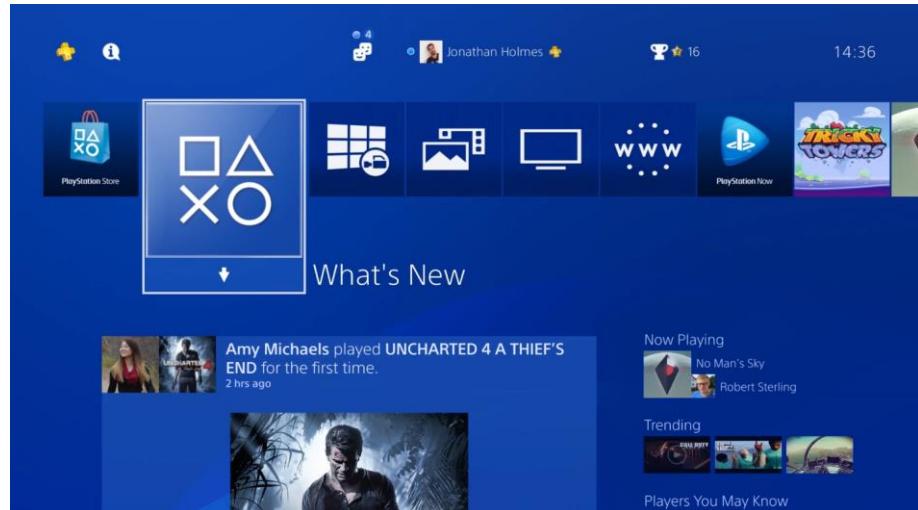


Рис. 1.29. Вариант интерфейса ОС для консоли

### Вариант 19

#### Задание для первой и второй части

Требуется создать макет интерфейса настроек BIOS (рис. 1.30).

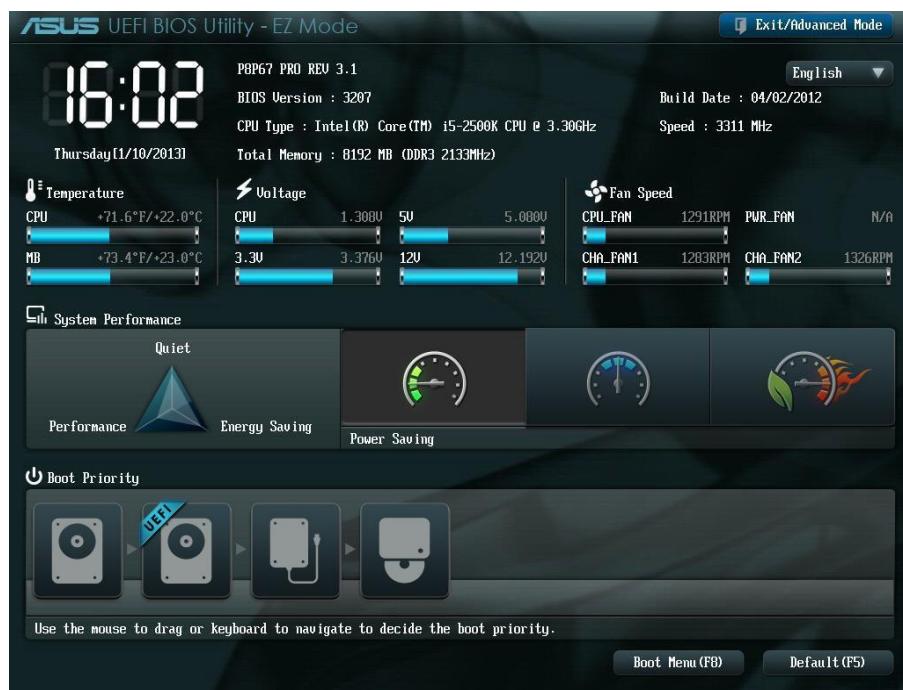


Рис. 1.30. Общий вид окна программы

Реализовывать работу не требуется.

### Задание к третьей части

Требуется создать с использованием QML или WPF прототип интерфейса клиентской программы онлайн кинотеатра. Учсть окно фильма, библиотеки и окна поиска (рис. 1.31).

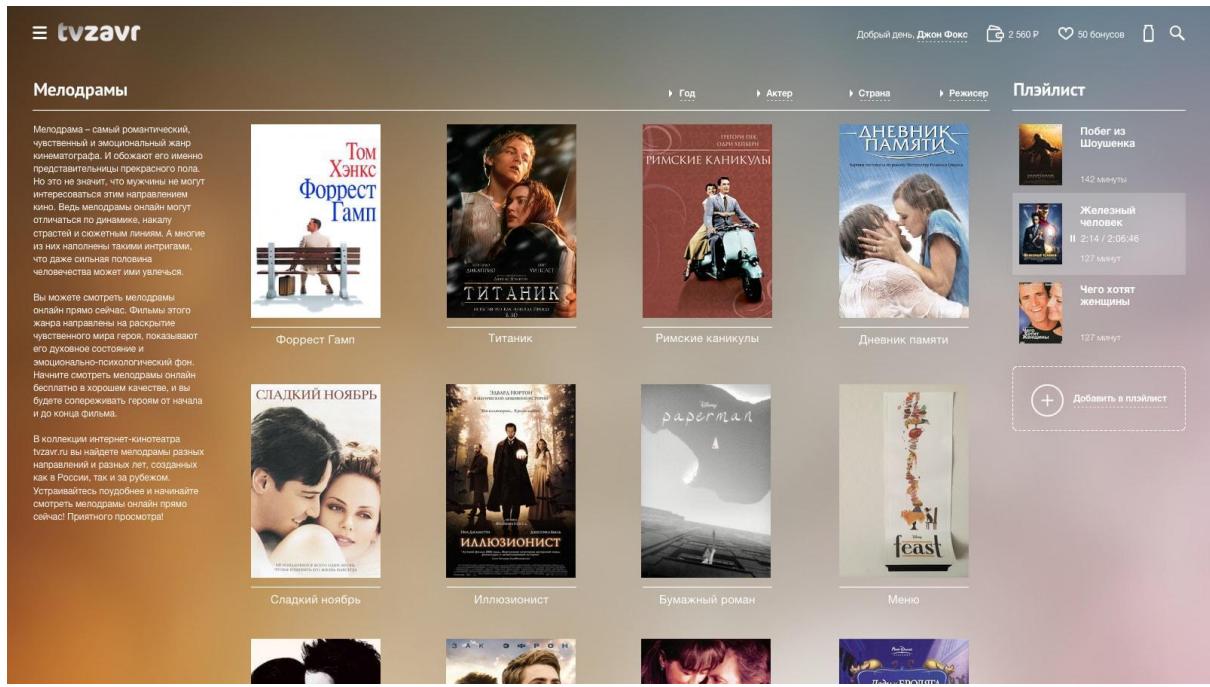


Рис. 1.31. Вариант интерфейса онлайн кинотеатра

### Вариант 20

#### Задание для первой и второй части

Требуется создать макет интерфейса клиентского приложения для системы мгновенного обмена сообщениями Discord (рис. 1.32)

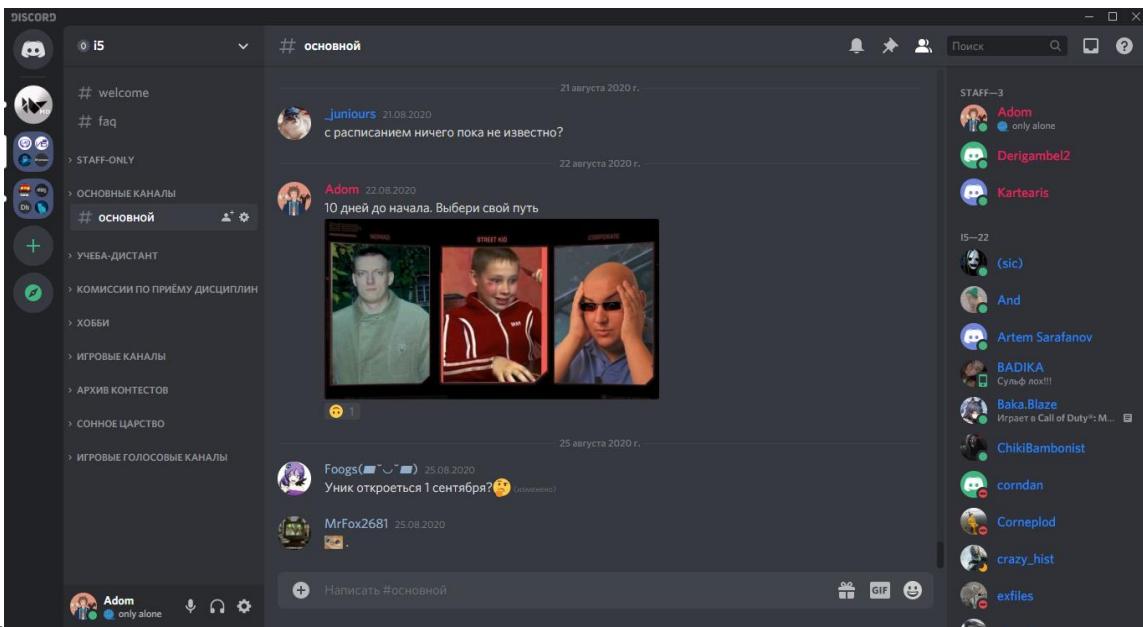


Рис. 1.32. Общий вид окна программы

Реализовывать работу не требуется.

### Задание к третьей части

Требуется создать с использованием QML или WPF прототип интерфейса мобильной версии того же клиентского приложения

### *Вариант 21*

#### Задание для первой и второй части

Требуется создать макет интерфейса игры “Сапёр” (рис. 1.33).



Рис. 1.33. Общий вид окна программы

Реализовывать работу не требуется.

## **Задание к третьей части**

Требуется создать с использованием QML или WPF прототип интерфейса управления инвентарём игры (рис. 1.34).



Рис. 1.34. Вариант интерфейса управления инвентарём игры

## **Вариант 22**

### **Задание для первой и второй части**

Требуется создать макет интерфейса программы для управления умного дома (несколько страниц). Учесть информационное окно, окна настройки отдельных модулей (хотя бы трёх) и панель управления (рис. 1.35).

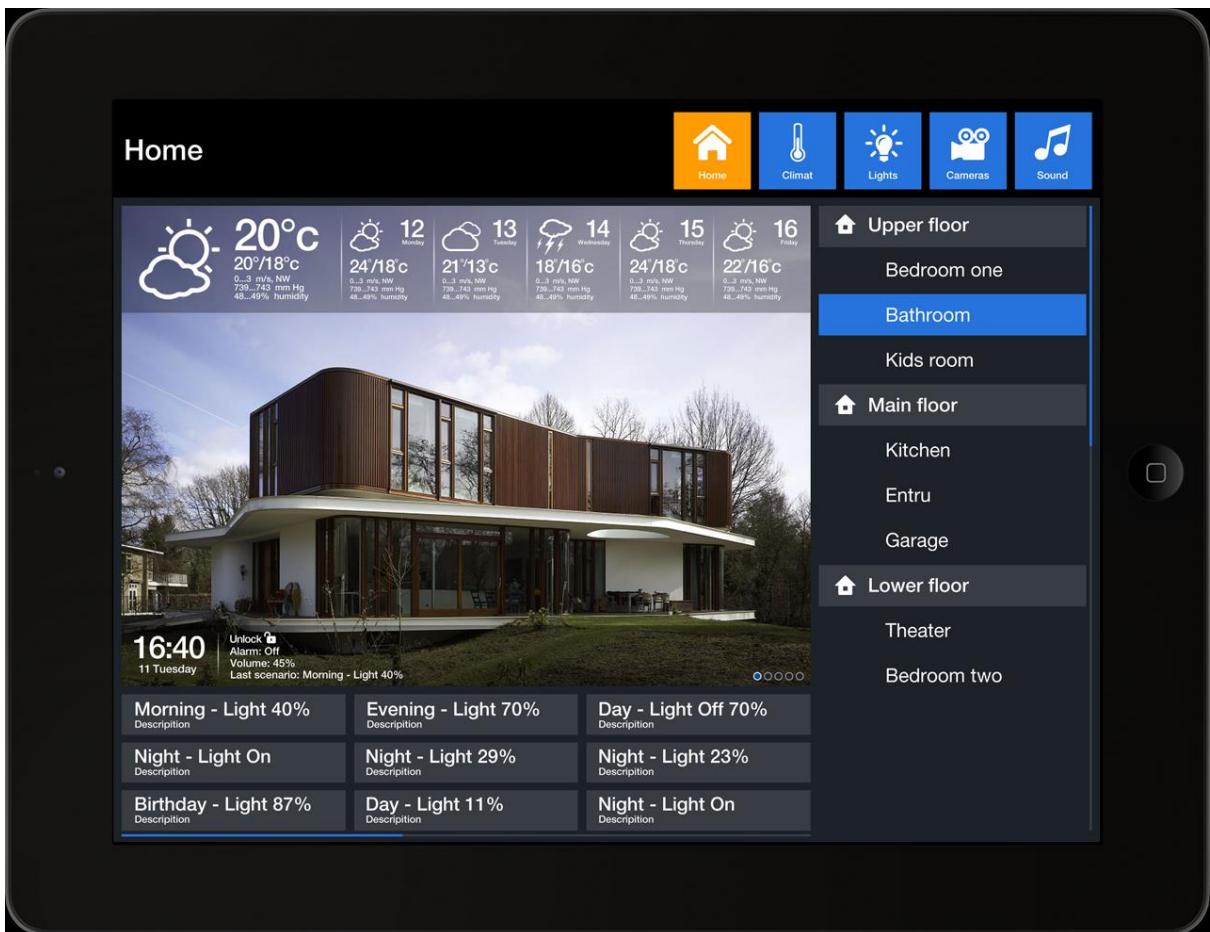


Рис. 1.35. Общий вид окна программы

### Задание к третьей части

Требуется создать с использованием QML или WPF прототип интерфейса настроек BIOS (рис. 1.36)



Рис. 36. Вариант интерфейса настроек BIOS

## **Вариант 23**

Задание для первой и второй части

Требуется создать макет интерфейса программы для управления ядерным реактором (рис. 1.37).

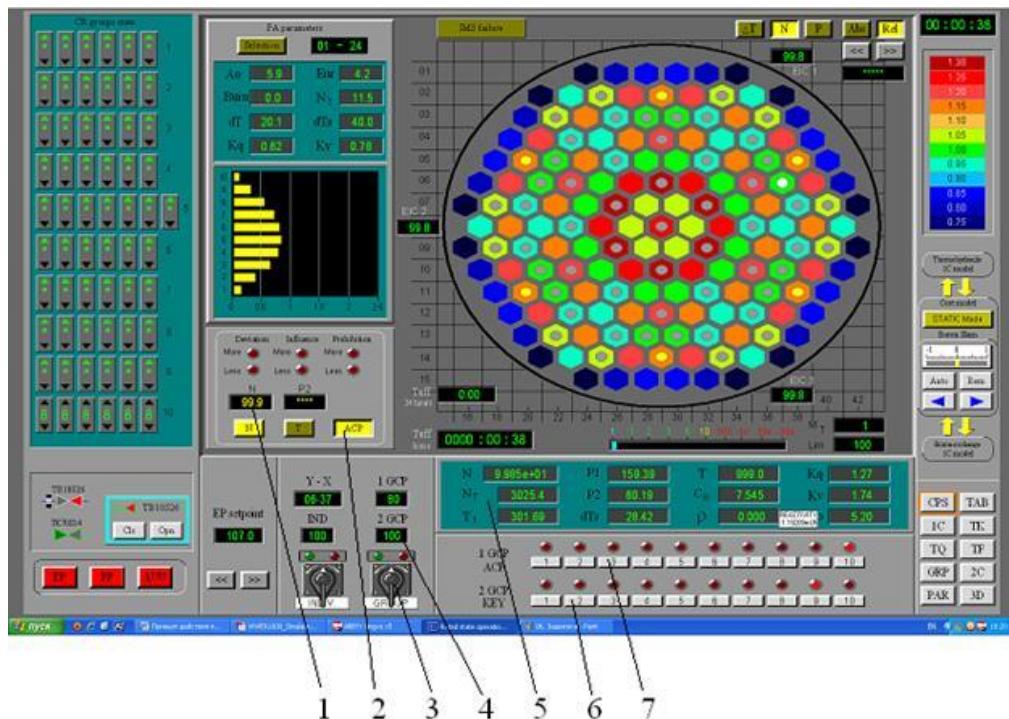


Рис. 1.37. Общий вид окна программы

Реализовывать работу программы не требуется!

## **Задание к третьей части**

Требуется создать с использованием QML или WPF прототип интерфейса браузера (рис. 1.38).

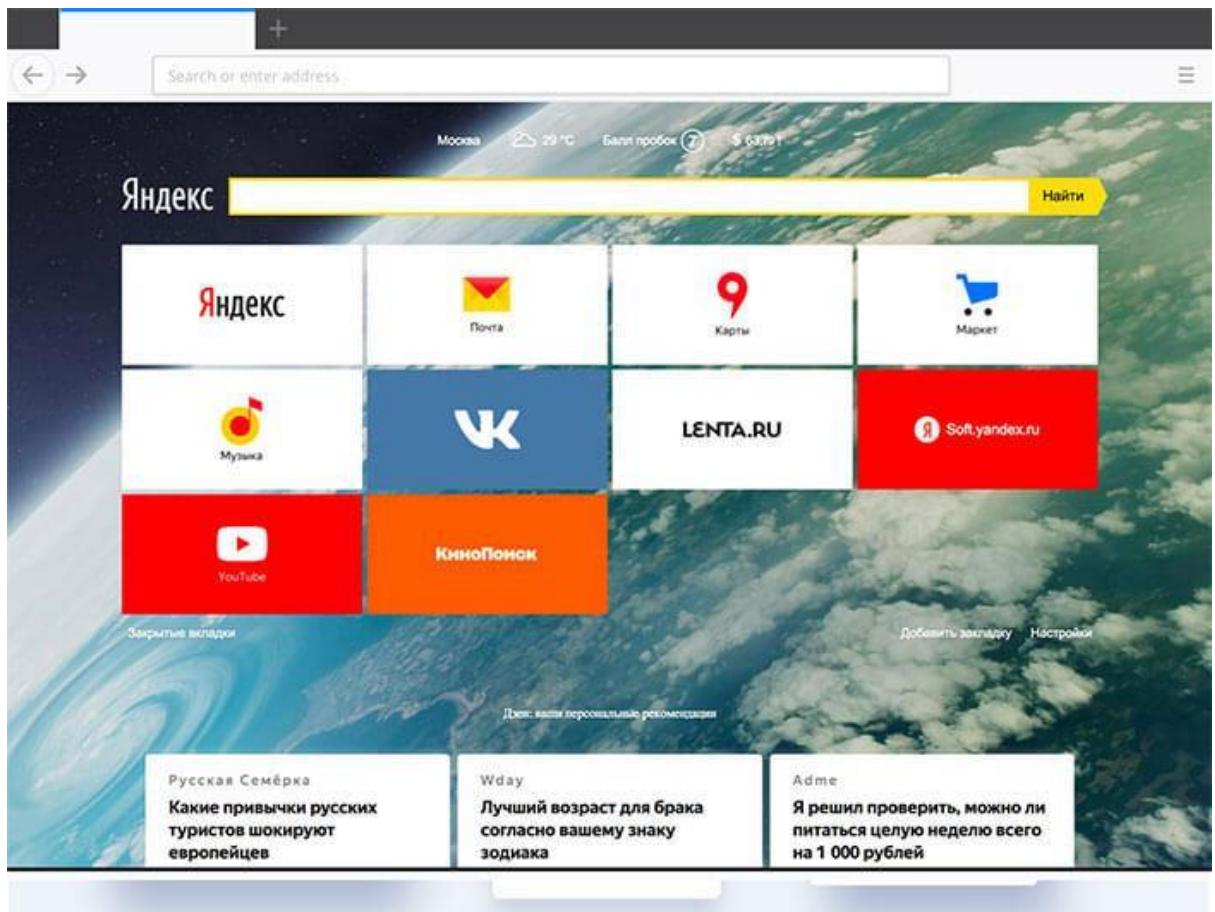


Рис. 1.38. Вариант интерфейса браузера

### ***Вариант 24***

#### **Задание для первой и второй части**

Требуется создать макет интерфейса программы для запуска межконтинентальных баллистических ракет.

Пример интерфейса реального приложения предоставлен по понятным причинам не будет, но приветствуется проявление фантазии при разработке.

Реализовывать работу программы не требуется!

#### **Задание к третьей части**

Требуется создать с использованием QML или WPF прототип мобильной версии такого же приложения для телефонов iPhone 8 Plus

## **Контрольные вопросы**

Контрольные вопросы к данной работе разбиты на три категории: общие, Qt и C#.

### ***Общие вопросы:***

1. Что такое визуальное программирование?
2. Дайте описание технологий (фреймворков) используемых во время работы над заданием.
3. Что является основой компонентной модели в используемом инструменте разработки?
4. Какие классы могут быть использованы в качестве основного графического элемента программы? Их особенности.
5. Что такое компоновщики? Их виды и использование.
6. Как происходит размещение компонентов на форме без использования компоновщиков?
7. Что входит в состав сборки или проекта, созданного в процессе выполнения задания?

### ***При выполнении работы средствами фреймворка Qt:***

1. Что представляет собой механизм слотов и сигналов в Qt?
2. Назовите основные типы виджетов в Qt.
3. Как происходит связывание сигналов и слотов?
4. Какова структура \*.ui файла формы в Qt?
5. Назовите основные элементы QML.
6. Какова структура файла \*.qml?
7. Как использовать компоновщики в QML?
8. Как использовать элементы QML в проекте C++?
9. Как использовать анимацию в QML?

### ***При выполнении работы с использованием языка программирования***

#### **C#:**

1. Как выполняется обработка событий в C#?
2. Назовите основные типы элементов управления в Windows Forms.

3. Как происходит обработка событий?
4. Какова структура файла формы в Windows Forms?
5. Какова структура файла XAML?
6. Назовите основные элементы управления WPF.
7. Как использовать компоновщики в WPF?
8. Как происходит взаимодействие с элементами WPF?
9. Как использовать анимацию в WPF?

## **ПРАКТИЧЕСКАЯ РАБОТА 2**

### **«Создание собственных элементов управления и работа с ними»**

#### **Теоретические сведения**

Инструменты визуального программирования как правило включают в себя богатый набор компонент для построения пользовательского интерфейса. Однако, зачастую задача требует создание собственного элемента управления, и в этом случае появляется желание сохранить его в виде отдельного компонента для дальнейшего использования в других проектах. Как Qt, так и C# предоставляют такую возможность.

Стандартный набор виджетов в Qt хоть и обширен, но по определению не может охватывать все возможные ситуации. На этот случай в Qt создан механизм создания собственных виджетов либо с нуля (как потомков класса QWidget), либо наследуя от одного из уже существующих. При этом существует возможность сохранить созданный виджет в виде отдельной библиотеки для подключения его к различным проектам. При подключении виджета, появляется возможность использовать его в дизайнере.

В то же время в C# создание нового элемента управления отличается в зависимости от используемой технологии создания пользовательского интерфейса и версии .NET.

При создании собственных элементов управления существуют три основных случая:

1. Компоновка уже существующих элементов в один с добавлением новых свойств и методов.
2. Создание нового элемента на основе одного уже существующего, путём расширения его функциональных возможностей или изменения внешнего вида.
3. Создание полностью нового элемента “с нуля”.

В рамках данной работы рассматривается исключительно первый случай, остальные рассматриваются в рамках лекций.

Далее будут кратко рассмотрены особенности создания элементов управления как средствами Qt, так и используя C#. Следует, однако, заметить, что созданные элементы управления могут использоваться только инструментом, аналогичном тому, с помощью которого они были созданы (нельзя применять виджеты Qt в проекте C#...).

### ***Создание собственного виджета средствами Qt***

Создание собственного виджета не сильно отличается от разработки проектов - возможно как использование дизайнера, так и предварительное тестирование в виде отдельного приложения.

Существуют два основных подхода для создания виджета, предполагаемого к использованию в виде библиотеки:

- создание проекта в виде библиотеки с добавлением к нему файла формы или описанием виджета без использования дизайнера;
- создание виджета как отдельного приложения с дальнейшим изменением настроек и структуры проекта для создания библиотеки.

Далее будут рассмотрены оба этих подхода.

### ***Создание виджета как библиотеки***

Для создания виджета по первому из приведённых подходов необходимо при создании проекта указать в качестве типа “библиотека C++” как показано на рисунке 2.1.

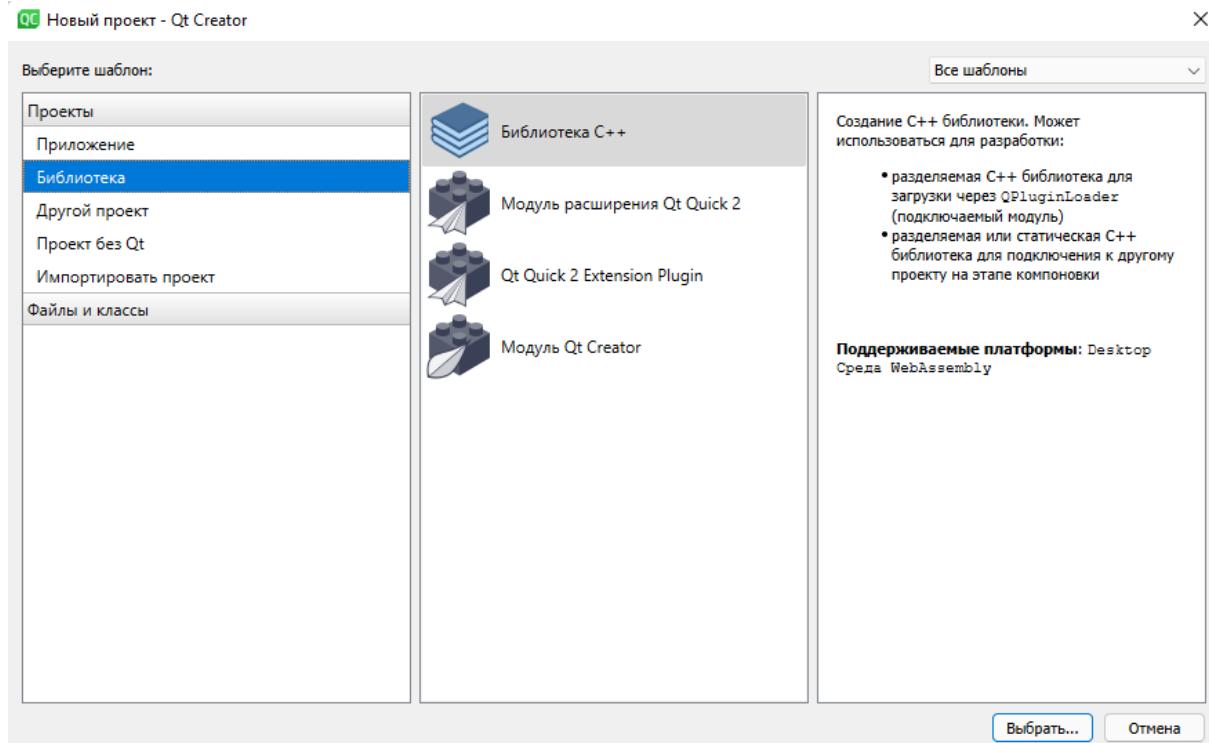


Рис. 2.1. Создание библиотеки C++

Далее при выборе модуля Qt следует указать `gui`. Результатом станет проект с заранее созданным классом. Затем при желании работать с использованием дизайнера следует добавить в проект форму Qt и подключить её к проекту, модифицировав заголовочные и исходные файлы класса и файл проекта следующим образом:

1. В файле проекта необходимо подключить модуль `qt widgets`:  
`QT += gui widgets`

2. В заголовочном файле необходимо подключить пространство имён формы:

```
QT_BEGIN_NAMESPACE  
namespace Ui { class ИМЯВИДЖЕТА; }  
QT_END_NAMESPACE
```

3. Сам класс следует сделать наследником того класса виджета, от которого планируется унаследовать основные свойства, например `QWidget`.

4. Необходимо добавить в описание класса макрос `Q_OBJECT`

5. В качестве поля класса следует добавить объект класса формы дизайнера:

```
private:
```

```
Ui::ИМЯВИДЖЕТА *ui;
```

6. В исходном файле необходимо подключить заголовочный файл, генерируемый дизайнером:

```
#include "ui_имяфайлаформы.h"
```

7. При реализации класса виджета в его конструкторе следует вызвать метод генерации интерфейса:

```
ui->setupUi(this);
```

Вместо данных шагов можно добавить в проект «класс формы Qt», выбрав в качестве шаблона Widget и задав ему то же название что и основному классу библиотеки. При этом следует не забыть подключить к нему файл НАЗВАНИЕ\_КЛАССА\_global.h и добавить макрос из него перед именем класса.

После подобной модернизации следует описать создаваемый класс и добавить необходимую реализацию. В случае корректного исполнения, после компиляции будет создана библиотека, состоящая из двух файлов - \*.a и \*.dll (в случае выполнения в ОС семейства Windows). Данные файлы вместе с заголовочными файлами проекта будут составлять готовый к распространению комплект виджета.

### ***Создание виджета как самостоятельного проекта***

Данный подход позволит протестировать поведение виджета, немного более прост, однако менее надёжен и может приводить к проблемам интеграции виджета в глобальный проект. В этом случае виджет создаётся как обычно, важно лишь чтобы он был наследником корректного класса (следует помнить, что если планируется размещение виджета на форме, то он не может быть прямым или косвенным наследником класса QWindow).

После создания виджета для подготовки его к преобразованию в библиотеку следует удалить файл main.cpp из состава проекта и добавить в файл проекта следующую строчку:

```
TEMPLATE = lib
```

Рекомендуется, но не обязательно также использовать идентификаторы Q\_DECL\_EXPORT и Q\_DECL\_IMPORT перед именем класса, подлежащего

экспорту. Для этого следует использовать следующие директивы препроцессора:

```
#if defined(ИМЯПРОЕКТА_LIBRARY)
# define ИМЯПРОЕКТА_EXPORT Q_DECL_EXPORT
#else
# define ИМЯПРОЕКТА_EXPORT Q_DECL_IMPORT
#endif
```

И объявлять класс примерно следующим образом:

```
class ИМЯПРОЕКТА_EXPORT ИмяКласса
```

В файл проекта при этом следует добавить следующую строчку:  
DEFINES += BUTTONSMASHERASLIB\_LIBRARY

Это позволит явно указывать компилятору что класс подлежит экспорту и последующему импорту и предотвратит ряд проблем.

### ***Подключение созданной библиотеки***

Для подключения созданной библиотеки к проекту следует создать директорию с тремя каталогами: debug, include, release. В каталоги debug и release помещаются файлы \*.a и \*.dll из соответствующей сборки проекта библиотеки. В директорию include помещаются все заголовочные файлы.

Затем в проекте, к которому будет подключена библиотека следует выбрать “добавить библиотеку” как показано на рисунке 2.2.

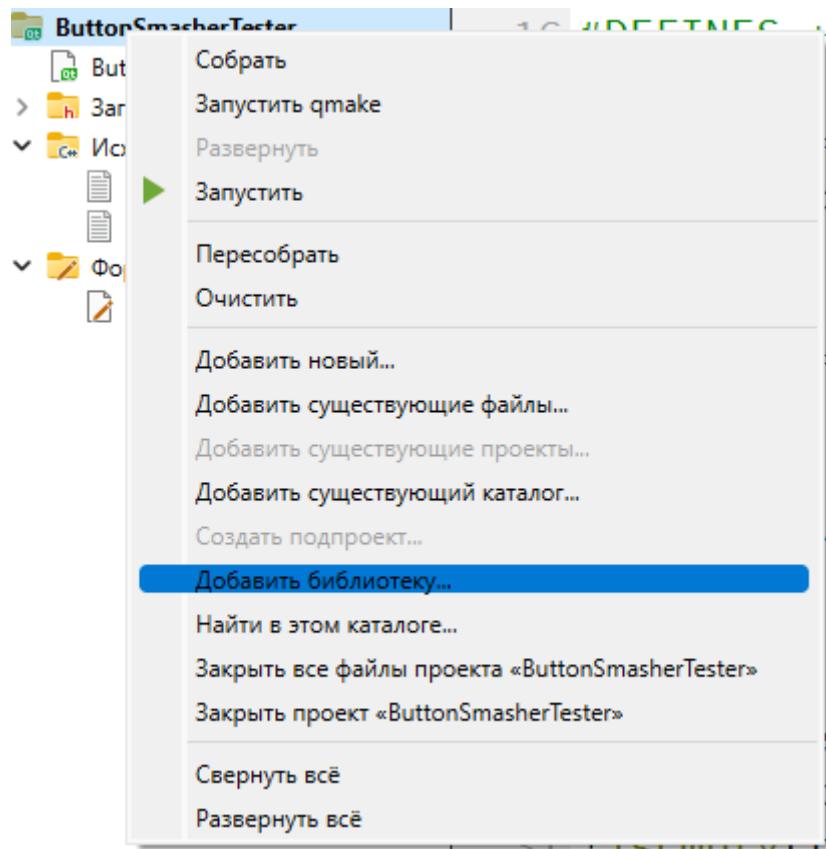


Рис. 2.2. Добавление библиотеки

В появившемся окне следует выбрать “внешняя (external) библиотека”.

Затем необходимо указать путь к файлу \*.a библиотеки и путь к директории include. В результате библиотека будет подключена к проекту и, после подключения заголовочного файла с использованием директивы #include будет готова к использованию.

### ***Использование собственных виджетов в дизайнере***

Созданные виджеты возможно использовать в дополнение к стандартным в Qt Designer, самый простой способ для этого - добавить на форму проекта с подключённой библиотекой виджет, класса, от которого унаследован созданный виджет, либо класса QWidget. После этого следует нажать на него правой кнопкой мыши и выбрать пункт “преобразовать в...” (promote to...). В появившемся окне следует ввести имя виджета и заголовочного файла, как показано на рисунке 2.3.

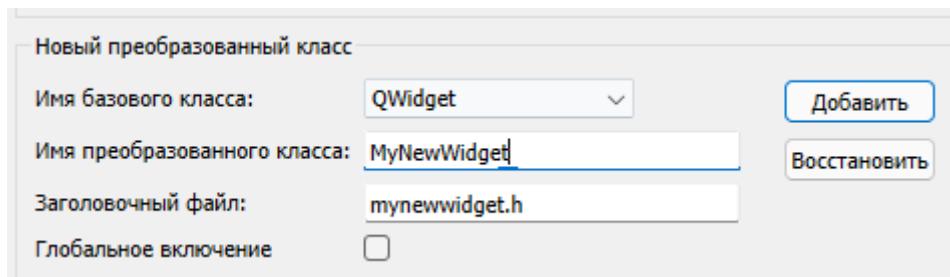


Рис. 2.3. Преобразование виджета

После этого следует нажать кнопку добавить и, выбрав добавленный виджет в списке выше, нажать на кнопку преобразовать.

Преобразованный подобным способом виджет не изменит свой внешний вид в дизайнере, однако будет корректно функционировать с точки зрения взаимодействия с ним.

Существует способ полноценно добавить новый виджет в дизайнер Qt, который предусматривает создание полноценного плагина с описанием его через объект класса QDesignerCustomWidgetInterface, однако это потребует добавление собственного плагина непосредственно в директорию Qt и сопряжено с дополнительными сложностями, а потому не будет рассмотрено в рамках данной работы. Информацию по процессу добавления плагина к дизайнеру можно найти в официальной документации Qt, размещённой по адресу

<https://doc.qt.io/qt-5/designer-creating-custom-widgets.html>.

### ***Создание элемента управления средствами C#***

В рамках данного пособия рассматриваются две технологии создания пользовательского интерфейса для проекта, написанного на C#: Windows Forms и WPF. Обе эти технологии предлагают хоть и схожий, но несколько различающийся в деталях подход к созданию пользовательского элемента управления. При этом созданный элемент (также называемый контрол) может распространяться как dll или как готовый пакет NuGet.

Далее будут рассмотрены особенности обеих технологий.

## *Создание элемента управления Windows Forms*

При создании собственного элемента управления Windows Forms следует учесть отличия между .NET 5-6 и .NET Framework, из-за ряда особенностей в Visual Studio, в первом случае подключение созданного компонента возможно только с использованием пакета NuGet, тогда как во втором случае достаточно подключить библиотеку.

Для создания элемента управления следует выбрать в качестве шаблона проекта Windows Forms Control Library, после чего будет создана заготовка под проект. Для создания библиотеки, после требуемой модификации контрола достаточно собрать (англ. build) решение.

При использовании .NET Framework, создание пакета NuGet потребует ручного редактирования манифеста сборки и выполнение команды nuget pack из командной строки. Подробнее об этом можно почитать в официальном руководстве по следующей ссылке:

<https://docs.microsoft.com/en-us/nuget/quickstart/create-and-publish-a-package-using-visual-studio-net-framework>

При использовании .NET версий 5 и 6, для создания пакета NuGet достаточно указать требуемую версию пакета в настройках проекта и, нажав правой кнопкой мыши на решение, выбрать пункт Pack как показано на рисунке 2.4.

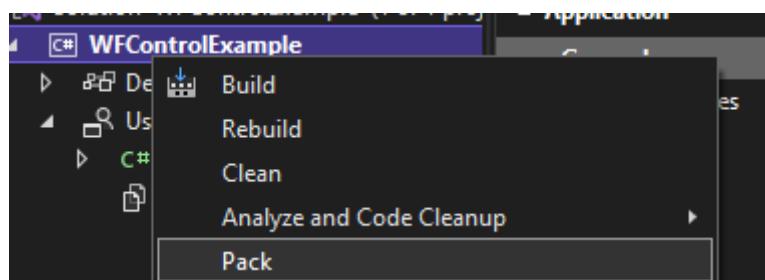


Рис. 2.4. Создание пакета NuGet

Созданный пакет будет находиться в каталоге Release или Debug решения.

При создании элемента средствами Windows Forms может потребоваться описание “обёртки” над обработчиком событий для

переадресации событий от создаваемого элемента его компонентам. Пример такой “обёртки” приведён ниже:

```
public event EventHandler btnShowClick
{
    add { btnShow.Click += value; }
    remove { btnShow.Click -= value; }
}
```

Данная конструкция, по сути, переопределяет операции `+=` (`add`) и `-=` (`remove`) применяемые к событию `btnShowClick` для создания очереди обработчиков. Так как компоненты создаваемого элемента будут скрыты для доступа извне, это позволит обеспечить возможность добавления обработчиков к происходящим с ними событиям.

В дальнейшем при необходимости добавить новый обработчик событию, используется следующая конструкция:

```
userControl11.btnShowClick += ControlUsed;
```

Для добавления созданного элемента из библиотеки следует нажать правой кнопкой мыши на панель элементов и выбрать пункт “choose items”, как показано на рисунке 2.5. В появившемся окне надо указать путь к библиотеке, содержащий элемент. Данный способ работает только в случае использования .NET Framework!

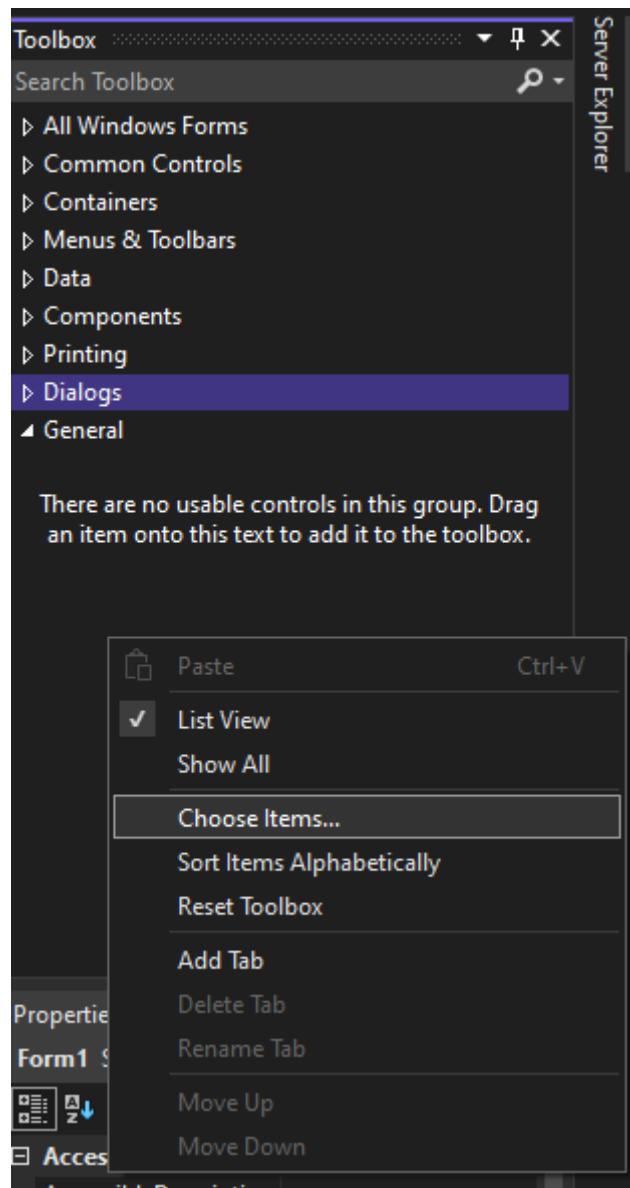


Рис. 2.5. Подключение библиотеки с элементом

Для подключения пакета NuGet необходимо в менеджере пакетов NuGet добавить в качестве источника пакетов каталог, содержащий созданный пакет, как показано на рисунке 2.6.

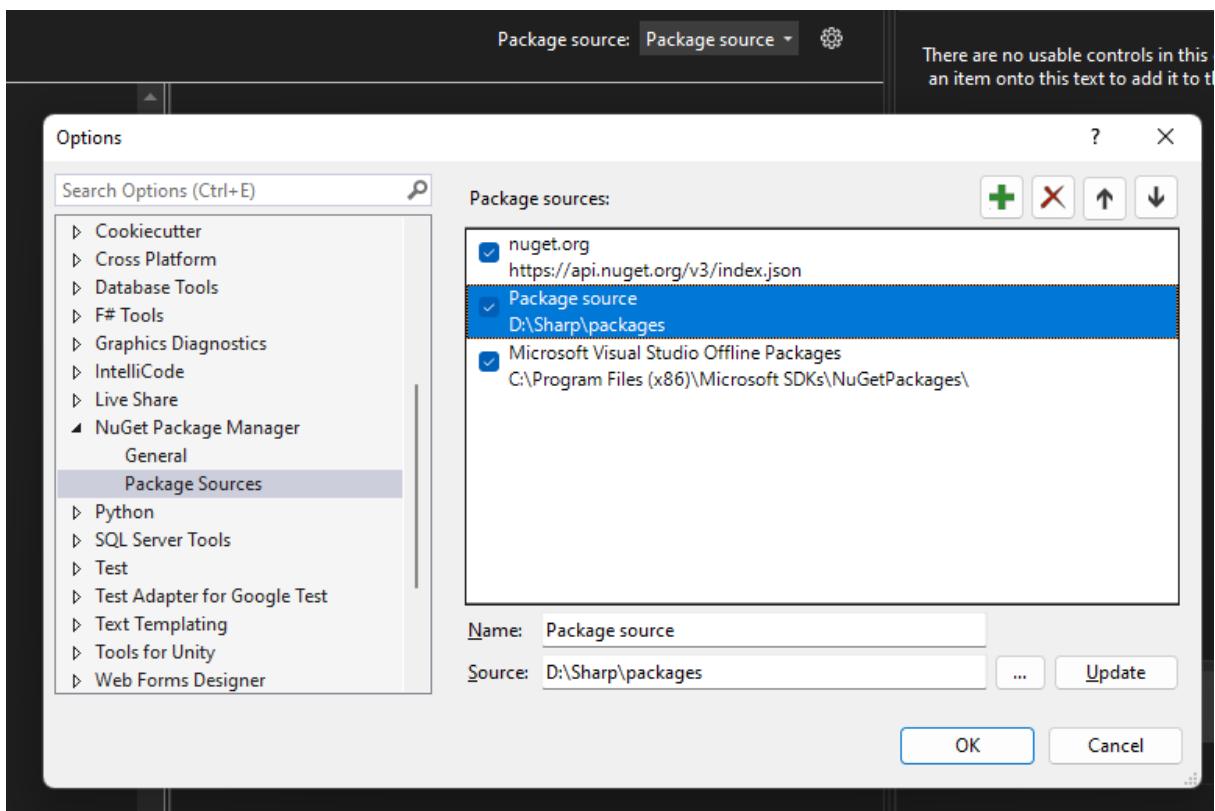


Рис. 2.6. Добавление каталога с пакетами NuGet

После этого необходимо установить пакет, и созданный элемент появится в панели элементов.

### ***Создание элемента управления WPF***

Создание нового элемента управления с использованием технологии WPF хоть и схоже с Windows Forms, но обладает рядом особенностей. Так же, как и в случае с Windows Forms, остаётся проблема подключения созданного элемента - .NET требует использования пакетов NuGet и имеет сложности с добавлением элемента в ToolBox.

При создании элемента требуется выбрать в качестве шаблона проекта WPF User Control Library. Сам элемент создаётся аналогично форме WPF. Важным моментом при этом является большая ориентированность на привязку данных, что позволяет создавать свойства в классе формы и привязывать их к свойствам дочерних элементов управления, а в дальнейшем передавать конкретные значения непосредственно при использовании созданного элемента.

Существенным отличием от Windows Forms является появление RoutedEventHandler для перенаправления обработчика событий от элемента управления к его дочерним элементам или наоборот. Для создания собственного псевдо-события в качестве «обёртки» для события дочернего элемента следует объявить обработчик:

```
public event RoutedEventHandler ClickB;
```

Затем требуется зарегистрировать его:

```
public static readonly RoutedEvent ClickBEvent = EventManager.RegisterRoutedEvent("ClickB",  
    RoutingStrategy.Bubble, typeof(RoutedEventHandler), typeof(UserControl1));
```

Последним шагом является использование его в основном обработчике событий дочернего элемента:

```
private void Button_Click(object sender, RoutedEventArgs e)  
{  
    if (ClickB != null)  
    {  
        ClickB(this, e);  
    }  
}
```

При использовании созданного элемента, новые обработчики можно передавать как в классе окна:

```
UC1.ClickB += Counter;
```

Так и в XAML:

```
<uc:UserControl1 x:Name="UC1" ClickB="Counter"/>
```

Подключение созданного элемента в случае использования .NET и пакета NuGet отличается от подключения элемента Windows Forms. В случае WPF после пакета NuGet необходимо подключить его пространство имён в файле XAML:

```
xmlns:uc="clr-namespace:WPFControlExample;assembly=WPFControlExample"
```

После этого можно добавлять созданный элемент, указывая при этом необходимые значения полей:

```
<uc:UserControl1 x:Name="UC1" LabelText="Тест" ClickB="Counter" />
```

Однако добавление элемента в ToolBox, как правило не происходит.

При использовании .NET Framework и библиотеки с элементом, добавление элемента происходит аналогично Windows Forms, подключение

пространства имён при этом будет выполнено автоматически при размещении элемента на форме и пересборке решения.

## **Общая постановка задачи**

В данной работе необходимо в соответствии с вариантом написать собственный элемент управления, выполненный в виде динамической библиотеки (в случае использования Qt или Windows Forms) или пакета NuGet. А затем написать небольшую программу, демонстрирующую использование данного элемента управления. Важно реализовать взаимодействие с созданным элементом управления из тестовой программы - элемент управления должен передавать в основную программу данные и принимать из неё команды либо в формате слотов и сигналов, либо через вызовы методов и подключение обработчика событий.

Созданный элемент управления должен быть подключена к тестовой программе. Должна быть продемонстрирована возможность работы с ним с использованием дизайнера для создания пользовательских интерфейсов.

Формулировки вариантов используют термины из фреймворка Qt. При выполнении работы средствами языка C#, требуется найти аналогичные элементы управления из числа стандартных для создания собственного композитного элемента управления. Вместо испускания сигнала потребуется добавить возможность назначения внешнего по отношению к создаваемому контролю обработчика события, который сможет получать указанное в варианте свойство элемента.

## **Варианты заданий.**

### ***Вариант 1***

Палитра для выбора цвета.

Виджет представляет собой несколько QPushButton (хотя бы 5), различных цветов. При щелчке на одну из них, она выделяется (любым способом). Реализованы методы, позволяющие получить код выбранного

цвета и задать цвет одной из QPushButton. При выборе цвета, испускается сигнал, передающий цвет.

### ***Вариант 2***

Слайдеры для задания цвета.

Виджет представляет из себя три слайдера и QLabel (возможно ещё добавление полей для ввода цифр). Слайдеры изменяются от 0 до 255. При движении слайдеров, цвет QLabel меняется (каждый из слайдеров задаёт компоненту системы RGB). Щелчок на QLabel испускает сигнал с выбранным цветом. Также должен быть реализован метод, получающий выбранный цвет и метод, устанавливающий слайдеры в соответствии с переданным цветом.

### ***Вариант 3***

Кнопка с подтверждением.

Виджет представляет из себя кнопку и QCheckBox. При нажатии на кнопку проверяется состояние QCheckBox - если флажок установлен, то испускается сигнал что всё в порядке. Если флажок не установлен, то отображается заданное сообщение. Необходимо реализовать метод, который позволяет передавать сообщение как в виде простого текста, так и в виде заранее подготовленного QMessageBox.

### ***Вариант 4***

Текстовое поле с автозаполнением.

Виджет представляет из себя текстовое поле с подключённым автозаполнением (QCompleter). Список слов для автозаполнителя может быть передан и получен как QStringList с помощью реализованных методов виджета. Также к виджету могут быть подключены несколько разных списков слов и реализовано переключение между ними с помощью методов.

### ***Вариант 5***

Индикатор заряда батареи.

Виджет представляет из себя QLabel и каким-либо образом реализованную картинку. QLabel отображает передаваемый виджету процент заряда батареи. Изображение меняется в зависимости от заряда. Передача заряда реализована через слот виджета, который принимает число и соответствующим образом меняет изображение и подпись. Также должны быть реализованы различные настройки отображения виджета, изменяемые методом. Данный метод позволяет выбрать один из заранее предустановленных вариантов отображения на выбор разработчика.

### ***Вариант 6***

Группа кнопок

Набор из нескольких (минимум 5) кнопок, расположенных либо горизонтально, либо вертикально. Расположение кнопок можно изменить с помощью метода виджета. Возможно изменение количества кнопок с помощью метода. Возможно получение указателя на определённую кнопку по её индексу. Внутреннее хранение кнопок остаётся на выбор разработчика.

### ***Вариант 7***

Спидометр

Виджет представляет из себя QLabel с текущим показателем скорости и графическим отображением. Передача скорости должна быть реализована через слот виджета, который принимает число и соответствующим образом меняет изображение и подпись. Также следует предусмотреть различные виды графического отображения спидометра и их переключении, вызовом метода.

### ***Вариант 8***

Виджет логин-пароль

Виджет представляет из себя набор текстовых полей для ввода логина и пароля, подписи к ним, QCheckBox скрыть/показать пароль и кнопки ввод и отмена. Нажатие на кнопку ввода проверяет о наличии значений в полях и при успехе, испускает сигнал с парой логин-пароль (пароль должен быть преобразован по выбранному разработчиком алгоритму). В случае неудачи отображается окно с предупреждением, указывающее какое поле не заполнено. Поле для ввода пароля должно скрывать ввод звёздочками и отображать при условии снятого или поставленного флажка. При нажатии на кнопку отмена поля очищаются и испускается сигнал об отмене.

### ***Вариант 9***

Виджет анкеты

Виджет представляет из себя набор текстовых полей (не менее пяти) для ввода данных, подписи к ним и кнопки ввод и отмена. Нажатие на кнопку ввода проверяет о наличии значений в полях и при успехе, испускает сигнал с QStringList, содержащем информацию о всех полях. В случае неудачи отображается окно с предупреждением, указывающее какое поле не заполнено. При нажатии на кнопку отмена поля очищаются и испускается сигнал об отмене.

### ***Вариант 10***

Виджет конструктор анкет

Виджет представляет из себя текстовое поле, подпись к нему и кнопку ввод. Нажатие на кнопку ввода проверяет о наличии значений в полях и при успехе, испускает сигнал о успехе. Необходимо организовать метод, добавляющий новое текстовое поле и подпись к нему. А также метод, позволяющий получить информацию как из всех полей, так и из конкретного. Метод хранения полей оставлен на усмотрение разработчика.

### ***Вариант 11***

#### D-pad

Виджет представляет из себя 8 кнопок, расположенных квадратом с изображёнными на них стрелками. Нажатие на кнопку испускает сигнал от виджета с указанием направления. Также кнопки нажимаются при нажатии соответствующих кнопок на клавиатуре.

### ***Вариант 12***

#### Графическая панель управления

Виджет представляет из себя 9 QPushButton, расположенных квадратом с различными изображениями. Нажатие на QPushButton испускает сигнал от виджета с указанием id и состояния и меняет изображение на другое. Также должен быть реализован метод, позволяющий получить состояние всех QPushButton в данный момент времени.

### ***Вариант 13***

#### Графическое отображение джойстика

Виджет представляет из себя два изображения - заполненный круг и окружность, представленные на рисунке 2.7, а также два QLabel A и B. У виджета должен быть реализован слот, принимающий четыре параметра - угол, радиус и состояние двух кнопок, который смещает заполненный круг на указанное расстояние относительно центра, а также меняет цвет QLabel в соответствии с состояниями. Также должен быть слот, сбрасывающий положение заполненного круга и метод, возвращающий текущее положение и состояние кнопок.

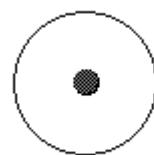


Рис. 2.7. Графическое отображение джойстика

### ***Вариант 14***

#### Группа выключателей

Виджет представляет из себя группу QRadioButton и QLabel. В QLabel отображается номер выбранной в данный момент кнопки или 0 если не выбрана ни одна. Нажатие на кнопку испускает сигнал с id кнопки и меняет номер в QLabel, также реализован слот, который устанавливает состояние виджета в соответствии с переданным ему индексом. Также необходимо реализовать метод, который позволяет добавить новую кнопку, с указанием подписи к ней.

### ***Вариант 15***

#### Улучшенный Text Edit

Виджет представляет из себя группу QTextEdit, QSlider с подписью или QSpinBox и раскрывающийся список шрифтов. Слайдер регулирует кегль, а список позволяет выбрать шрифт для редактора. Также должна быть возможность передать шрифт и кегль в виджет извне, предпочтительнее через соответствующий слот.

### ***Вариант 16***

#### Поле для игры в 15

Виджет представляет из себя 16 QLabel, расположенных квадратом 4x4. Все кроме правого нижнего пронумерованы и белого цвета. Правый нижний без цифры и чёрного цвета. Должен быть реализован метод или слот, который меняет местами чёрный QLabel с одним из соседних (направление передаётся индексом). И метод, который возвращает матрицу, отображающую текущее состояние на поле с 0 на месте чёрного квадрата.

### ***Вариант 17***

#### Поле для игры в шахматы

Виджет представляет из себя 64 QPushButton, расположенных квадратом 8x8 чередующихся белого и чёрного цвета. Нажатие на QPushButton испускает сигнал с его координатами. Должен быть реализован слот или метод, который получает изображение и номер QPushButton и рисует на указанном QPushButton переданное изображение. Также должен быть реализован слот или метод, который принимает номер QPushButton и удаляет нарисованное на нём.

### ***Вариант 18***

Электронные часы

Виджет представляет из себя электронные часы, отображающие время с набором кнопок:

1. Первая переводит часы в режим секундомера, который испускает сигнал каждую секунду
2. Вторая переводит часы в режим таймера, в котором можно задать время, через которое часы издают сигнал
3. Третья возвращает часы в стандартный вид

Также возможно переключение часов в один из режимов, передачей сигнала. И получение текущего времени.

### ***Вариант 19***

Виджет для отображения плиточного панно

Виджет представляет из себя набор QPushButton, в виде прямоугольника NxM, размеры которого задаются пользователем. Изначально все QPushButton белые. Нажатие левой кнопкой по QPushButton возвращает его номер. Щелчок правой - окрашивает его в белый цвет. У виджета реализован метод либо слот с двумя параметрами - изображение и id, который устанавливает переданное изображение в указанный QPushButton. Также должен быть реализован слот сброса, который приводит всё к изначальному виду.

### ***Вариант 20***

Простой видеоплеер

Виджет представляет из себя видеоплеер с окном для воспроизведения, кнопками играть, пауза и стоп. Видео передаётся в плеер с помощью метода. Также должны быть реализованы слоты, доступные извне с функциями, аналогичными кнопкам.

### ***Вариант 21***

Простой просмотрщик изображений

Виджет представляет из себя просмотрщик изображений со слайдером приближения и кнопками открыть и “сохранить как”. Изображение либо открывается нажатием на кнопку, либо передаётся извне вызовом метода. Слайдер увеличивает или уменьшает изображение в диапазоне 1-300%. Слайдером можно также управлять вызовом слота.

### ***Вариант 22***

Треугольник Максвелла для выбора цвета

Виджет представляет из себя изображение треугольника Максвелла и три QLabel. Нажатие на треугольник испускает сигнал с выбранным цветом и устанавливает его составляющие в QLabel.

### ***Вариант 23***

Настраиваемая полоса прогресса

Виджет представляет из себя сильно настраиваемую полосу прогресса - можно задать один из предустановленных внешних видов (полукруг, круг, прямая и т.п.), диапазон значений, единицы измерения. При достижении максимума или минимума виджет издаёт сигнал. Положение полосы прогресса устанавливается слотом.

## ***Вариант 24***

Контейнер для виджетов

Виджет представляет из себя область с полосой прокрутки (возможно реализуемой слайдером, а может стандартной). Для виджета реализованы методы, которые позволяют разместить в области другие виджеты, любого типа. Размещенные виджеты добавляются в компоновщик (горизонтальный или вертикальный в зависимости от настроек). Также возможно получить список всех размещенных виджетов, чтобы управлять каждым из них.

### **Контрольные вопросы**

Раскройте содержание следующих тем по выполненной работе (выбор темы зависит от использованного средства разработки):

#### ***В случае использования Qt***

1. Создание DLL с помощью Qt.
2. Создание элемента управления с помощью Qt.
3. Подключение элемента управления в проект Qt.
4. Добавление свойств элементам управления.
5. Создание новых сигналов в Qt.
6. Приведение типов элементов управления.
7. Получение имени класса виджета средствами Qt.
8. Использование иерархии виджетов Qt (методы, связанные с понятиями child и parent).
9. Работа с событиями.
10. Отличия виджетов от элементов QML.

#### ***В случае использования C#***

1. Создание DLL с помощью C#.
2. Пакеты NuGet.
3. Создание элемента управления с помощью C#.
4. Подключение элемента управления в проект C#.
5. Добавление свойств элементам управления.

6. Перенаправление обработчика события в C#.
7. Приведение типов элементов управления.
8. Получение имени класса элемента управления в C#.
9. Работа с событиями.
10. Различия элементов управления, созданных средствами Windows Forms и WPF.

## ПРАКТИЧЕСКАЯ РАБОТА 3

### «Работа с файлами XML/JSON»

#### Теоретические сведения

В процессе работы приложениям часто требуется взаимодействовать с внешними файлами для сохранения или загрузки информации. Разновидностью такого процесса является сериализация и десериализация. Общая идея сериализации - сохранение состояния объекта на внешнее, по отношению к программе хранилище. Десериализацией же называется процесс восстановление объекта к сохранённому состоянию.

Одним из способов сериализации и десериализации является сохранение состояний объектов в структурированный файл. Наиболее популярны при этом форматы XML и JSON. Другой популярный формат INI как правило используется только для хранения настроек, поэтому рассмотрен не будет.

При выполнении задания, обучающийся может сам выбрать как формат хранения данных (XML или JSON), так и средства разработки (Qt или C#), поэтому далее будет приведена краткая информация как по каждому из форматов, так и по принципам работы с ним.

#### *Краткое описание формата XML*

Формат XML (англ. eXtensible Markup Language) — расширяемый язык разметки) представляет собой иерархию сущностей (или элементов), каждая из которых описывается следующими понятиями:

- теги - записываются в угловых скобках, обозначают начало, конец и тип сущности;
- значение - записывается между открывающим и закрывающим тегами, может представлять собой как некий текст, так и вложенную сущность;
- атрибуты - свойства, описывающие сущность, описываются внутри тега в формате название=»значение».

Существуют также другие элементы XML, однако в рамках данной работы они не будут использованы.

Структура файла XML должна представлять собой жёсткую иерархию с единственным корневым элементом, все теги должны идти парами (открывающий и закрывающий), за исключением “пустых” тегов.

### *Краткое описание формата JSON*

Формат JSON (англ. JavaScript Object Notation) представляет собой упорядоченный набор значений или связок ключ: значение.

В качестве значений в JSON могут выступать:

- запись или JSON объект — это неупорядоченное множество пар ключ:значение, заключённое в фигурные скобки «{ }». Ключ описывается строкой, между ним и значением стоит символ «:». Пары ключ-значение отделяются друг от друга запятыми.

- массив (одномерный) — это упорядоченное множество значений.

Массив заключается в квадратные скобки «[ ]». Значения разделяются запятыми. Массив может быть пустым, то есть не содержать ни одного значения. Значения в пределах одного массива могут иметь разный тип.

- число (целое или вещественное).
- литералы true (логическое значение «истина»), false (логическое значение «ложь») и null.
- строка — это упорядоченное множество из нуля или более символов юникода, заключённое в двойные кавычки.

Благодаря этой структуре, JSON файл может быть легко преобразован в массив объектов и обратно, при этом ключами будут являться имена свойств объекта, а значениями - значения этих свойств.

Также JSON может быть преобразован в массив коллекций типа map или dictionary.

Важно, что из-за своей структуры, весь файл JSON, по сути, представляет собой одно значение, в качестве которого обычно выступает массив из объектов.

## ***Работа с файлами формата XML средствами Qt***

Для работы с данными, хранящимися во внешних файлах средствами Qt необходимо сначала открыть файл, создав объект класса QFile, передав в него путь к файлу и вызвав метод open() с параметром, определяющим режим доступа к файлу.

Для упрощения работы с файлом формата XML, в Qt существуют классы QXmlStreamReader и QXmlStreamWriter, которые предоставляют интерфейс для навигации по такому файлу и работе с ним. Объекты данных классов должны быть связаны с, предварительно открытым, объектом класса QFile.

Для чтения данных из файла формата XML, используется класс QXmlStreamReader, который предоставляет набор методов, основные из которых представлены в таблице 1.

Таблица 1.

### **Методы класса QXmlStreamReader**

| Метод                  | Описание  |
|------------------------|---|
| setDevice(QFile)       | Связывает объект с файлом XML   |
| readNextStartElement() | Переходит к началу следующей сущности (в том числе дочерней по отношению к текущей) в файле и устанавливает тег её начала в качестве текущего |
| readNext()             | Переходит к следующему тегу в файле и устанавливает его в качестве текущего   |
| name()                 | Возвращает строку, содержащую имя текущего тега   |
| readElementText()      | Возвращает текст текущей сущности если текущий тег - открывающий  |
| attributes()           | Возвращает коллекцию атрибутов текущего тега, если он - открывающий   |
| skipCurrentElement()   | Переходит к закрывающему тегу для текущего, пропуская все вложенные сущности  |
| atEnd()                | Возвращает истину при достижении конца файла  |

Атрибуты тега представляют собой объекты класса QXmlAttribute и содержат методы name() и value() для получения имени и значения атрибута

соответственно. Значения считывают в формате QVariant и должны быть далее преобразованы в требуемый тип.

Для записи данных в файл формата XML используется класс QXmlStreamWriter, работа с которым схожа - так же необходимо изначально связать объект данного класса с файлом XML. Основные методы класса приведены в таблице 2.

Таблица 2.

### Методы класса QXmlStreamWriter

| Метод                      | Описание   |
|----------------------------|--|
| setDevice(QFile)           | Связывает объект с файлом XML                                      |
| writeStartDocument()       | Записывает служебные данные, обозначающие начало файла             |
| writeStartElement(QString) | Записывает начальный тег сущности с указанием имени тега           |
| writeAttribute(name,value) | Добавляет в текущий тег атрибут с указанным именем и значением     |
| writeCharacters(QString)   | Записывает указанный текст в качестве содержимого текущей сущности |
| writeEndElement()          | Закрывает текущий тег  |
| writeEndDocument()         | Закрывает текущий документ   |

После завершения работы с файлом его необходимо закрыть, вызвав метод close() класса QFile.

### *Работа с файлами формата JSON средствами Qt*

При работе с файлами формата JSON также используется объект класса QFile, который связывается с непосредственно файлом на жёстком диске.

Для чтения данных изначально весь текст из файла должен быть считан в объект класса QByteArray, который в дальнейшем будет преобразован в документ JSON с помощью метода QJsonDocument::fromJson(массив).

Полученный документ может быть при необходимости преобразован в массив JSON объектов с помощью метода array() или в один объект с помощью метода object().

Объект JSON представляет собой объект класса `QJsonObject`, обращение к конкретному значению, в котором осуществляется с помощью ключа, аналогично работе со словарём:

```
obj[“ключ”]
```

Значение же является объектом класса `QJsonValue` и может быть преобразовано как в один из стандартных типов данных, так и в массив или объект JSON.

Массив JSON представляет собой коллекцию объектов JSON и может быть перебран с помощью цикла `foreach` или напрямую.

Запись в файл формата JSON использует те же классы и сводится к формированию `QJsonDocument`, преобразованию его в строку с помощью метода `toJson()` и записью полученной строки в файл методом `QFile::write(QString)`.

Формирование объекта JSON происходит аналогично созданию словаря - для указанных ключей задаются значения, при этом добавление новых ключей происходит автоматически.

Также возможно создание объекта JSON напрямую из словаря методом `QJsonObject::fromVariantMap(QVariantMap)`

Массив JSON формируется из объектов JSON методом `QJsonArray::append(QJsonObject)`

Документ JSON формируется из массива или объекта JSON непосредственно передачей объекта требуемого класса в конструктор объекта класса `QJsonDocument`.

### ***Работа с файлами формата XML средствами C#***

Для работы с файлами формата XML в C# существует пространство имен `System.Xml`. Основные классы из данного пространства имён - `XmlReader` и `XmlWriter`, которые предоставляют базовые функции для работы с файлами XML. Также существует возможность работы с использованием несколько более затратного по ресурсам класса  `XmlDocument`, который, однако, предоставляет более удобный интерфейс для работы с файлом. В

рамках данного пособия будет рассмотрен только вариант работы с использованием XmlDocument.

После создания объекта данного класса, можно загрузить файл XML с помощью метода LoadXml, который принимает в качестве параметра строку, путь к файлу или ранее загруженный в XmlReader файл XML.

Основным элементом класса XmlDocument является свойство DocumentElement, которое содержит корневую сущность файла в виде объекта класса XmlNode. Данный класс содержит следующие свойства, представленные в таблице 3.

Таблица 3.

#### Свойства класса XmlNode

| Свойство   | Описание   |
|------------|--|
| Attributes | Коллекция, содержащая все атрибуты сущности, получить значение конкретного атрибута можно используя следующую конструкцию:<br>Attributes[“Название”].Value |
| ChildNodes | Коллекция, содержащая дочерние по отношению к данной сущности  |
| InnerText  | Строка, содержащая текст сущности, игнорирующая вложенные сущности   |
| Name       | Строка, содержащая имя сущности  |

Для добавления новой сущности следует создать новый объект класса XmlNode, задать его свойства и добавить в список дочерних сущностей методом AppendChild.

Аналогично атрибут добавляется к сущности путём создания объекта класса XmlAttribute с указанным именем, задания его значения и добавления его в коллекцию Attributes методом Attributes.Append(XmlAttribute)

Для сохранения файла XML достаточно вызвать метод save(“путь”) объекта класса XmlDocument.

## ***Работа с файлами формата JSON средствами C#***

Работа с форматом JSON средствами языка C# заключается в описании класса, хранящего свойства объекта JSON и десериализации файла JSON в объект или коллекцию объектов данного класса.

Требуемые инструменты расположены в пространстве имён System.Text.Json.

Для чтения информации из файла JSON следует сначала считать всё содержимое файла в строку:

```
string jsonString = File.ReadAllText(path);
```

Затем эта строка передаётся в обобщённый статический метод Deserialize<> класса JsonSerializer с указанием требуемого типа или коллекции:

```
collection = JsonSerializer.Deserialize<List<UserType>>(jsonString);
```

Данный метод вернёт десериализованную коллекцию.

Для сериализации используется статический метод Serialize класса JsonSerializer, который возвращает строку, пригодную для записи в файл:

```
string jsonString = JsonSerializer.Serialize(collection);
```

Запись в файл осуществляется стандартными методами C#:

```
File.WriteAllText(docPath, jsonString);
```

## **Общая постановка задачи**

Необходимо разработать приложение с графическим пользовательским интерфейсом согласно индивидуальному варианту. Для разработки можно использовать фреймворк Qt или язык программирования C# совместно с технологиями Windows Forms или WPF. По согласованию с преподавателем возможно использование фреймворка Xamarin или технологии MAUI.

В ходе работы приложение должно хранить свои данные в файле формата XML или JSON и обращаться к ним по мере необходимости.

## **Варианты заданий.**

### ***Вариант 1***

Аудиоплеер со списками воспроизведения. Список воспроизведения должен сохраняться в формате XML/JSON, хранить для каждой композиции её название и путь к файлу. Также должен сохраняться прогресс в рамках списка воспроизведения.

### ***Вариант 2***

Программа для запуска приложений. Позволяет запускать различные приложение. Каждое представлено кнопкой или QLabel с иконкой и названием рядом. Нажатие на кнопку или QLabel запускает приложение. В файле хранятся путь к исполняемому файлу приложения, название и иконка. Должна быть возможность добавления и удаления приложений из файла.

### ***Вариант 3***

Программа для тестирования. В файле хранятся вопросы и варианты ответов с отмеченным верным. Программа отображает вопросы по одному и собирает статистику прохождения (хранится в отдельном файле). Должна быть возможность добавления и удаления вопросов и просмотра статистики.

### ***Вариант 4***

Мозаичное панно. На форме находится квадрат 5x5 из QPushButton. Нажатие на каждую позволяет выбрать изображение для неё. В файле хранятся изображения назначенные каждому из QPushButton. Должна быть возможность сохранять и загружать файлы с полученными панно.

### ***Вариант 5***

Программа для чтения текстовых файлов. В файле хранятся стандартные настройки отображения текста (шрифт, кегль, начертание), а также настройки

для каждого из ранее открытых файлов (отличия от стандартных плюс прогресс чтения).

### ***Вариант 6***

Простой дизайнер формы. На форме могут быть один компоновщик (тип указывается в свойствах), QPushButton, QLabel, QLineEdit. В основной форме выбирается тип компоновщика и добавляются в него поэлементно произвольное количество виджетов указанных типов. Для каждого виджета указывается текст на нём. Данные сохраняются в файл, на основе которого после этого строится форма.

### ***Вариант 7***

Работа с анкетами. В файле хранятся заголовки анкет и вопросы к ним. На форме представлен список анкет, выбирая любую из них можно либо перейти в режим редактирования, либо пройти опрос. Результаты опроса сохраняются в отдельном файле.

### ***Вариант 8***

Палитра. В файле хранятся 16 цветов, заданных в системе RGB. На форме представлены 16 QPushButton соответствующих цветов. При нажатии на QPushButton появляется окно выбора цвета. Обязательна реализация возможности выбора файла с палитрой.

### ***Вариант 9***

Часы. На форме представлены текстовые поля, являющиеся многофункциональными часами. Каждое поле может работать в одном из следующих режимов: демонстрация времени (с учётом часового пояса), обратный отсчёт, секундомер, будильник. Режим и настройки для каждого поля хранятся в файле. Должна быть возможность добавления полей.

### ***Вариант 10***

Список студентов для людей с аллергией на БД. В файле хранятся студенты, поделённые на группы, поделённые по направлениям подготовки. Для каждого студента хранится его ФИО и номер зачётки. На форме представлены три QListWidget и текстовое поле. Первый хранит список направлений подготовки. После выбора направления, во втором появляются группы, относящиеся к нему. После выбора группы, в третьем списке появляются ФИО студентов из этой группы. Выбор студента приводит к отображению в текстовом поле его номера зачётки. Также должна быть реализована возможность добавления нового студента, группы и направления подготовки

### ***Вариант 11***

Ежедневник. В файле хранятся дела, поделённые по датам. Для каждой даты может быть несколько дел. Если на дату не назначено дел, информация о ней не хранится. Для каждого дела хранится заголовок, текст, время начала и длительность. Созданная программа позволяет выбрать дату и посмотреть дела, назначенные на неё (если такие есть), а также редактировать список дел.

### ***Вариант 12***

Список покупок. В файле хранятся покупки, поделённые по датам. Для каждой покупки хранится цена, количество, название. Приложение позволяет просматривать покупки за прошедшие дни (но не изменять), составлять список покупок (без указания цены), отмечать покупки как совершенные (указывая цену), и подсчитывать траты за день.

### ***Вариант 13***

Конфигурация компьютера. В одном файле хранятся комплектующие, разбитые на категории (хотя бы пять категорий), для каждой хранится цена. На форме представлены раскрывающиеся списки комплектующих для каждой категории. Пользователь выбирает комплектующие, а также задаёт имя для конфигурации. Отображается общая цена и предлагается сохранить конфигурацию в отдельный файл. Также необходима возможность добавления комплектующих и загрузки ранее сохранённых конфигураций.

### ***Вариант 14***

Меню ресторана. В файле хранятся блюда, разбитые на категории. Для каждого хранится цена и описание. На форме отображаются кнопки, с названиями категорий, каждая из которых открывает форму с блюдами из неё. Блюда можно заказать и увидеть итоговый заказ и стоимость, а также сохранить заказ в отдельный файл.

### ***Вариант 15***

Текстовый план выставки. В файле хранятся экспонаты, разбитые по залам и этажам. Для каждого экспоната хранится название и описание, а также год создания и автор. В форме выводится информация о экспонатах, с возможностью выбора конкретного этажа и зала.

### ***Вариант 16***

Веб браузер. Используя QWebEngine создать простой многовкладочный веб браузер с поддержкой избранного и истории (хранятся в отдельных файлах). Также должна быть возможность добавить вкладку и восстановить последний сеанс.

### **Вариант 17**

Каталог музыки. В файле хранятся композиции, разбитые по авторам и альбомам. Для каждой указана информация - название, год выпуска, жанр. В форме можно выбрать автора, альбом и получить список композиций или осуществить поиск по неполному названию композиции.

### **Вариант 18**

Преподаватели. В файле хранятся информация о преподавателях, такая как ФИО, путь к файлу с фото, кафедра, должность. На форме выводится информация о преподавателях выбранной кафедры. Также должна быть возможность поиска преподавателя по фамилии. Примеры фото приведены на рисунке 3.1.



Рис. 3.1. Примеры фотографий преподавателей кафедры

### **Вариант 19**

Инвентаризация склада. В файле хранятся предметы, разбитые по помещениям и полкам. Для каждого хранится инвентарный номер, название,

описание, масса, размеры. На основной форме расположены кнопки, соответствующие помещениям, и кнопка поиска. Нажатие кнопки с помещением открывает немодальную форму, относящуюся к данному помещению с информацией о предметах в нём. Нажатие кнопки поиск выводит информацию о предмете по его инвентарному номеру, включая расположение предмета.

### ***Вариант 20***

Статистика борцов. В файле хранятся борцы. Для каждого приведены ФИО, рост, вес, возраст, процент побед. В форме можно выбрать бойца, с помощью поиска по фамилии, посмотреть его параметры. Также можно выбрать двух борцов и, если они в одной весовой категории, посмотреть вероятный исход поединка, вычисляемый на основе параметров бойца. Точная формула оставлена на усмотрение разработчика.

### ***Вариант 21***

База карт Magic: The Gathering. В файле хранятся карты одного сета. Для каждой хранится название, тип и в зависимости от него цена, цвет, здоровье/атака (если существо), особенности (если planeswalker), текст и т.п. В программе выводится информация о картах из сета с возможностью фильтрации списка карт.

### ***Вариант 22***

Класс “Строка” наносит ответный удар. В программе описаны класс “Строка” и два унаследованных от него - “двоичная строка” (допускает только символы 0 и 1) и “десятичная строка” (допускает знак минус в начале, цифры и ровно одну точку). Для каждого класса определены не менее чем три различных метода. В файле хранится список объектов. Для каждого указан тип и начальное значение, а также приведены методы, которые будут вызваны (любая комбинация из методов, задаваемая пользователем). Программа

загружает файл, создаёт объекты указанных типов с указанным значением и выводит результаты выполняемых операций вместе со значениями объектов в текстовое поле или список.

### ***Вариант 23***

Помощник огородника. В одном файле хранится информация об овощах - название, время посева, время сбора урожая, особенности ухаживания и т.п. В другом файле хранится перечень грядок, для каждой указано что там засеяно, когда, было ли удобрено и чем, дата последнего полива и т.п. В форме всё это выводится в удобном виде - можно посмотреть перечень грядок, выбрать нужную и посмотреть особенности засеянного, указать дату полива и т.п.

### ***Вариант 24***

Справочник по грызунам. В файле хранится информация по различным представителям отряда грызунов с разбиением по семействам. О каждом хранится описание, ареал обитания, путь к изображению, максимальный возраст, вес. Программа предоставляет возможность просмотреть информацию из файла с использованием QTreeWidget. А также поиск по названию. В файле должна быть калибара.

### **Контрольные вопросы**

1. Структура файла XML.
2. Структура файла JSON.
3. Сериализация и десериализация.
4. Работа с файлами XML.
5. Работа с файлами JSON.
6. Работа со словарями в Qt или C#.
7. Коллекции и работа с ними.
8. Работа с QListBox и QComboBox или их аналогами в C#.
9. Работа с многооконным приложением.

10. Запуск внешних приложений средствами Qt или C#

.

## **ПРАКТИЧЕСКАЯ РАБОТА 4**

### **«Использование шаблонов проектирования при разработке программ»**

#### **Теоретические сведения**

При разработке сложного программного обеспечения, взаимодействующего с разнородными внешними данными, следует использовать архитектурные шаблоны, обеспечивающие независимость обработки данных от их отображения, тем самым снижая сложность разработки. Также использование таких шаблонов позволяет применять готовые изолированные компоненты, что обеспечивает реализацию компонентного подхода, являющегося одной из основной парадигм визуального программирования.

Как фреймворк Qt, так и технология WPF предлагают реализации подобных архитектур, которые и должны быть использованы при выполнении данной работы.

#### ***Архитектурный шаблон MVC***

Одним из самых простых архитектурных шаблонов является MVC, который подразумевает деление программы на три основных модуля:

1. Модель (англ. Model) предоставляет данные и реагирует на команды контроллера, изменяя своё состояние
2. Представление (англ. View) отвечает за отображение данных модели пользователю, реагируя на изменения модели
3. Контроллер (англ. Controller) интерпретирует действия пользователя, оповещая модель о необходимости изменений

Каждый из модулей является независимым, что позволяет разрабатывать и тестировать его вне зависимости от остальных модулей. По сути, каждый модуль является инкапсулированной сущностью, предоставляющей программный интерфейс для взаимодействия с нею.

Это позволяет с одной стороны использовать один модуль в различных проектах, а с другой относительно легко заменять модули в проекте в случае необходимости.

Взаимодействие между элементами данного шаблона представлено на рисунке 4.1.



Рис. 4.1. Архитектурный шаблон MVC

### *Реализация шаблона MVC в фреймворке Qt*

Фреймворк Qt предлагает иной подход к реализации архитектуры MVC - он предполагает объединение контроллера и представления, но добавляет понятия делегат и выделение. Предлагаемая архитектура изображена на рисунке 4.2.



Рис. 4.2. - Реализация архитектуры MVC в фреймворке Qt

В данной реализации добавляются два новых понятия:

1. Выделение - специальная модель, облегчающая работу с выделенными в представлении данными.

2. Делегат — это компонент, который отображает данные одного элемента модели и обеспечивает их редактирование. Экземпляры делегата создаются для каждого элемента модели и располагаются в представлении, которое по сути является контейнером. Именно делегат решает, как должны отображаться и редактироваться данные конкретного элемента.

Для реализации приложения согласно данной архитектуре, компонентная модель Qt предлагает заранее созданные представления, основными из которых являются:

- `QListView` - представление в виде одномерного списка элементов. Поддерживает добавление иконок к элементам списка.
- `QTableView` - представление в виде таблицы,
- `QTreeView` - представление в виде иерархического дерева элементов.

Выбор представления обычно обусловлен задачей. К каждому представлению в один момент времени может быть подключена одна модель и один делегат, отвечающий за отображение элементов модели.

### ***Виды и использование моделей в фреймворке Qt***

Основным элементом программы в данной работе должна являться разработанная обучающимся на основе стандартной модель, обеспечивающая взаимодействие с данными. В рамках работы запрещено использовать стандартную модель, необходимо разработать свою, унаследованную от одно из двух абстрактных классов:

1. `QAbstractItemModel` - представляет собой базовый класс для всех моделей Qt и позволяет реализовать древовидную структуру модели.

2. `QAbstractListModel` - представляет собой линейную модель данных и применяется при необходимости взаимодействовать с линейной структурой данных, например списком.

Каждая из этих моделей требует переопределения следующих методов, представленных в таблице 4. При этом параметр role указывает текущую роль. О ролях будет написано ниже. Параметр parent важен для древовидной модели и позволяет создать иерархию элементов, для линейной модели он игнорируется.

Таблица 4.

### Методы модели, требующие переопределения

| Метод   | Описание   |
|---|--|
| QVariant headerData(int section, Qt::Orientation orientation, int role) | Позволяет задавать подписи в заголовках. Section содержит номер строки или столбца в зависимости от orientation. |
| QVariant data(const QModelIndex &index, int role)                       | Возвращает данные по указанному индексу модели   |
| int rowCount(const QModelIndex &parent)                                 | Возвращает число строк в модели  |
| bool setData(const QModelIndex &index, const QVariant &value, int role) | Записывает значение value по указанному индексу модели   |
| Qt::ItemFlags flags(const QModelIndex& index)                           | Возвращает флаги, установленные для данного элемента, позволяет защитить конкретный элемент модели от изменений  |
| bool insertRows(int row, int count, const QModelIndex &parent)          | Добавляет count строк после строки с индексом row.   |
| bool removeRows(int row, int count, const QModelIndex &parent)          | Удаляет count строк начиная со строки с индексом row.  |
| Только для QAbstractItemModel   |  |
| QModelIndex index(int row, int column, const QModelIndex &parent)       | Возвращает индекс элемента модели, основываясь на индексах строки, столбца и родительского элемента              |
| QModelIndex parent(const QModelIndex &index)                            | Возвращает индекс родительского элемента по отношению к указанному   |
| bool hasChildren(const QModelIndex &parent)                             | Проверяет есть ли у указанного элемента потомки  |
| int columnCount(const QModelIndex &parent)                              | Возвращает число столбцов в модели   |

#### Продолжение таблицы 4

|  |  |
|--|--|
| bool insertColumns(int column, int count, const QModelIndex &parent) | Добавляет count столбцов после столбца с индексом column.    |
| bool removeColumns(int column, int count, const QModelIndex &parent) | Удаляет count столбцов начиная со столбца с индексом column. |

Модель как правило работает с набором данных, при этом в зависимости от ситуации может потребоваться только часть из них, скажем при отображении элемента модели в списке требуется его имя, а при щелчке по нему - структура, содержащая данные об элементе. Также каждому элементу могут соответствовать иконка, подсказка при наведении на него курсора и прочее. Для того чтобы обеспечить хранение таких разнородных данных и выдачу только тех, которые требуются в конкретном случае, в Qt используется система ролей. Роль определяет какие именно данные от модели требуются представлению в конкретном случае. Для корректного выполнения задания потребуется использование как минимум следующих ролей:

- Qt::DisplayRole - сообщает что данные требуются для отображения в представлении (например, в виде списка)
- Qt::EditRole - сообщает что данные требуются для редактирования элемента в представлении (позволяет при редактировании выводить более полную информацию, либо, например шаблон данных)
- Qt::UserRole - дополнительный набор ролей, данные потребовались по специальному запросу и таким образом может быть, например получена структура, описывающая весь элемент. Таких ролей может быть несколько, константа Qt::UserRole хранит номер первой из таких ролей.

#### ***Реализация шаблона MVVM средствами WPF***

В качестве основного архитектурного шаблона и альтернативы MVC в технологии WPF выступает шаблон MVVM (Model - View - ViewModel), предполагающий использование следующих модулей:

Представление (англ. View) - набор элементов пользовательского интерфейса, как правило описываемый в XAML и отвечающий за отображение данных пользователю и взаимодействие с ним.

Модель (англ. Model) - описывает структуру данных и, при необходимости, содержит методы для их обработки.

Модель-Представление (англ. ViewModel) - строго структурированный буфер между моделью и представлением, отвечающий за взаимодействие с источником данных посредством модели, выделение требуемых данных в виде отдельных свойств, обновление данных и представления.

Взаимоотношение между этими модулями показано на рисунке 4.3.

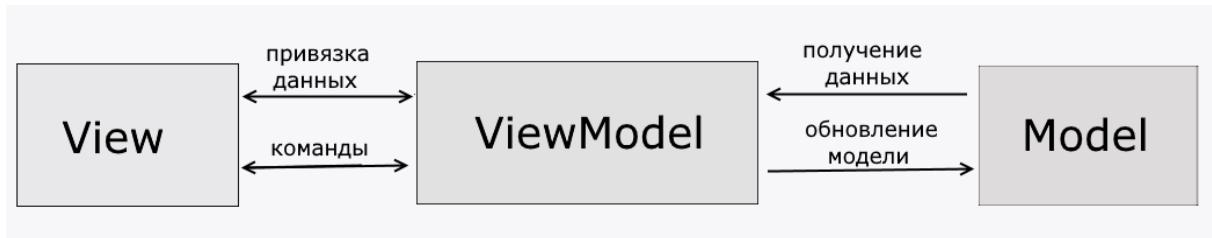


Рис. 4.3. Структура шаблона MVVM

Непосредственно источник данных в рамках данного шаблона может быть как инкапсулирован в модели, так и подключён к модели-представлению. В последнем случае модель описывает интерфейс взаимодействия с источником данных.

Реализация шаблона MVVM неразрывно связана с понятием привязки данных (англ. data binding), которое позволяет связывать значения, установленные для элемента пользовательского интерфейса напрямую со свойствами объекта, хранящими требуемые данные как на этапе проектирования разметки интерфейса, так и внутри реализации соответствующего класса. Для этого описываемой форме сопоставляется контекст данных, в роли которого выступает созданная модель-представление, свойства которой сопоставляются значениям конкретных элементов интерфейса.

Общая идея реализации шаблона MVVM сводится к созданию как минимум двух классов - модели и модели-представления. При этом в качестве

одного из полей класса модели-представления должен выступать объект или коллекция объектов класса модели. Объект класса модель-представления устанавливается в качестве контекста данных для формы, которая выступает в качестве представления.

Для обеспечения актуальности данных в представлении, модель или модель-представление, или оба модуля сразу должны реализовывать интерфейс `INotifyPropertyChanged`, который вызывает событие `PropertyChanged` при любом изменении свойств. Реализация данного интерфейса потребует добавить в описание соответствующего класса поле, принимающее обработчик событий и реализацию метода  `OnPropertyChanged`, который будет вызывать событие. Пример требуемой реализации представлен далее:

```
public event PropertyChangedEventHandler PropertyChanged;
public void OnPropertyChanged([CallerMemberName]string prop = "") {
    if (PropertyChanged != null)
        PropertyChanged(this, new PropertyChangedEventArgs(prop));
}
```

При этом если модель или модель-представление в качестве одного из свойств должна хранить коллекцию объектов, то рекомендуется использовать наблюдаемые варианты коллекций, например `ObservableCollection`, чтобы получать актуальные события об её изменении.

Для манипуляции данными, модель-представление использует механизм команд. Команды являются элементо-независимым аналогом событий - если обработка события как правило жёстко привязана к элементу интерфейса, вызвавшему ему, то команды являются более универсальным механизмом и описывают действия, которые могут быть выполнены с различными элементами интерфейса, как например получить значение или установить его.

Каждая команда представляет собой объект класса, реализующего интерфейс `ICommand`:

```
public interface ICommand
```

```

{
event EventHandler CanExecuteChanged;
bool CanExecute(object parameter);
void Execute(object parameter);
}

```

Данный интерфейс требует добавить в класс поле, содержащие обработчик события, вызываемого при изменении условий выполнения команды, а также два метода: `CanExecute`, который проверяет параметры команды и сообщает о возможности её выполнить, и `Execute`, который описывает требуемые действия.

Сами команды должны храниться в виде отдельных свойств внутри модели-представления и быть связаны со свойством `Command` элементов представления посредством привязки данных.

В качестве примера, можно привести популярную универсальную реализацию интерфейса `ICommand`, которая позволяет назначить любое действие команде.

```

public class RelayCommand : ICommand
{
    private Action<object> execute;
    private Func<object, bool> canExecute;

    public event EventHandler CanExecuteChanged
    {
        add { CommandManager.RequerySuggested += value; }
        remove { CommandManager.RequerySuggested -= value; }
    }

    public RelayCommand(Action<object> execute, Func<object, bool> canExecute = null)
    {
        this.execute = execute;
        this.canExecute = canExecute;
    }

    public bool CanExecute(object parameter)
    {
        return this.canExecute == null || this.canExecute(parameter);
    }
}

```

```
public void Execute(object parameter)
{
    this.execute(parameter);
}
```

Данная реализация принимает на вход конструктора один или два параметра - делегат типа Action, определяющий требуемые действия и делегат типа Func<object,bool>, позволяющий проверить возможность выполнения команды с переданными параметрами.

После этого в классе модели-представления следует объявить в качестве поля объект класса RelayCommand, передав в него необходимые делегаты. А в XAML описании представления привязать данное поле с помощью свойств Command и CommandParameter:

```
<Button Command="{Binding Command}" CommandParameter = "{Binding Element}" />
```

Подробнее про шаблон проектирования MVVM и его реализацию с помощью технологии WPF рассказано в лекциях и рекомендованной литературе.

### ***Работа с иерархическими моделями данных средствами WPF***

Для работы с иерархической структурой данных с использованием шаблона проектирования MVVM потребуется создать набор классов моделей. Для каждого элемента в цепочке предок-потомок потребуется свой класс модели, который в качестве одного из свойств должен содержать ObservableCollection из дочерних объектов. Эти объекты могут быть как объектами того же класса, так и объектами другого класса модели в зависимости от решаемой задачи.

Модель-представление при этом в качестве свойства помимо требуемых команд должна содержать ObservableCollection из объектов класса «корневой» модели.

В качестве представления такой иерархии рекомендуется использовать Tree View, которому в качестве источника элементом необходимо назначить коллекцию «корневых» объектов из модели-представления.

Следующим этапом станет настройка свойств Tree View – необходимо описать шаблоны иерархии элементов, назначив для каждого уровня иерархии соответствующий класс модели, свойство модели и элемент его отображение.

Пример такой настройки приведён ниже:

```
<TreeView x:Name="MainTree" ItemsSource="{Binding LEGOs}" TreeViewItem.Selected="OnItemSelected">
    <TreeView.ItemTemplate>
        <HierarchicalDataTemplate ItemsSource="{Binding Packs}" DataType="{x:Type local:Pack}">
            <Label Content="{Binding LEGOName}" />
            <HierarchicalDataTemplate.ItemTemplate>
                <HierarchicalDataTemplate ItemsSource="{Binding Details}" DataType="{x:Type local:Detail}">
                    <Label Content="{Binding PackNum}" />
                    <HierarchicalDataTemplate.ItemTemplate>
                        <DataTemplate DataType="{x:Type local:Detail}">
                            <Label Content="{Binding DetailName}" />
                        </DataTemplate>
                    </HierarchicalDataTemplate.ItemTemplate>
                </HierarchicalDataTemplate>
            </HierarchicalDataTemplate.ItemTemplate>
        </HierarchicalDataTemplate>
    </TreeView.ItemTemplate>
</TreeView>
```

В данном случае рассматривается иерархия, описывающая конструкторы ЛЕГО: Корневым элементом является коллекция объектов класса LEGO под названием LEGOs, каждый из которых содержит своё наименование в поле LEGOName и коллекцию объектов следующего уровня. Следующим уровнем является коллекция Packs, содержащая объекты класса Pack, каждый из которых содержит своё наименование в поле PackNum и коллекцию объектов следующего уровня. Последним уровнем является коллекция Details, содержащая объекты класса Detail, каждый из которых содержит своё наименование в поле DetailName и коллекцию объектов следующего уровня.

При верной организации иерархии, настроенный подобным образом TreeView позволит отображать и управлять иерархической моделью данных.

## ***Работа с таймером***

При выполнении задания данной работы помимо создания модели может потребоваться использование таймера для динамического изменения содержимого модели данных.

В Qt для этого используется объект класса QTimer, который предпочтительно создавать в конструкторе формы и, после задания времени срабатывания с использованием метода setInterval(int), связать его сигнал timeout() со слотом, который будет срабатывать на каждом тике таймера. Запускается таймер методом start(), останавливается - методом stop().

В WPF также присутствует такая возможность, для этого используется компонент DispatcherTimer. Он работает практически так же, как и QTimer, с поправкой на особенности технологии, его не нужно размещать на форме; он создается и используется только в программном коде.

Для использования класса DispatcherTimer необходимо задать интервал времени и подписаться на событие Tick, которое возникает каждый раз при истечении этого интервала. Для запуска таймера DispatcherTimer необходимо вызвать его метод Start() или присвоить true его свойству IsEnabled.

Также нужно помнить о том, что как QTimer, так и DispatcherTimer не всегда обеспечивают абсолютную точность. События таймера помещаются в очередь диспетчера, т.е. если компьютер сильно загружен, то выполнение ваших операций может происходить с задержкой. Как правило, инструменты разработки гарантируют, что событие Tick никогда не возникнет раньше, чем закончится интервал времени, но не гарантирует, что оно не возникнет немного позже. Тем не менее, в большинстве случаев как QTimer, так и DispatcherTimer обеспечивают более чем достаточную точность.

## ***Общая постановка задачи***

В данной работе необходимо написать программу в соответствие с индивидуальным вариантом. Выполнять задание можно либо средствами

фреймворка Qt, либо используя технологию WPF и язык программирования C#.

В вариантах ниже указаны виджеты Qt, при использовании C# необходимо использовать аналогичные элементы управления WPF.

При создании программы необходимо использовать один из архитектурных шаблонов и описать свою модель данных, унаследованную от одной из базовых. В ряде вариантов предполагается также использование таймеров.

**При выполнении средствами Qt, недопустимо использование стандартных моделей! Обязательным требованием к работе является создание своей модели данных!**

### **Варианты заданий.**

#### ***Вариант 1***

Необходимо провести моделирование и визуализацию процесса работы с информацией на HDD.

Для начала крайне краткие и упрощённые сведения из теории. Время считывания или записи одного блока диска определяется следующими тремя факторами:

1. Время поиска (время перемещения головки на нужный цилиндр);
2. Задержка вращения (время, требуемое для поворота нужного сектора под головку);
3. Время передачи данных.

Для большинства дисков первая составляющая (время поиска) существенно превосходит остальные две, поэтому значительного увеличения производительности системы можно добиться, снижая время поиска.

При высокой загрузке диска высока вероятность поступления новых запросов во время перемещения головки для обработки предыдущего запроса. При этом, дисковые драйверы содержат таблицу, индексированную по

номерам цилиндров, в который в единый связный список собираются все поступившие и ждущие обработки обращения к цилиндрам.

В рамках данной работы требуется реализовать программу, работающую по следующей схеме: при запуске пользователь задаёт параметры диска (количество цилиндров, скорость (или время) перемещения головки между цилиндрами и т.д.), начальное количество запросов на работу с данными. После этого генерируется указанное количество запросов, из которых составляется изначальный список запросов. На каждом шаге таймера с определённой вероятностью (указывается пользователем) в список добавляется новый запрос. На каждом шаге таймера список запросов просматривается и сортируется по указанному ниже алгоритму. Необходимо хранить текущее положение головки диска.

Необходимо хранить информацию о каждом запросе в виде структуры. Обязательные поля структуры: время поступления запроса, длительность запроса, номер цилиндра, id запроса, время завершения запроса. При желании можно добавить поле со случайно генерируемым именем файла.

Используемый алгоритм сортировки:

Данный алгоритм выбирает каждый раз ближайший цилиндр для поиска, т.е. цилиндр, номер которого минимально отличается от текущего.

## *Вариант 2*

Необходимо провести моделирование и визуализацию процесса работы с информацией на HDD.

Для начала крайне краткие и упрощённые сведения из теории. Время считывания или записи одного блока диска определяется следующими тремя факторами:

1. Время поиска (время перемещения головки на нужный цилиндр);
2. Задержка вращения (время, требуемое для поворота нужного сектора под головку);
3. Время передачи данных.

Для большинства дисков первая составляющая (время поиска) существенно превосходит остальные две, поэтому значительного увеличения производительности системы можно добиться, снижая время поиска.

При высокой загрузке диска высока вероятность поступления новых запросов во время перемещения головки для обработки предыдущего запроса. При этом, дисковые драйверы содержат таблицу, индексированную по номерам цилиндров, в который в единый связный список собираются все поступившие и ждущие обработки обращения к цилиндрам.

В рамках данной работы требуется реализовать программу, работающую по следующей схеме: при запуске пользователь задаёт параметры диска (количество цилиндров, скорость (или время) перемещения головки между цилиндрами и т.д.), начальное количество запросов на работу с данными. После этого генерируется указанное количество запросов, из которых составляется изначальный список запросов. На каждом шаге таймера с определённой вероятностью (указывается пользователем) в список добавляется новый запрос. На каждом шаге таймера список запросов просматривается и сортируется по указанному ниже алгоритму. Необходимо хранить текущее положение головки диска.

Необходимо хранить информацию о каждом запросе в виде структуры. Обязательные поля структуры: время поступления запроса, длительность запроса, номер цилиндра, id запроса, время завершения запроса. При желании можно добавить поле со случайно генерируемым именем файла.

Используемый алгоритм сортировки:

Головка диска движется в одном направлении и производит поиск требуемых цилиндров до тех пор, пока на этом направлении не останется запросов. Затем направление головки меняется на противоположное.

### *Вариант 3*

Необходимо провести моделирование и визуализацию процесса работы лифтов в многоэтажном доме.

В многоэтажном доме имеется несколько лифтов. Они перемещаются сверху вниз и снизу вверх. Если нет запросов на вызов для лифтов, и поступает новый запрос на  $i$ -ом этаже, то его обрабатывает ближайший лифт. Если при обработке текущего запроса поступают новые, то действует элеваторный алгоритм: лифт движется в одном направлении до тех пор, пока не обработает все запросы. Если все лифты движутся в одном и том же направлении, а запрос на вызов поступает в противоположном направлении, то его принимает для обработки тот лифт, который сделает это быстрее. При этом элеваторный алгоритм (движение в одном направлении) перестает работать до обработки данного запроса. В каждом лифте ограниченное количество мест.

Запросы хранятся в виде структуры со следующими обязательными полями: этаж, с которого вызван лифт, целевой этаж, время ожидания, id лифта, который обработал запрос.

На входе задается количество этажей и лифтов в доме, а также вероятность появления запросов. В каждый тик времени с указанной вероятностью появляется один или несколько запросов.

Необходимо хранить информацию о каждом запросе в виде структуры.

#### ***Вариант 4***

Требуется разработать программу для отображения иерархической структуры данных о птицах.

В данной работе требуется создать интерактивный справочник птиц. Все роды птиц организованы согласно принятой иерархии, пример которой приведён на рисунке 4.4:

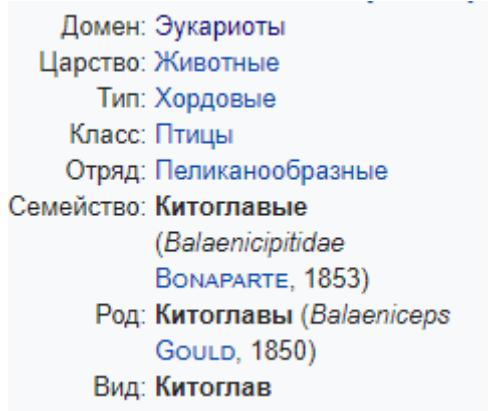


Рис. 4.4. Пример иерархии

В левой части приложения должна быть представлена данная иерархия, для отображения которой должен использоваться QTreeView.

По каждому объекту иерархии (род, семейство, класс и т.п.) должна быть возможность получения дополнительной информации - изображения и описания, которые отобразятся в правой части приложения.

Каждый объект иерархии хранится в виде структуры, обязательными полями в которой являются имя, текст и изображение.

Следует реализовать возможность полноценной работы с моделью: добавление, удаление и редактирование объектов, а также изменения иерархии. Также следует предусмотреть возможность поиска птиц по ареалу обитания (скажем отобразить список птиц из Северной Америки с возможностью получения информации об иерархии для каждой из них)

### ***Вариант 5***

Требуется разработать программу для отображения иерархической структуры данных о растениях.

В данной работе требуется создать интерактивный справочник растений. Все роды растений организованы согласно принятой иерархии, пример которой приведён на рисунке 4.5:

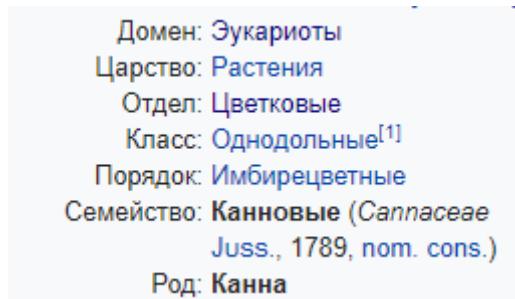


Рис. 4.5. Пример иерархии

В левой части приложения должна быть представлена данная иерархия, для отображения которой должен использоваться QTreeView.

По каждому объекту иерархии (род, семейство, класс и т.п.) должна быть возможность получения дополнительной информации - изображения и описания, которые отобразятся в правой части приложения.

Каждый объект иерархии хранится в виде структуры, обязательными полями в которой являются имя, описание и изображение.

Следует реализовать возможность полноценной работы с моделью: добавление, удаление и редактирование объектов, а также изменения иерархии. Также необходимо предусмотреть возможность сохранения иерархии в файл формата JSON и загрузки ранее сохранённой.

### *Вариант 6*

Необходимо провести моделирование и визуализацию процесса выделения памяти под совокупность задач. Каждая задача представляется следующими атрибутами: имя, размер (в байтах), максимально допустимое время начала выполнения задачи, длительность выполнения. Совокупность задач образует очередь задач.

Необходимо хранить информацию о каждой задаче в виде структуры.

На входе пользователь задает общий объем памяти (в байтах), количество задач и прочие настройки. После этого происходит генерация указанного количества задач в очередь. Задачи помещаются в очередь в порядке, соответствующем времени начала выполнения. Задачи выбираются из очереди начиная с самой первой, после чего происходит оценка

возможности помещения задачи в список на выполнение. Если максимальное время начала выполнения текущей задачи истекло, а для неё не нашлось места в списке выполняемых задач, она отправляется в список задач с истекшим временем начала выполнения.

На каждом шаге таймера у всех задач из списка выполняемых длительность уменьшается на единицу, по достижению 0 задача перемещается в список завершённых.

В данном варианте дополнительную сложность составляет организация памяти. Используется организация памяти в виде списка свободных областей. Изначально существует только одна область, размер которой равен общему размеру памяти.

При поступлении задачи, менеджер памяти просматривает весь список областей и выбирает наименьший по размеру подходящий участок памяти. Затем этот участок делится на 2 части: одна (размер которой равен размеру задачи) отдается задаче, а другая остается неиспользуемой. После выполнения задачи, занимаемый ею участок помечается как свободный. Объединение участков не происходит.

### ***Вариант 7***

Необходимо провести моделирование и визуализацию процесса работы фуд трака.

В фуд траке предлагается фиксированное меню из блюд, для каждого определена цена, прибыль (цена минус себестоимость, может быть и отрицательной), время обработки заказа и категория (основное блюдо, закуска, напиток и т.п.).

К фуд траку подходят клиенты, образуя очередь. Каждый клиент представляется в виде структуры со следующими обязательными полями: id желаемых блюд (обязательно из разных категорий, для части категорий может стоять 0, что будет означать что из данной категории заказа не будет), максимальное время ожидания. Первый клиент в очереди подходит к фуд

траку и делает заказ. Фуд трак может обрабатывать одновременно заказы нескольких клиентов. После получения заказа клиент ставит оценку по 5ти бальной шкале (критерии определяет разработчик, проявите фантазию или просто рандом). Данные получившего заказ клиента, включая оценку, добавляется в список “обслуженных”.

Если клиент стоит в очереди время, превышающее его время ожидания, то он в гневе уходит и данные о нём помещаются в список “недовольных” с пометкой “долго стоял”.

При запуске программы пользователь задаёт параметры фуд трака (можно использовать заранее заданное меню или реализовать его заполнение пользователем) и начальное количество клиентов в очереди. После этого генерируется указанное количество клиентов. На каждом шаге таймера с определённой вероятностью (указывается пользователем) в очередь добавляется новый клиент.

Необходимо хранить информацию о каждом клиенте в виде структуры.

Также необходимо вести подсчёт полученной прибыли и упущеной.

### ***Вариант 8***

Необходимо провести моделирование и визуализацию процесса работы автоматизированной системы обороны SKNT-1.

В зону действия SKNT-1 поступают различные объекты. В распоряжении системы SKNT-1 имеется определённое количество орудий противодействия трёх различных типов (земля-земля, земля-воздух, земля-вода). Для эффективного противодействия каждому объекту требуется работа определённого количества орудий на протяжении заданного количества времени.

Информация о каждом объекте хранится в виде структуры со следующими полями: скорость (определяет время, которое объект находится в зоне действия орудий), тип, наименование, требуемое количество орудий для

эффективного противодействия (не может быть больше общего числа орудий данного типа), потенциальный ущерб, выраженный в условных единицах.

Каждый объект, попавший в зону действия орудий SKNT-1, находится в ней в течении ограниченного времени, определяемого его скоростью. SKNT-1 отслеживает все объекты в зоне действия орудий и сортирует список этих объектов согласно оставшемуся времени на их “обработку”. В каждый такт времени SKNT-1 выбирает из данного списка объект, с максимальным приоритетом, на “обработку” которого хватает свободных орудий. Объект, подвергающийся противодействию, уже не может покинуть зону действия орудий.

На входе пользователь указывает параметры SKNT-1 - количество орудий каждого типа и время, требуемое на эффективное противодействия объектам противника.

Необходимо хранить информацию о каждом автомобиле в виде структуры.

Также следует подсчитывать предотвращённый и понесённый ущерб.

### ***Вариант 9***

Необходимо провести моделирование и визуализацию процесса работы бистро.

В бистро предлагается фиксированное меню из блюд, для каждого определена цена, время обработки заказа и категория (основное блюдо, закуска, напиток и т.п.).

К бистро подходят клиенты, образуя очередь. Каждый клиент представляется в виде структуры со следующими обязательными полями: id желаемых блюд (обязательно из разных категорий, для части категорий может стоять 0, что будет означать что из данной категории заказа не будет), максимальное время ожидания. Первый клиент в очереди подходит к бистро и делает заказ. бистро может готовить одновременно по определённому количеству блюд из каждой категории. Если для каждого блюда из заказа

клиента есть место, то заказ принимается и начинается готовка, иначе клиент продолжает ждать своей очереди. Данные успешно получившего заказ клиента, добавляется в список “обслуженных”.

Если клиент стоит в очереди время, превышающее его время ожидания, то он в гневе уходит и данные о нём помещаются в список “недовольных” с пометкой “долго стоял”.

При запуске программы пользователь задаёт параметры быстро: меню (можно использовать заранее заданное меню или реализовать его заполнение пользователем), количество блюд каждой категории, которые можно готовить одновременно и начальное количество клиентов в очереди. После этого генерируется указанное количество клиентов. На каждом шаге таймера с определённой вероятностью (указывается пользователем) в очередь добавляется новый клиент.

Необходимо хранить информацию о каждом клиенте в виде структуры.

### ***Вариант 10***

Требуется разработать программу для отображения иерархической структуры данных о космических объектах.

В данной работе требуется создать интерактивный справочник космических объектов. Для примера можно привести следующую иерархию:

Земля → Солнечная система → Местное межзвездное облако → Местный пузырь → Пояс Гулда → Рукав Ориона → Млечный Путь → Подгруппа Млечного Пути → Местная группа → Местный лист → Местное скопление галактик → Ланиакея → Комплекс скоплений Рыб-Кита → Объём Хаббла → Метагалактика → Вселенная → Мультивселенная

В левой части приложения должна быть представлена данная иерархия, для отображения которой должен использоваться QTreeView.

По каждому объекту иерархии должна быть возможность получения дополнительной информации - изображения и описания, которые отобразятся в правой части приложения.

Каждый объект иерархии хранится в виде структуры, обязательными полями в которой являются имя, текст и изображение.

Следует реализовать возможность полноценной работы с моделью: добавление, удаление и редактирование объектов, а также изменения иерархии. Также необходимо предусмотреть возможность сохранения иерархии в бинарный файл, не используя общепринятый язык разметки, и загрузки из него.

### ***Вариант 11***

Необходимо провести моделирование и визуализацию процесса выделения памяти под совокупность задач. Каждая задача представляется следующими атрибутами: имя, размер (в байтах), максимально допустимое время начала выполнения задачи, длительность выполнения. Совокупность задач образует очередь задач.

Необходимо хранить информацию о каждой задаче в виде структуры.

На входе пользователь задает общий объем памяти (в байтах), количество задач и прочие настройки. После этого происходит генерация указанного количества задач в очередь. Задачи помещаются в очередь в порядке, соответствующем времени начала выполнения. Задачи выбираются из очереди начиная с самой первой, после чего происходит оценка возможности помещения задачи в список на выполнение. Если максимальное время начала выполнения текущей задачи истекло, а для неё не нашлось места в списке выполняемых задач, она отправляется в список задач с истекшим временем начала выполнения.

На каждом шаге таймера у всех задач из списка выполняемых длительность уменьшается на единицу, по достижению 0 задача перемещается в список завершённых.

В данном варианте дополнительную сложность составляет организация памяти. Используется организация памяти в виде списка свободных областей. Изначально существует только одна область, размер которой равен общему размеру памяти.

При поступлении задачи, менеджер памяти просматривает список областей до тех пор, пока не находит достаточно большой свободный участок для размещения процесса. Затем этот участок делится на 2 части: одна (размер которой равен размеру задачи) отдается задаче, а другая остается неиспользуемой. После выполнения задачи, занимаемый ею участок помечается как свободный. Объединение участков не происходит.

### *Вариант 12*

Необходимо провести моделирование процесса движения на перекрёстке.

На улице расположен перекресток, по которому движется поток автомобилей. Движение является двухсторонним и строго ограничено в направлениях: С-Ю, Ю-С, З-В, В-З. На перекрестке расположены два светофора, работающие синхронно. Когда один из них показывает красный цвет сигнала, то другой показывает зеленый, и наоборот. Время проезда  $k$ -ого автомобиля с одной стороны перекрестка случайно. Кроме того, оно зависит от количества машин, стоящих перед  $k$ -ой автомашиной. Для наглядности, схема перекрёстка отображена на рисунке 4.6. Отображать её в программе не требуется.

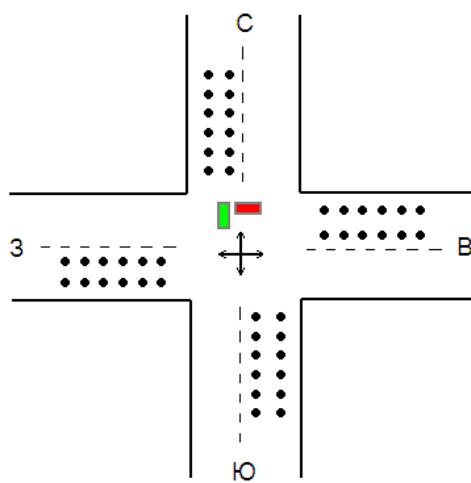


Рис. 4.6. Схема перекрёстка

Необходимо написать приложение, позволяющее моделировать данную ситуацию. На входной форме пользователь задает количество автомашин в

начальный момент времени. При этом, случайным образом выбирается маршрут автомобиля (возможны 4 варианта: С-Ю, Ю-С, З-В, В-З). На выходной форме необходимо отобразить: состояние светофоров; очереди автомобилей, направляющихся по маршруту С-Ю, Ю-С, З-В, В-З; список автомобилей, проехавших перекресток.

Информация о каждом автомобиле хранится в виде структуры со следующими полями: маршрут, номер, время прибытия на перекрёсток, время, затраченное на преодоление перекрёстка.

Необходимо хранить информацию о каждом автомобиле в виде структуры.

Графическое отображение перекрёстка не требуется.

По желанию, можно добавить аварии.

### ***Вариант 13***

Необходимо провести моделирование и визуализацию процесса выделения памяти под совокупность задач. Каждая задача представляется следующими атрибутами: имя, размер (в байтах), максимально допустимое время начала выполнения задачи, длительность выполнения. Совокупность задач образует очередь задач.

Необходимо хранить информацию о каждой задаче в виде структуры.

На входе пользователь задает общий объем памяти (в байтах), количество задач и прочие настройки. После этого происходит генерация указанного количества задач в очередь. Задачи помещаются в очередь в порядке, соответствующем времени начала выполнения. Задачи выбираются из очереди начиная с самой первой, после чего происходит оценка возможности помещения задачи в список на выполнение. Если максимальное время начала выполнения текущей задачи истекло, а для неё не нашлось места в списке выполняемых задач, она отправляется в список задач с истекшим временем начала выполнения.

На каждом шаге таймера у всех задач из списка выполняемых длительность уменьшается на единицу, по достижению 0 задача перемещается в список завершённых.

В данном варианте дополнительную сложность составляет организация памяти: память представляет собой массив, заданного пользователем размера. Изначально он заполнен 0. При помещении задачи в список выполняемых, соответствующее число подряд идущих элементов массива заменяется на id задачи. После выполнения задачи, занимаемая ею область освобождается (значения заменяются на 0). Соответственно задачей менеджера памяти является поиск в массиве нужного числа подряд идущих 0 для размещения каждой новой задачи. Дефрагментация массива не поддерживается.

### ***Вариант 14***

Требуется разработать программу для отображения иерархической структуры данных о животных.

В данной работе требуется создать интерактивный справочник животных. Все роды животных организованы согласно принятой иерархии, пример которой приведён на рисунке 4.7:

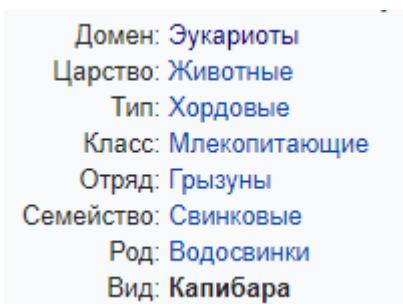


Рис. 4.7. Пример иерархии

В левой части приложения должна быть представлена данная иерархия, для отображения которой должен использоваться QTreeView.

По каждому объекту иерархии (род, семейство, класс и т.п.) должна быть возможность получения дополнительной информации - изображения и описания, которые отобразятся в правой части приложения.

Каждый объект иерархии хранится в виде структуры, обязательными полями в которой являются имя, текст и изображение.

Следует реализовать возможность полноценной работы с моделью: добавление, удаление и редактирование объектов, а также изменения иерархии. Также необходимо предусмотреть возможность сохранения иерархии в файл формата ini и загрузки ранее сохранённой.

### ***Вариант 15***

Необходимо провести моделирование и визуализацию процесса работы магазина.

В магазине выстроилась очередь за товарами. Список товаров строго ограничен. На каждого покупателя продавец тратит определенное время (в зависимости от списка покупаемых товаров и их количества). При этом каждый покупатель очень воспитан, а именно: мужчины пропускают женщин; все пропускают пенсионеров; пенсионеры-мужчины пропускают пенсионеров-женщин.

Покупатели хранятся в виде структуры со следующими обязательными полями: пол, возраст, время попадания в очередь, время, потраченное на обслуживание, список товаров. Для простоты можно ограничиться 10-15ю товарами на магазин и для каждого покупателя хранить по каждому товару покупаемое количество (0 - товар не нужен) и отображать только те товары, количество которых больше 0.

На входе задаются параметры каждого товара. Каждый тakt времени в очередь с определённой вероятностью приходит новый покупатель и очередь сортируется в соответствие с приоритетами. Очередь бесконечная.

Необходимо хранить информацию о каждом покупателе в виде структуры.

Также следует хранить статистику о спросе на каждый товар.

### ***Вариант 16***

Необходимо провести моделирование и визуализацию процесса проведения семинара.

В городе X проходит семинар. Докладчики, которые должны выступать на семинаре являются людьми непунктуальными. Приходя в произвольный момент времени, они проходят в зал и ожидают своего выхода. В данном случае очередность докладчиков определяется временем прихода (кто раньше пришел, тот и выступает). Время, отведенное на изложение доклада, является случайным. После каждого выступления те докладчики, что находятся в зале (как уже выступившие, так и ожидающие очереди, но не сам выступающий) с определённой вероятностью задают выступающему вопросы, на ответы на которые докладчик тратит по одной минуте на вопрос. После доклада каждый присутствующий в зале, исключая выступающего, ставит докладу оценку по 5ти бальной системе. Первый доклад начинается только когда в зале собралось не менее 4 х человек, включая выступающего.

Докладчики хранятся в виде структуры со следующими обязательными полями: время прихода на семинар, время ожидания, тема доклада, время на доклад, количество заданных вопросов, средняя оценка выступления.

На входной форме указывается начальное количество докладчиков в зале и общее число докладчиков. Очередность для них определяется случайно. После выступления всех докладчиков необходимо отсортировать их список соответственно оценкам доклада.

Необходимо хранить информацию о каждом докладчике в виде структуры.

### ***Вариант 17***

Необходимо провести моделирование и визуализацию процесса буферизации (буферизатор) печати.

Буферизатор принимает задание на печать и вставляет файл, который должен печататься, в очередь. При освобождении принтера буферизатор

удаляет задание из очереди печати, печатает файл и добавляет его в очередь выполненных заданий.

Задание на печать – это структура, содержащая номер задания, случайно сгенерированное имя файла, время поступления и число страниц.

При запуске программы пользователь задаёт параметры печати (количество одновременно печатающихся страниц (можно ограничиться одной), скорость печати одной страницы, максимальный размер очереди и т.д.) и начальное количество заданий на печать. После этого генерируется указанное количество заданий. На каждом шаге таймера с определённой вероятностью (указывается пользователем) в список добавляется новое задание. Если достигнут максимальный размер очереди, то задание переходит в список отклонённых заданий.

Необходимо хранить информацию о каждом задании в виде структуры.

### ***Вариант 18***

Требуется разработать программу для отображения иерархической структуры данных о структуре подразделений БГТУ “ВОЕНМЕХ” им. Д.Ф.Устинова.

В данной работе требуется создать интерактивный справочник о структуре подразделений БГТУ “ВОЕНМЕХ” им. Д.Ф.Устинова.

В левой части приложения должна быть представлена иерархия объектов, для отображения которой должен использоваться QTreeView.

По каждому объекту иерархии должна быть возможность получения дополнительной информации - изображения, контактов, ФИО руководителя и описания, которые отобразятся в правой части приложения.

Каждый объект иерархии хранится в виде структуры, обязательными полями в которой являются имя, текст, контакты, ФИО руководителя и изображение.

Следует реализовать возможность полноценной работы с моделью: добавление, удаление и редактирование объектов, а также изменения

иерархии. Также необходимо предусмотреть возможность сохранения иерархии в файл формата JSON и загрузки ранее сохранённой.

### ***Вариант 19***

Необходимо провести моделирование и визуализацию процесса работы цеха по разливу напитков.

В цехе по разливу напитков работают два конвейера. На первом конвейере напиток разливается по бутылкам, а на втором происходит закупоривание бутылок. После схода с первого конвейера очередной бутылки она сразу же поступает на второй. Поскольку используется различная тара, то каждая бутылка имеет свое время заполнения и время закупорки.

Бутылки хранятся в виде структуры со следующими обязательными полями: объём, тип напитка, скорость заполнения одного литра (общее время заполнение рассчитывается как объём x скорость), время закупоривания.

Все бутылки на конвейерах заполняются и закупориваются одновременно. Бутылки сходят с конвейера в порядке очереди при условии полного заполнения.

На каждом такте времени генерируется случайным образом вновь поступающая тара на конвейер 1. Если конвейер 1 заполнен, программа останавливается и выдаётся сообщение об ошибке. Если конвейер 2 заполнен, то бутылка, вышедшая с конвейера, помещается в буфер. Если буфер заполнен, конвейер останавливается и выдаётся сообщение об ошибке.

На входе пользователь задаёт допустимые параметры бутылок, длины конвейеров и буфера.

Необходимо хранить информацию о каждом покупателе в виде структуры.

### ***Вариант 20***

Необходимо провести моделирование и визуализацию процесса выделения памяти под совокупность задач. Каждая задача представляется

следующими атрибутами: имя, размер (в байтах), максимально допустимое время начала выполнения задачи, длительность выполнения. Совокупность задач образует очередь задач.

Необходимо хранить информацию о каждой задаче в виде структуры.

На входе пользователь задает общий объем памяти (в байтах), количество задач и прочие настройки. После этого происходит генерация указанного количества задач в очередь. Задачи помещаются в очередь в порядке, соответствующем времени начала выполнения. Задачи выбираются из очереди начиная с самой первой, после чего происходит оценка возможности помещения задачи в список на выполнение. Если максимальное время начала выполнения текущей задачи истекло, а для неё не нашлось места в списке выполняемых задач, она отправляется в список задач с истекшим временем начала выполнения.

На каждом шаге таймера у всех задач из списка выполняемых длительность уменьшается на единицу, по достижению 0 задача перемещается в список завершённых.

В данном варианте дополнительную сложность составляет организация памяти. Используется организация памяти в виде списка свободных областей. Изначально существует только одна область, размер которой равен общему размеру памяти.

При поступлении задачи, менеджер памяти просматривает весь список областей и выбирает наибольший по размеру подходящий участок памяти. Затем этот участок делится на 2 части: одна (размер которой равен размеру задачи) отдается задаче, а другая остается неиспользуемой. После выполнения задачи, занимаемый ею участок помечается как свободный. Объединение участков не происходит.

### ***Вариант 21***

Необходимо провести моделирование и визуализацию процесса работы автомата по продаже кофе или иных напитков.

В автомате предлагается фиксированное меню из напитков, для каждого определена стоимость и время обработки заказа.

К автомату подходят покупатели, образуя очередь. Каждый покупатель представляется в виде структуры со следующими обязательными полями: id желаемого напитка, максимальное время ожидания, бюджет. Первый покупатель в очереди подходит к автомату и пытается купить желаемый напиток. Если цена напитка превышает бюджет, огорчённый покупатель уходит, а данные о нём помещаются в список “недовольных” с пометкой “слишком дорого”. В ином случае автомат обрабатывает заказ и, подождав указанное время, довольный покупатель уходит, а данные о нём помещаются в список “довольных”. Если покупатель стоит в очереди время, превышающее его время ожидания, то он в гневе уходит и данные о нём помещаются в список “недовольных” с пометкой “долго стоял”.

При запуске программы пользователь задаёт параметры автомата (можно использовать заранее заданное меню или реализовать его заполнение пользователем) и начальное количество покупателей в очереди. После этого генерируется указанное количество покупателей. На каждом шаге таймера с определённой вероятностью (указывается пользователем) в очередь добавляется новый покупатель.

Необходимо хранить информацию о каждом покупателе в виде структуры.

### ***Вариант 22***

Требуется разработать программу для отображения иерархической структуры данных о территориальном делении РФ.

В данной работе требуется создать интерактивный справочник территориальных объектов РФ.

В левой части приложения должна быть представлена иерархия объектов, для отображения которой должен использоваться QTreeView.

По каждому объекту иерархии должна быть возможность получения дополнительной информации - изображения и описания, которые отобразятся в правой части приложения.

Каждый объект иерархии хранится в виде структуры, обязательными полями в которой являются имя, текст, координаты центра и изображение.

Следует реализовать возможность полноценной работы с моделью: добавление, удаление и редактирование объектов, а также изменения иерархии. Также необходимо предусмотреть возможность вычисления расстояния между центрами любых двух объектов в иерархии.

### ***Вариант 23***

Необходимо провести моделирование и визуализацию процесса работы с информацией на HDD.

Для начала крайне краткие и упрощённые сведения из теории. Время считывания или записи одного блока диска определяется следующими тремя факторами:

1. Время поиска (время перемещения головки на нужный цилиндр);
2. Задержка вращения (время, требуемое для поворота нужного сектора под головку);
3. Время передачи данных.

Для большинства дисков первая составляющая (время поиска) существенно превосходит остальные две, поэтому значительного увеличения производительности системы можно добиться, снижая время поиска.

При высокой загрузке диска высока вероятность поступления новых запросов во время перемещения головки для обработки предыдущего запроса. При этом, дисковые драйверы содержат таблицу, индексированную по номерам цилиндров, в который в единый связный список собираются все поступившие и ждущие обработки обращения к цилиндрам.

В рамках данной работы требуется реализовать программу, работающую по следующей схеме: при запуске, пользователь задаёт

параметры диска (количество цилиндров, скорость (или время) перемещения головки между цилиндрами и т.д.), начальное количество запросов на работу с данными. После этого генерируется указанное количество запросов, из которых составляется изначальный список запросов. На каждом шаге таймера с определённой вероятностью (указывается пользователем) в список добавляется новый запрос. На каждом шаге таймера список запросов просматривается и сортируется по указанному ниже алгоритму. Необходимо хранить текущее положение головки диска.

Необходимо хранить информацию о каждом запросе в виде структуры. Обязательные поля структуры: время поступления запроса, длительность запроса, номер цилиндра, id запроса, время завершения запроса. При желании можно добавить поле со случайно генерируемым именем файла.

Используемый алгоритм сортировки:

Головка диска всегда движется в одном направлении. При достижении последнего цилиндра головка перемещается на нулевой цилиндр и процесс поиска продолжается.

### ***Вариант 24***

Необходимо провести моделирование и визуализацию процесса выделения памяти под совокупность задач. Каждая задача представляется следующими атрибутами: имя, размер (в байтах), максимально допустимое время начала выполнения задачи, длительность выполнения. Совокупность задач образует очередь задач.

Необходимо хранить информацию о каждой задаче в виде структуры.

На входе пользователь задает общий объем памяти (в байтах), количество задач и прочие настройки. После этого происходит генерация указанного количества задач в очередь. Задачи помещаются в очередь в порядке, соответствующем времени начала выполнения. Задачи выбираются из очереди начиная с самой первой, после чего происходит оценка возможности помещения задачи в список на выполнение. Если максимальное

время начало выполнения текущей задачи истекло, а для неё не нашлось места в списке выполняемых задач, она отправляется в список задач с истекшим временем начала выполнения.

На каждом шаге таймера у всех задач из списка выполняемых длительность уменьшается на единицу, по достижению 0 задача перемещается в список завершённых.

В данном варианте дополнительную сложность составляет организация памяти. Используется организация памяти в виде списка свободных областей. Изначально существует только одна область, размер которой равен общему размеру памяти.

При поступлении задачи, менеджер памяти просматривает список областей до тех пор, пока не находит достаточно большой свободный участок для размещения процесса. Затем оценивается размер этого участка - если он превосходит размер, требуемый для задачи более чем в два раза, то этот участок делится на 2 равные части: одна отдается задаче, а другая остается неиспользуемой. В противном случае весь участок отдаётся задаче. После выполнения задачи, занимаемый ею участок помечается как свободный. Объединение участков не происходит.

## **Контрольные вопросы**

Контрольные вопросы к данной работе разбиты на три категории: общие, Qt и C#. Раскройте содержание следующих тем по выполненной работе:

### ***Общие темы:***

1. Архитектурный шаблон MVC и его элементы.
2. Создание собственной линейной модели.
3. Особенности работы с иерархической моделью данных.
4. Работа с таймерами.
5. Особенности табличной модели данных.

***В случае выполнения работы с использованием фреймворка Qt:***

1. Реализация MVC в Qt.
2. Отличия QListView и QListWidget.
3. Роли в моделях Qt.
4. Работа со строками в Qt.
5. Делегаты в Qt.

***В случае выполнения работы с использованием технологии WPF:***

1. MVVM - описание и реализация.
2. Команды и их отличие от событий в WPF.
3. Привязка данных в WPF.
4. Настройка элементов представления в WPF.
5. ObservableCollection и интерфейс INotifyPropertyChanged.

## **ПРАКТИЧЕСКАЯ РАБОТА 5**

### **«Объектно-ориентированный подход к созданию графических сцен»**

#### **Теоретические сведения**

При разработке графических приложений, в определённый момент может возникнуть необходимость динамического создания, удаления и перемещения различных графических объектов. При этом целесообразно использовать объектно-ориентированный подход в сочетании с возможностями используемого инструмента разработки по созданию и управлению виртуальной сценой.

Фреймворк Qt предоставляет такие возможности посредством использования QGraphicScene, тогда как при использовании технологии WPF схожую функциональность предоставляет класс Canvas.

#### ***Использование QGraphicScene для создания графической сцены***

Основой рисования в фреймворке Qt является использование объекта класса QPainter, который позволяет рисовать различные примитивы, используя объекты класса QPen (перья) для рисования границ и класса QBrush (кисти) для заполнения примитивов. Сам процесс рисования происходит в обработчике события paintEvent конкретного виджета. При этом в качестве кисти может выступать в том числе заранее загруженное изображение.

Основная идея создания сложных графических сцен состоит в использовании связки QGraphicScene и QGraphicView. При этом первый объект выступает в качестве виртуальной сцены и хранит все объекты и их свойства, а второй является графическим представлением, отображающим сцену и отвечающим за взаимодействие с пользователем. Это позволяет в числе прочего связывать одну сцену с несколькими представлениями, каждое из которых может отображать различные фрагменты сцены или отображать модифицированный при помощи аффинных преобразований вариант сцены.

Объектом сцены может являться объект, унаследованный от класса QGraphicsItem, при этом, благодаря одному из его дочерних классов -

`QGraphicsProxyWidget`, существует также возможность использовать любой виджет в качестве объекта графической сцены.

В отличие от классического подхода к рисованию объектов, использование графической сцены позволяет проводить манипуляции над уже размещёнными на ней объектами - перемещать их, изменять их внешний вид, удалять их и добавлять реакцию на события. Это достигается благодаря тому, что все операции по отображению изменённой сцены берёт на себя представление, тогда как разработчик описывает исключительно виртуальное представление объектов - правила их отрисовки, возможные реакции на события, взаимное расположение объектов.

В целом создание собственного объекта сцены заключается в описании класса, наследующего от `QGraphicsItem`, и, при необходимости использовать обработку событий, от `QObject`. В описываемом классе необходимо переопределить следующие методы:

`QPainterPath shape()` - описывает контур объекта, служит для определения границ объекта при обработке коллизий, выделения и прочего.

`QRectF boundingRect()` - определяет прямоугольную область, занимаемую объектом. Данная область необходимо для быстрой перерисовки объекта в случае необходимости.

`void paint(QPainter *painter, const QStyleOptionGraphicsItem *option, QWidget *widget)` - служит для рисования объекта, именно в данном методе выполняется всё рисование.

Для корректности работы также следует явно связать унаследованное от `QObject` поле `pos` с сеттером `setPos()` и геттером `pos()` для полноценной поддержки механизма свойств в Qt.:

```
Q_PROPERTY(QPointF pos READ pos WRITE setPos)
```

Перегружать методы `pos()` и `setPos()` не требуется, но возможно.

Также, при необходимости, возможно добавить объекту обработчики событий, например `mousePressEvent`, а также сигналы и слоты. Для добавления

возможности перетаскивать объекты с помощью указателя мыши можно добавить ему соответствующий флаг: `setFlag(ItemIsMovable);`

После этого объекты описанного класса можно создавать в основной программе и связывать со сценой методом `QGraphicsScene::addItem(QGraphicsItem*)`.

Размещённые на сцене объекты в любой момент могут быть удалены с неё с помощью метода `QGraphicsScene::removeItem(QGraphicsItem*)`. Следует не забыть после этого удалить сам элемент с помощью оператора `delete`.

### ***Фильтрация событий средствами фреймворка Qt***

Фреймворк Qt поддерживает переопределение методов, отвечающих за обработку событий в дочерних классах виджетов, однако иногда требуется изменить метод обработки события для стандартного виджета, не создавая его потомка. В такой ситуации механизма слотов-сигналов может оказаться недостаточно и потребуется добавление промежуточного фильтра событий, который будет перехватывать события от конкретного виджета и изменять их обработку.

Фильтр событий — это класс, унаследованный от `QObject`, в котором переопределён метод `eventFilter`, который можно привязать к любому виджету с помощью метода `installEventFilter`.

Метод `bool eventFilter(QObject *pobj, QEvent *pe)` принимает в качестве параметров указатели на событие и объект, с которым это событие произошло.

Путём обращения к методам класса `QEvent` и использования мета-объектной систем `Qt` можно узнать как тип произошедшего события, так и тип и даже имя виджета и задать необходимые действия. Один фильтр может быть привязан к различным виджетам и событиям, однако корректнее создавать отдельные классы фильтры для каждого типа события и виджета во избежание усложнения текста программы.

## ***Объектно-ориентированный подход к графике средствами WPF***

Язык C# не предоставляет полного аналога механизма графической сцены, однако позволяет воплотить многие из её идей посредством технологии WPF и, в частности, использования элемента компоновки Canvas.

Canvas является компоновщиком и управляет размещением и отображением всех дочерних для него элементов управления. Общая стратегия использования Canvas и объектно-ориентированного подхода для отображения сложной сцены следующая:

1. На форме размещается Canvas, которому обязательно следует задать имя.

2. Создаётся класс WPF Custom Control, унаследованный от Control. Этот класс будет описывать поведение графических объектов определённого типа, размещаемых на Canvas. В данном классе помимо конструкторов обязательно необходимо реализовать метод `protected override void OnRender(DrawingContext drawingContext)`, отвечающий за рисование объекта.

3. В классе, связанном с основной формой создаются объекты класса описанного на предыдущем шаге графического объекта, их координаты на Canvas задаются методами `Canvas.SetTop(UIElement obj, double coord)` и `Canvas.SetLeft(UIElement obj, double coord)`, где `obj` - созданный объект, `coord` - расстояние до соответствующего края Canvas.

4. Графический объект добавляется на Canvas с помощью метода `MainCanvas.Children.Add(obj)`, где `MainCanvas` - имя Canvas, `obj` - имя графического объекта.

Для удаления добавленных объектов можно вызвать метод `MainCanvas.Children.Remove(obj)`, где `MainCanvas` - имя Canvas, `obj` - имя графического объекта.

Созданный класс графического объекта будут полностью поддерживать механизм событий, что позволит, например добавить реакцию на нажатие на него клавишей мыши.

Отдельно стоит отметить реализацию перетаскивания объектов в пределах Canvas, так как это потребует описания дополнительных полей и методов формы.

Первым шагом при этом будет модификация XAML разметки формы - для элемента Canvas необходимо задать обработку событий PreviewMouseMove и PreviewMouseLeftButtonUp, как показано ниже:

```
<Canvas x:Name="MainCanvas" PreviewMouseMove = "MainCanvas_PreviewMouseMove"  
PreviewMouseLeftButtonUp = "MainCanvas_PreviewMouseLeftButtonUp"/>
```

Событие PreviewMouseMove происходит во время перемещения указателя мыши, PreviewMouseLeftButtonUp - при отпускании левой кнопки мыши. Префикс Preview означает что данные события “туннелируемые”, что требуется для обработки их компоновщиком, а не его дочерними элементами, на случай если у них есть обработка таких событий. В конце обработки события обязательно необходимо установить его параметр Handled в True.

Затем необходимо создать обработчик события MouseLeftButtonDown, происходящего с каждым из графических объектов, которые планируется перемещать. Данное событие наоборот “пузырьковое” и будет обрабатываться только элементом, но не компоновщиком. В конце обработки события также обязательно необходимо установить его параметр Handled в True.

Последним подготовительным шагом является объявление в классе формы двух вспомогательных полей, хранящих ссылку на перемещаемый объект и его относительные координаты:

```
private UIElement draggedItem;  
private Point itemRelativePosition;
```

Обработчики же событий будут выглядеть следующим образом:

```
//Обработчик "захвата" графического объекта  
private void AGI_MouseLeftButtonDown( object sender, MouseEventArgs e )  
{  
    //Данный объект назначается текущим передвигаемым  
    this.draggedItem = (UIElement)sender;  
    //Сохраняется его текущая позиция  
    itemRelativePosition = e.GetPosition(this.draggedItem);  
    //Пометка что событие обработано и больше его обрабатывать не требуется
```

```

e.Handled = true;
}

//Обработчик передвижения графического объекта
private void MainCanvas_PreviewMouseMove(object sender, MouseEventArgs e)
{
    //Если передвигаемого объекта нет - закончить обработку события
    if (this.draggedItem == null)
        return;

    //Иначе - вычислить новые координаты относительно положения Canvas
    var newPos = e.GetPosition(MainCanvas) - itemRelativePosition;
    //Установить требуемое смещение
    Canvas.SetTop(this.draggedItem, newPos.Y);
    Canvas.SetLeft(this.draggedItem, newPos.X);

    //Canvas захватывает мышь во избежание случайной реакции со стороны других элементов
    MainCanvas.CaptureMouse();
    //Пометка что событие обработано и больше его обрабатывать не требуется
    e.Handled = true;
}

//Обработчик "отпускания" графического объекта
private void MainCanvas_PreviewMouseLeftButtonUp(object sender, MouseButtonEventArgs e)
{
    //Если объект в данный момент переносился
    if (this.draggedItem != null)
    {
        //Объект отпускается
        this.draggedItem = null;
        //Canvas "отпускает" мышь
        MainCanvas.ReleaseMouseCapture();
        //Пометка что событие обработано и больше его обрабатывать не требуется
        e.Handled = true;
    }
}

```

Таким образом, использование Canvas в сочетании с WPF Custom Control позволяет создавать сложные графические сцены с использованием объектно-ориентированного подхода.

### **Общая постановка задачи**

В данной работе необходимо написать программу в соответствие с индивидуальным вариантом. Все варианты описывают набор объектов для

графической сцены. Если в варианте не сказано иного, то должны использоваться объекты трёх разных классов, определяющих отображение объекта и реакцию на события.

Часть объектов являются стандартными элементами управления, в вариантах даны названия виджетов Qt, при выполнении задания с использованием технологии WPF требуется подобрать схожий по возможностям элемент управления.

В зависимости от варианта либо для каждого из классов должна быть переопределена функция обработки указанного в варианте события, либо, в случае использования фреймворка Qt, должен быть создан класс для фильтрации событий и подключён к объектам.

Необходимо реализовать возможность выбора типа объекта для добавления на сцену, добавление неограниченного числа объектов любого из указанных в задании классов на сцену, удаление объекта (в зависимости от варианта либо через список, либо через обработку события), выбор и перемещение объекта. Если в качестве объекта используется стандартный виджет, то он должен выполнять указанную в варианте функцию.

Некоторые варианты заданий могут противоречить написанному выше, в этом случае приоритет у написанного в варианте.

Работа должна быть выполнена либо с использованием фреймворка Qt, либо с использованием языка программирования C# и технологии WPF.

## **Варианты заданий.**

### ***Вариант 1***

Написать программу, позволяющую разместить на графической сцене произвольное число объектов следующих трёх типов:

1. Зелёный равносторонний треугольник
2. Надпись “Капиbara”
3. Изображение, выбираемое пользователем с помощью диалога открытия файла

Должно быть предусмотрено перемещение добавляемых объектов путём перетаскивания их левой кнопкой мыши и удаление при нажатии правой кнопкой мыши.

Обработка нажатия правой кнопки мыши должна быть реализована внутри каждого из классов объектов. Возможна реализация единого базового класса для всех трёх типов объектов и обработка нажатия правой кнопки мыши внутри него, однако каждый тип объекта должен быть представлен своим классом.

### ***Вариант 2***

Написать программу, позволяющую разместить на графической сцене произвольное число объектов следующих трёх типов:

1. Двумерная проекция тессеракта
2. QRadioButton
3. Изображение бегемота

Должно быть предусмотрено перемещение добавляемых объектов путём перетаскивания их левой кнопкой мыши.

Также должен быть представлен список из всех добавленных элементов, с возможностью выбора и удаления любого из них

С помощью фильтра событий, должна быть изменена реакция QRadioButton на нажатие на неё - помимо обычного переключения, должно изменяться её название в списке объектов.

### ***Вариант 3***

Написать программу, позволяющую разместить на графической сцене произвольное число объектов следующих трёх типов:

1. Зелёный квадрат с чёрной границей
2. Синий квадрат без границы
3. Красный круг с надписью 57 лаймового цвета в центре

Должно быть предусмотрено перемещение добавляемых объектов путём перетаскивания их левой кнопкой мыши.

С помощью обработки события нажатия клавиши мыши должно быть реализовано следующее изменение объектов при нажатии на них правой кнопкой мыши:

1. Первый объект превращается в чёрный треугольник с зелёной границей и назад после повторного нажатия
2. Второй объект превращается в флаг РФ и назад после повторного нажатия
3. Третий объект превращается в лимонно-кремовый квадрат, с надписью Х11 по нижней границе и назад после повторного нажатия

#### ***Вариант 4***

Написать программу, позволяющую разместить на графической сцене произвольное число объектов следующих трёх типов:

1. Кнопка, закрывающая окно программы
2. Написанное курсивом слово “Циркумдукция” в рамке полуночно-синего цвета
3. Портрет Артура Дента с полотенцем

Должно быть предусмотрено перемещение добавляемых объектов путём перетаскивания их левой кнопкой мыши.

С помощью обработки события нажатия клавиши мыши должно быть реализовано следующее изменение объектов при нажатии на них правой кнопкой мыши:

1. Первый объект удаляется
2. Второй объект превращается в слово “дезоксирибонуклеиновая”, записанное вертикально и назад после повторного нажатия
3. Третий объект превращается в изображение корабля Энтерпрайз (NCC-1701-C) и назад после повторного нажатия

### ***Вариант 5***

Написать программу, позволяющую разместить на графической сцене произвольное число объектов следующих трёх типов:

1. Красная кнопка, закрывающая окно программы
2. Зелёная кнопка, выводящая сообщение об ошибке
3. Фрагмент фотографии, взятой по этому адресу:

<https://disk.voenmeh.ru/index.php/s/GCtj2y66H3M1uDJ> (возможна замена по согласованию с преподавателем), лицо должно быть различимо

Должно быть реализовано следующее поведение объектов

1. Первый объект убегает от курсора
2. Второй объект хаотично перемещается по сцене
3. Третий объект привязывается к курсору

### ***Вариант 6***

Написать программу, позволяющую разместить на графической сцене произвольное число объектов следующих двух типов:

1. График синусоиды
2. Изображение станции Вавилон 5

Должно быть предусмотрено перемещение добавляемых объектов путём перетаскивания их левой кнопкой мыши.

С помощью обработки события нажатия клавиши мыши должно быть реализовано следующее изменение объектов при нажатии на них правой кнопкой мыши:

1. Первый объект превращается в график уравнения Бэтмена (<https://math.stackexchange.com/questions/54506/is-this-batman-equation-for-real>) График должен строится, а не быть заранее заданной картинкой!
2. Второй объект превращается в перечёркнутое красной линией изображение станции Глубокий Космос 9 и назад после повторного нажатия

### ***Вариант 7***

Написать программу, позволяющую разместить на графической сцене произвольное число объектов следующих трёх типов:

1. Изображение калибары
2. Счётчик (QSpinBox), ограниченный диапазоном (0;10)
3. Текст “Пригнись!”

Должно быть предусмотрено перемещение добавляемых объектов путём перетаскивания их левой кнопкой мыши.

С помощью обработки события нажатия клавиши мыши должно быть реализовано следующее изменение объектов при нажатии на них правой кнопкой мыши:

1. Первый объект не меняется, он идеален
2. Второй объект создаёт указанное число объектов первого типа в случайных позициях сцены
3. Третий объект превращается в жёлтый круг и обратно через две секунды

### ***Вариант 8***

Написать программу, позволяющую разместить на графической сцене произвольное число объектов следующих трёх типов:

1. Кнопка, закрывающая программу
2. Кнопка, добавляющая ещё один объект первого типа
3. Кнопка, убирающая все объекты первого типа

Должно быть предусмотрено перемещение добавляемых объектов путём перетаскивания их левой кнопкой мыши.

Также должен быть представлен список из всех добавленных элементов, с возможностью выбора и удаления любого из них

С помощью фильтра событий, должна быть изменена реакция кнопок на события - при нажатии левой кнопкой мыши они должны выдавать сообщение об ошибке, а свою основную функцию выполнять при нажатии правой

кнопкой мыши. Исключение - кнопка второго типа, которая должна вести себя как обычная кнопка.

### ***Вариант 9***

Написать программу, позволяющую разместить на графической сцене произвольное число объектов следующих трёх типов:

1. Расман
2. Призрак Клайд (именно этот, остальные не принимаются)
3. Жёлтый круг

Расман может быть только один. Клайдов не больше трёх, кругов любое число. Координаты Клайда указываются при добавлении объекта, остальные объекты появляются в случайных местах сцены.

Изначально все объекты статичны. После нажатие правой кнопки мыши на Расман, он становится привязан к курсору, а Клайды начинают хаотично двигаться по полю. При совпадении координат Клайда и Расман, выводится сообщение об ошибке и окно программы закрывается. При совпадении координаты Расман и жёлтого круга ничего не происходит (это всё же не игра).

При щелчке правой кнопкой мыши по жёлтому кругу он синеет.

### ***Вариант 10***

Написать программу, позволяющую разместить на графической сцене произвольное число объектов следующих трёх типов:

1. Текст, предварительно введённый пользователем (для каждого объекта свой)
2. Квадрат одного из трёх цветов: синий кадетский, тёмный лосось, помидор по выбору пользователя
3. Изображение одного из трёх греческих богов: Зевса, Посейдона или Аида

Должно быть предусмотрено перемещение добавляемых объектов путём перетаскивания их левой кнопкой мыши.

Также должен быть представлен список из всех добавленных элементов, с возможностью выбора и удаления любого из них, а также изменения бога, текста или цвета.

С помощью фильтра событий, должна быть добавлена возможность выбирать объект в списке при нажатии на него правой кнопкой мыши на сцене.

### ***Вариант 11***

Написать программу, позволяющую разместить на графической сцене произвольное число объектов следующих трёх типов:

1. QLabel с одной случайной буквой или цифрой
2. Вот это видео: <https://www.youtube.com/watch?v=dQw4w9WgXcQ>  
(ссылка для скачивания - <https://disk.voenmeh.ru/index.php/s/OpiYtNPfSJ3QUVi>)
3. Фиолетовый ромб

Должно быть предусмотрено перемещение добавляемых объектов путём перетаскивания их левой кнопкой мыши.

При нажатии правой кнопкой мыши на объект первого типа он выделяется, если после этого нажать клавишу с буквой или цифрой, то надпись на объекте меняется на соответствующую.

При нажатии правой кнопкой мыши на объект второго типа он удаляется.

Обработка данных событий должна быть реализована при помощи фильтра событий.

### ***Вариант 12***

Написать программу, позволяющую разместить на графической сцене произвольное число объектов следующих трёх типов:

1. Надпись “Опоссум спит”

2. Флаг Таиланда
3. Белый квадрат с чёрной рамкой.

У сцены в качестве фона должно быть выбранное пользователем из заранее определённого списка изображение.

Должно быть предусмотрено перемещение добавляемых объектов путём перетаскивания их левой кнопкой мыши.

С помощью обработки события нажатия клавиши мыши должно быть реализовано следующее изменение объектов при нажатии на них правой кнопкой мыши:

1. Первый объект превращается в надпись “AAAAAA” красного цвета и выдаёт сообщение “Вы разбудили опоссума”
2. Второй объект превращается в флаг Мьянмы и назад после повторного нажатия.
3. Третий объект превращается в военно-морской флаг Королевства Франции, использовавшийся с XVII в. до 1790 г на военных судах, и назад после повторного нажатия.

Флаги должны рисоваться программным способом, использование готовых изображений недопустимо

### ***Вариант 13***

Написать программу для рисования линий на графической сцене.

Использовать минимум два из трёх методов по выбору разработчика:

1. Первый щелчок мыши по сцене указывает начало линии, после этого линия рисуется вслед за указателем мыши до второго щелчка мыши.
2. Линия рисуется при нажатой клавиши мыши, следя за курсором до отпускания клавиши мыши
3. Каждое нажатие левой кнопкой мыши задаёт опорную точку для линии, после нажатия правой кнопки мыши или клавиши Escape, опорные точки соединяются ломанной линией.

Перед рисованием линии можно выбрать метод рисования и её цвет из числа заранее заданных, либо введя его числовое значение.

Каждая линия представляет собой отдельный объект, созданного разработчиком класса. После рисования, объекты добавляются в список, в котором их можно выбирать и удалять.

### ***Вариант 14***

Написать программу для рисования составных кривых Безье на графической сцене.

Использовать следующие два метода:

1. Все нечётные нажатия левой кнопкой мыши (включая первый) задают точки, через которые линия должна проходить, все нечётные - опорные точки для кривой Безье второго порядка. После нажатия правой кнопки мыши или клавиши Escape линия завершается.

2. Линия состоит из кривых Безье третьего порядка, соответственно первая, четвёртая, седьмая, десятая и т.д. нажатия задают точки, через которые должна проходить линия, а остальные - опорные точки для каждого из отрезков.

Перед рисованием линии можно выбрать метод рисования и её цвет из числа заранее заданных, либо введя его числовое значение.

Каждая линия представляет собой отдельный объект, созданного разработчиком класса. После рисования, объекты добавляются в список, в котором их можно выбирать и удалять.

### ***Вариант 15***

Написать программу, позволяющую разместить на графической сцене произвольное число объектов следующих трёх типов:

1. Квадрат весенне-зелёного цвета с чёрной границей
2. Дорожный знак “Кирпич”
3. Надпись “Налево”

Также должен быть представлен список из всех добавленных элементов, с возможностью выбора и удаления любого из них

Должен быть реализована возможность выбора объекта как в списке, так и щелчком по нему. Выбранный объект можно перемещать с помощью клавиш HJKL (рис. 5.1)

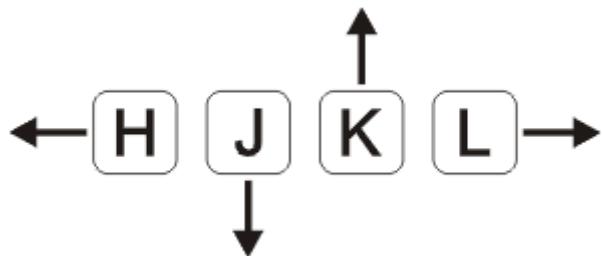


Рис. 5.1. Схема управления

Выбранный элемент должен быть обведён контуром тёмно-бордового цвета, повторяющим очертания объекта.

### ***Вариант 16***

Написать программу для рисования по точкам на графической сцене.

Реализовать два режима размещения и соединения точек - автоматический и ручной. Режимы выбираются отдельно для размещения и соединения точек.

При автоматическом режиме размещения точек, пользователь задаёт количество точек и минимальное расстояние между ними, и они размещаются на сцене в случайных местах с учётом минимального расстояния.

При ручном режиме размещения точек пользователь может нажатием левой кнопки разместить точку в любом месте сцены.

При ручном режиме соединения точек пользователь либо выбирает точки, которые хочет соединить из списка точек, либо выбирает их на сцене, и они соединяются линией.

При автоматическом режиме соединения точек пользователь указывает число линий для каждой точки и алгоритм: соединять ближайшие, соединять наиболее удалённые, соединять случайные.

### ***Вариант 17***

Написать программу, позволяющую разместить на графической сцене произвольное число объектов следующих трёх типов:

1. Изображение калибары, нарисованное программным способом (использовать внешние изображения нельзя!)
2. Красный квадрат с подписью под ним “Синий”
3. Логотип кафедры И5 с прозрачным фоном (рис. 5.2)

Ссылка для скачивания логотипа:

<https://disk.voenmeh.ru/index.php/s/osMPq6qvjsbfIIj>



Рис 5.2. Логотип кафедры И5

Должно быть предусмотрено перемещение добавляемых объектов путём перетаскивания их левой кнопкой мыши.

Также должен быть представлен список из всех добавленных элементов, с возможностью выбора и удаления любого из них

### ***Вариант 18***

Написать программу, позволяющую разместить на графической сцене произвольное число объектов следующих трёх типов:

1. Рабочие электронные часы
2. Ирландский клевер, нарисованный программным способом
3. Фраза “ЕРЕСЬ!” написанная мокасиновым цветом

Должно быть предусмотрено перемещение добавляемых объектов путём перетаскивания их левой кнопкой мыши.

Также должен быть представлен список из всех добавленных элементов, с возможностью выбора и удаления любого из них

### ***Вариант 19***

Написать программу, позволяющую разместить на графической сцене произвольное число объектов следующих трёх типов:

1. Белый квадрат
2. Вот это анимированное изображение:

<https://disk.voenmeh.ru/index.php/s/8Kk345BoeS3cgbe>

3. Изображение кролика

Должно быть предусмотрено перемещение добавляемых объектов путём перетаскивания их левой кнопкой мыши.

С помощью обработки события нажатия клавиши мыши должно быть реализовано следующее изменение объектов при нажатии на них правой кнопкой мыши:

1. Первый объект превращается в изображение испанского инквизитора и появляется сообщение “Никто не ждал испанскую инквизицию!”
2. Начинается и останавливается воспроизведение
3. Программа закрывается

### ***Вариант 20***

Написать программу, позволяющую разместить на графической сцене произвольное число объектов следующих трёх типов:

1. Изображение актёра Уильяма Хартнелла
2. Нарисованная программным способом синяя полицейская будка (рис. 5.3)

Ссылка для скачивания картинки:

<https://disk.voenmeh.ru/index.php/s/GCtj2y66H3M1uDJ>



Рис.5.3. Синяя полицейская будка

3. Вот этот аудиофайл:

<https://disk.voenmeh.ru/index.php/s/I0IVazm45ETa10a>

Должно быть предусмотрено перемещение добавляемых объектов путём перетаскивания их левой кнопкой мыши.

С помощью обработки события нажатия клавиши мыши должно быть реализовано следующее изменение объектов при нажатии на них правой кнопкой мыши:

1. Изображение на первом объекте меняется на следующее в этом списке, по завершению списка, изображение меняется на изначальное:

- а. Уильям Хартнелл
- б. Патрик Тротон
- в. Джон Пертви
- г. Том Бейкер
- д. Питер Дэвис
- е. Колин Бейкер
- ж. Сильвестр Маккой
- з. Пол Макганн
- и. Джон Харт

- к. Кристофер Экклстон
  - л. Дэвид Теннант
  - м. Мэтт Смит
  - н. Питер Капальди
  - о. Джоди Уиттакер
  - п. Дэвид Теннант
  - р. Шути Гатва
2. Второй объект перемещается в случайное место
  3. Воспроизведение начинается и останавливается

### ***Вариант 21***

Написать программу, позволяющую разместить на графической сцене произвольное число объектов следующих трёх типов:

1. Вот это изображение: (рис. 5.4)

Ссылка для скачивания картинки:

<https://disk.voenmeh.ru/index.php/s/osMPq6qvjsbfIIj>



Рис. 5.4. Изображение “За Альянс”

2. Следующий текст: “978-1616558451”

### 3. Изображение грани игрального кубика d6

Должно быть предусмотрено перемещение добавляемых объектов путём перетаскивания их левой кнопкой мыши.

С помощью обработки события нажатия клавиши мыши должно быть реализовано следующее изменение объектов при нажатии на них правой кнопкой мыши:

1. Цвет фона сцены меняется на синий и назад после повторного нажатия
2. Второй объект поворачивается на 60 градусов по часовой стрелке
3. Изображение сменяется на другую случайную грань

### ***Вариант 22***

Написать программу, позволяющую разместить на графической сцене произвольное число объектов, следующих двух типов:

1. Кнопка с надписью “Выход”
2. Программным способом нарисованное изображение целого арбуза

Должно быть предусмотрено перемещение добавляемых объектов путём перетаскивания их левой кнопкой мыши.

При нажатии правой кнопкой мыши на объект второго типа, он уничтожается, а в случайных местах сцены появляется 8 объектов, представляющих из себя программным способом нарисованное изображение дольки арбуза. При нажатии правой кнопкой мыши на дольку, она уничтожается.

Должно быть переписана обработка события закрытие окна - если на сцене ровно 256 объектов всех типов, то происходит выход из программы. В противном случае появляется окно с сообщением “Отсюда не сбежать!”. Для удобства пользователя, стоит добавить счётчик объектов, желательно в виде отдельного объекта на сцене.

### ***Вариант 23***

Написать упрощённую версию игры “Снег на голову”, используя графическую сцену:

В случайном месте верхней части сцены появляется объект, в дальнейшем называемый “Снег”, который со случайной скоростью начинает падать вниз.

В нижней части экрана расположен объект, в дальнейшем называемый “Игрок”, которым можно управлять клавишами мыши. В случае столкновения снега с игроком, выводится сообщение “Ну вооот” и приложение закрывается. В случае, если снег достиг нижней границы сцены, он перемещается в случайное место сверху экрана.

При желании, можно добавить подсчёт очков.

### ***Вариант 24***

Написать программу, позволяющую разместить на графической сцене произвольное число объектов следующих двух типов:

1. Пентаграмма тёмно-лососёвого цвета
2. Вертолёт, нарисованный программным способом

Должно быть предусмотрено перемещение добавляемых объектов путём перетаскивания их левой кнопкой мыши.

С помощью обработки события нажатия клавиши мыши должно быть реализовано следующее изменение объектов при нажатии на них правой кнопкой мыши:

1. Первый объект начинает вращаться по часовой стрелке
2. Второй объект взрывается с анимацией взрыва

### ***Контрольные вопросы***

Раскройте содержание следующих тем по выполненной работе (выбор темы зависит от использованного средства разработки):

***В случае использования Qt:***

1. Рисование с помощью QPainter.
2. Использование QGraphicsView.
3. Использование QCharts.
4. Что такое графическая сцена?
5. Создание элементов графической сцены.
6. Методы BoundingRect и Shape объектов графической сцены.
7. События в Qt.
8. Отличия событий от механизма слотов и сигналов.
9. Использование аффинных преобразований при работе с графической сценой.
10. Использование механизма слотов и сигналов с объектами графической сцены.
11. Задание цветов в Qt.
12. QPainterPath и его использование.
13. Определение взаимного расположения элементов графической сцены.
14. Флаги элементов графической сцены.

***В случае использования C#:***

1. Рисование с помощью DrawingContext.
2. Использование Canvas.
3. Создание графиков с помощью WPF Toolkit.
4. Метод OnRender.
5. Создание элементов графической сцены.
6. Метод InvalidateVisual.
7. Виды обработки событий в WPF. (Tunnel, Bubble, Direct)
8. Routed Events и их отличие от обычных событий.
9. Использование аффинных преобразований при работе с Canvas.
10. Перетаскивание элементов средствами WPF.
11. Задание цветов в WPF.

12. Path и его использование.
13. Определение взаимного расположения элементов графической сцены.
14. Отличие User Control и Custom Control в WPF.

## **ОГЛАВЛЕНИЕ**

|                                |     |
|--------------------------------|-----|
| Предисловие .....              | 3   |
| Практическая работа 1 .....    | 4   |
| Практическая работа 2 .....    | 40  |
| Практическая работа 3 .....    | 62  |
| Практическая работа 4 .....    | 77  |
| Практическая работа 5 .....    | 112 |
| Библиографический список ..... | 137 |

## БИБЛИОГРАФИЧЕСКИЙ СПИСОК

*Шлее М.* Qt 5.10. Профессиональное программирование на C++. – СПб.: БХВ. 2018. – 1072 с.

*Lee Zhi Eng* Hands-On GUI Programming with C++ and Qt5. – England.: Packt. 2018 – 404 с.

*Lazar G., Penea R.* Mastering Qt 5 - Second Edition. – England.: Packt. 2018 – 534 с.

*Троелсен Э., Джепикс Ф.* Язык программирования C# 9 и платформа .NET 5: основные принципы и практики программирования. - 10-е изд. - М: Диалектика- Вильямс, 2022. - 1392 с.

*Adam Nathan.* Windows Presentation Foundation Unleashed. - USA: Sams Publishing, 2006. – 638 с.

*Petzold, C.* Applications = Code + Markup: A Guide to the Microsoft Windows Presentation Foundation. - USA: Microsoft Press, 2006. - 1024 с.

*Sells C.* Windows Forms Programming in C#. - USA: AddisonWesley, 2003. - 734 с.

*Litvinavicius T.* Exploring Windows Presentation Foundation: With Practical Applications in .NET 5. - USA: Apress, 2020. - 236 с.

Что такое Windows Presentation Foundation - WPF .NET Microsoft Docs.  
– URL:<https://docs.microsoft.com/ru-ru/dotnet/desktop/wpf/overview/>

Qt Documentation. – URL: <https://doc.qt.io/>

*Вальштейн Константин Владимирович*

*Бармина Анастасия Александровна*

*Магомедов Ибрагим Набиуллаевич*

**Визуальное программирование: практикум**

Редактор

Подписано в печать 2023. Формат бумаги Бумага

Печать Усл. печ. л. Уч.-изд. л. Тираж экз. Заказ №

Балтийский государственный технический университет

Типография БГТУ

190005, С-Петербург, 1-я Красноармейская ул., д. 1