Урок 3

1. Переопределение слотов и простейшая графика

Кроме создания новых слотов и использования уже готовых, у наследников класса QWidget существует возможность изменять логику обработки некоторых сигналов. Изменения достигаются путем переопределения методов родительского класса. К часто встречающимся кейсам можно отнести следующие методы:

* paintEvent() – использование метода update или repaint(~~не рекомендуется~~) у экземпляра;
* mouseMoveEvent() – движение мыши;
* mousePressEvent() – нажатие кнопки мыши;
* mouseDoubleClickEvent() – двойной клик кнопки мыши;
* mouseReleaseEvent() – отпускание кнопки мыши;
* keyPressEvent() – нажатие клавиши;
* keyReleaseEvent() – отпускание клавиши;
* resizeEvent() – изменение размеров виджета.

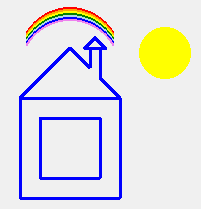
С этими и другими методами можно ознакомиться в [документации](https://doc.qt.io/qt-5/qwidget.html#events). Следует заметить, что в каждый метод как аргумент передается специализированный Event.

Рассмотрим пример переопределения логики на приложении, которое, отслеживая положение курсора, рисует под ним красный круг.

Для рисования в Qt существует специальный класс QPainter. Он создает растровые изображения, являющиеся частью заднего фона виджета. Сразу стоит заметить, что для получения результата на виджете не должно быть дополнительных эффектов стилизации заднего фона. Возможный вариант использования класса выглядит следующим образом:



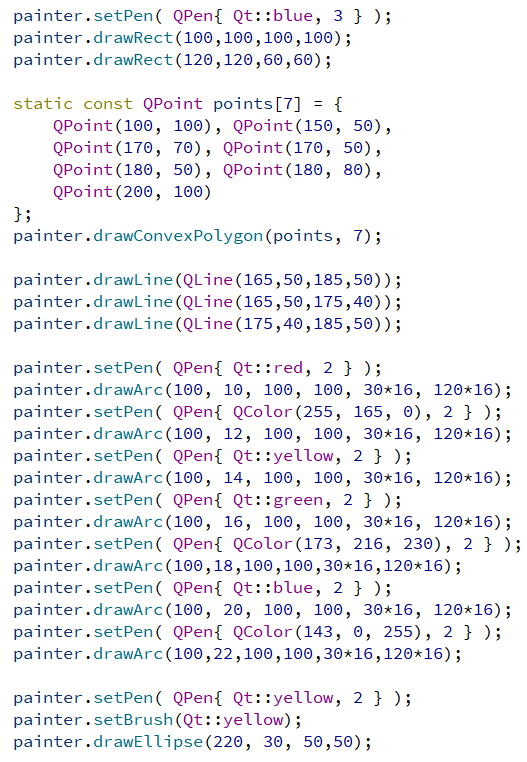
Результат выполнения кода:



К этапам использования класса QPainter относятся:

* Инициализация экземпляра класса QPainter;
* Создание кисти;
* Присваивание кисти и цвета фона фигуры экземпляру класса QPainter;
* Вызов методов для добавления элементов на фон.

В создании изображений используются экземпляры класса QPen и QColor, которые олицетворяют кисть и цвет. Существует множество способов их инициализации. Ниже представлен пример более краткого создания этого же изображения:



У QPainter существуют вспомогательные методы, которые позволяют поменять изображение не изменяя значений, используемых при создании изображения. Например, положение рисунка сместится вправо на 100 пикселей, если перед рисунком использовать следующий вызов:



Аналогичным способом, используя метод rotate(), можно повернуть изображение.

Разобравшись в том, как создавать собственные растровые изображения в Qt, осталось только написать код программы. Если создавать программу на основе пустого проекта, файл mainwindow.h будет выглядеть следующим образом:



В 5-7 строках подключаются нужные для работы классы. В 20-21 строке объявляется переопределение методов, которые нужны для реагирования на действия пользователя. В 25 строке объявлена переменная clickPoint, которая будет хранить положение мыши. С помощью нее будет происходить передача состояния из mouseMoveEvent() в pointEvent().

Реализация методов сводится к получению позиции курсора и созданию рисунка:

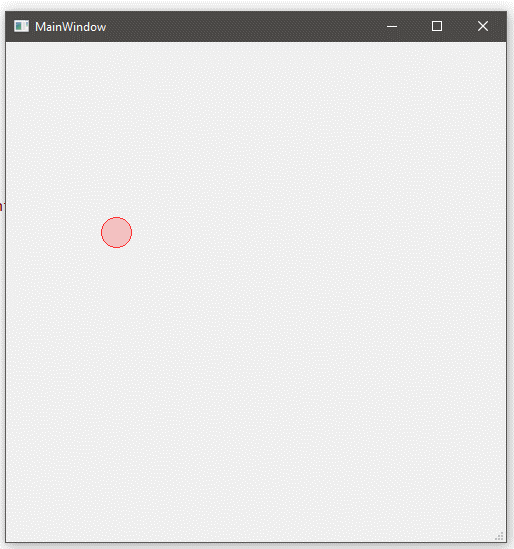


В 9 и 10 строке производятся важные для работы программы изменения поведения виджетов по отношению к действиям пользователя. Для центрального виджета устанавливается состояние, в котором он будет прозрачным для действий пользователя и передаст события родительскому элементу. Метод setMouseTracking() вызывается для включения постоянного реагирования на действия мыши на слое. Если убрать вызовы, то mouseMoveEvent() будет срабатывать только, когда нажата любая клавиша мыши. Для понимания “родственных отношений” слоев в пустом проекте следует изучить древо элементов в файле mainwindow.ui.

Далее в 15 строке описывается поведение программы при движении мыши. В переменную clickPoint сохраняются координаты, на которых произошло событие и вызывается метод update(), производящий событие перерисовки компонента.

В 20 строке реализован метод pointEvent(). Стоит отметить, что любые использования QPainter за пределами этого метода будут приводить к ошибкам и отсутствию результата. Проверка на пустой clickPoint сделана не просто так. Метод pointEvent() вызывается при первой отрисовке и перерисовке, даже если ее инициировал не код разработчика, поэтому эта тривиальная проверка позволит избежать ошибок при первом рендере интерфейса.

Результат работы приложения может выглядеть следующим образом:



То, что приложение самостоятельно убирает старые рисунки при нанесении новых, удобно в разработке динамических анимаций, но бывают ситуации, в которых состояние заднего фона должно изменятся без очищения. Для таких случаях используется класс QPixmap.

Для изменения приложения до уровня простой программы при рисовании одной кистью следует изменить файл mainwindow.h:



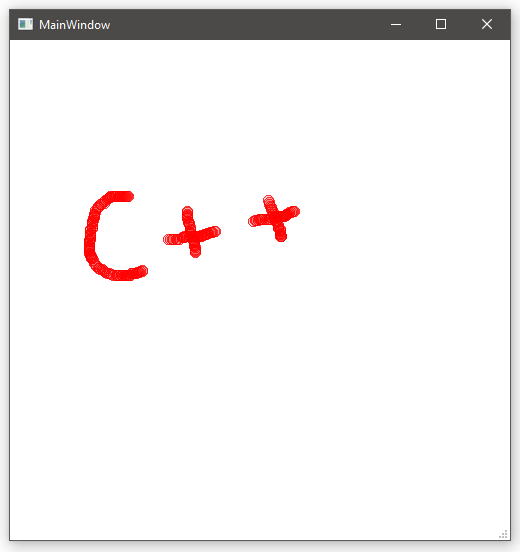
В класс добавлено поле типа QPixmap для хранения рисунка. Эту переменную можно представить, как холст, хранящий все изменения, который будет рисоваться на фоне виджета.

Изменения файла mainwindow.cpp не велики:



В 10-11 строках инициализируется холст для хранения рисунков, а в методе paintEvent() теперь существуют два экземпляра QPainter, каждый из которых отвечает за свою часть ранее описанной логики – изменения на холсте и изменения на фоне виджета соответственно.

Возможный результат работы программы:



1. Работа с изображениями

Основная, но не единственная, задача QPixmap – преобразовывать изображения для использования их в приложении. Кроме него можно использовать следующие классы:

**QImage**

QPixmap не может предоставлять операции на уровне пикселей, в то время как QImage позволяет это.

**QBitmap**

QBitmap наследуется от QPixmap и в основном используется для отображения монохромных растровых изображений. Это подкласс QPixmap, поэтому имеет все его характеристики. Глубина цвета QBitmap всегда равна 1. Понятие глубины цвета происходит из компьютерной графики и является числом двоичных цифр, используемых для представления цветов. Один бит имеет только два состояния: 0 и 1, поэтому он имеет два цвета, черный и белый. Простыми словами, QBitmap – это черно-белое состояние изображения.

**QPicture**

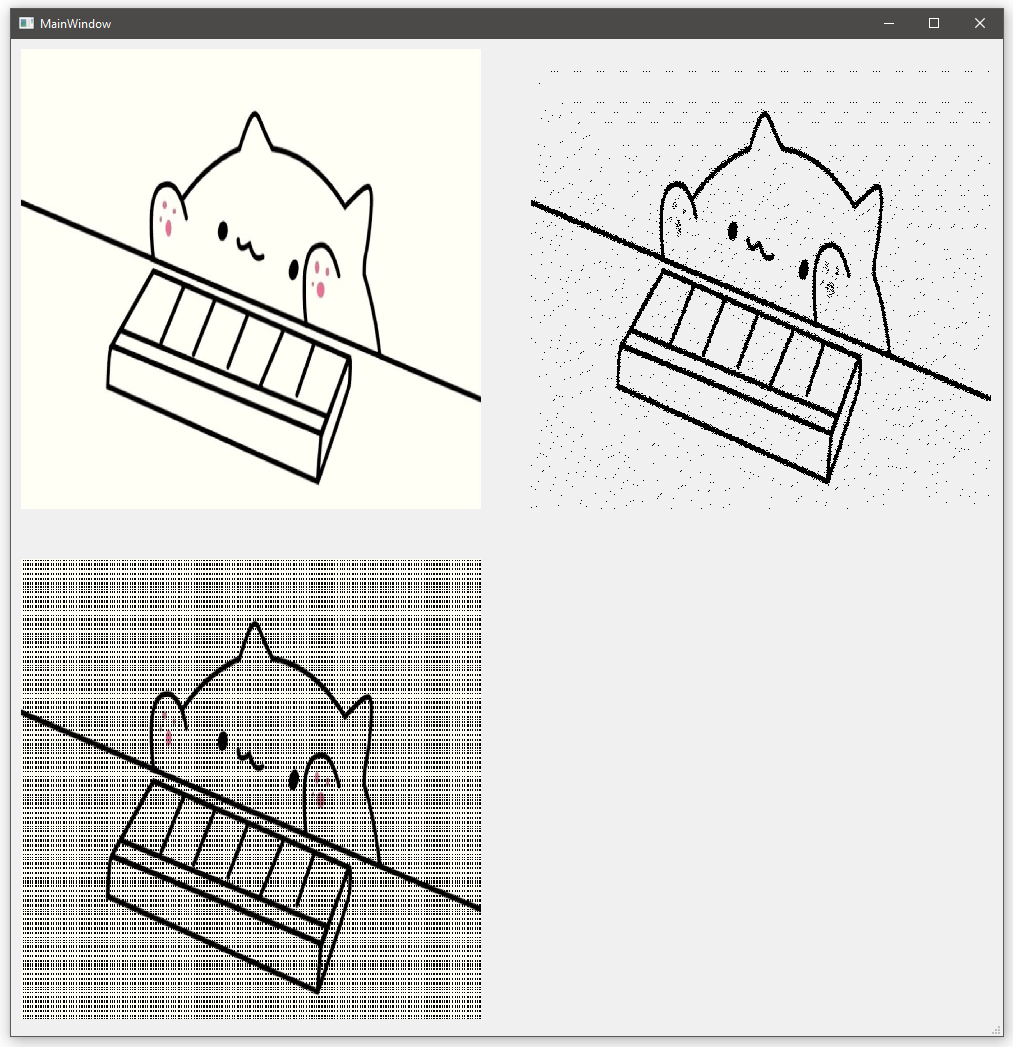
QPicture – это устройство для рисования, которое может записывать и воспроизводить команды QPainter. QPicture сериализует команды QPainter для устройства ввода-вывода и сохраняет их в виде независимого от платформы двоичного формата файла .pic, который не имеет ограничений на содержание.

QPicture используется в тех случаях, где важной частью программы является сохранение результатов работы QPainter. В остальных же случаях удобнее использовать предыдущие варианты.

Для отображения одного изображения тремя способами понадобится следующий код:



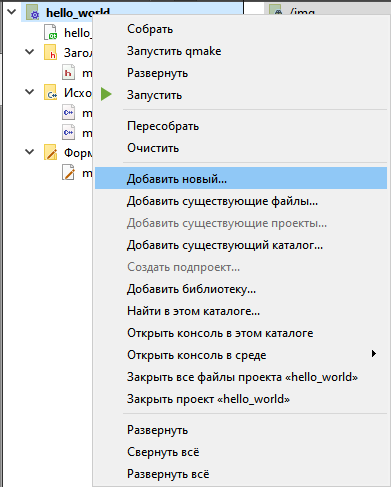
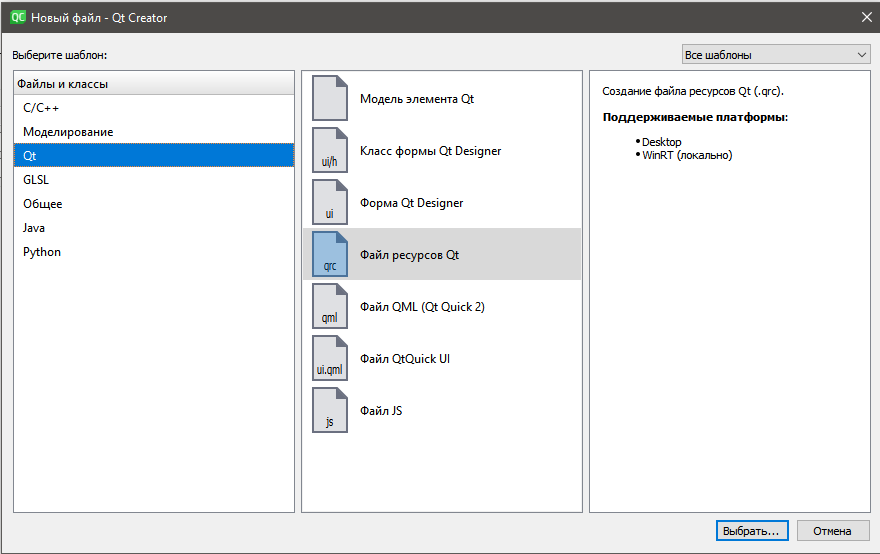
Результат работы программы:



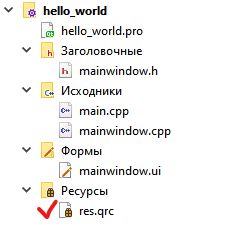
1. Ресурсы проекта

Кроме программного метода создания картинок существует также и возможность добавить ее через редактор интерфейса, но для этого изображение нужно подключить как ресурс проекта.

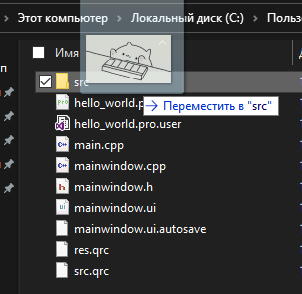
Создать новый файл ресурсов можно через контекстное меню проекта. “Добавить новый” > “Файл ресурсов Qt”:

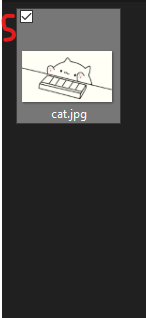
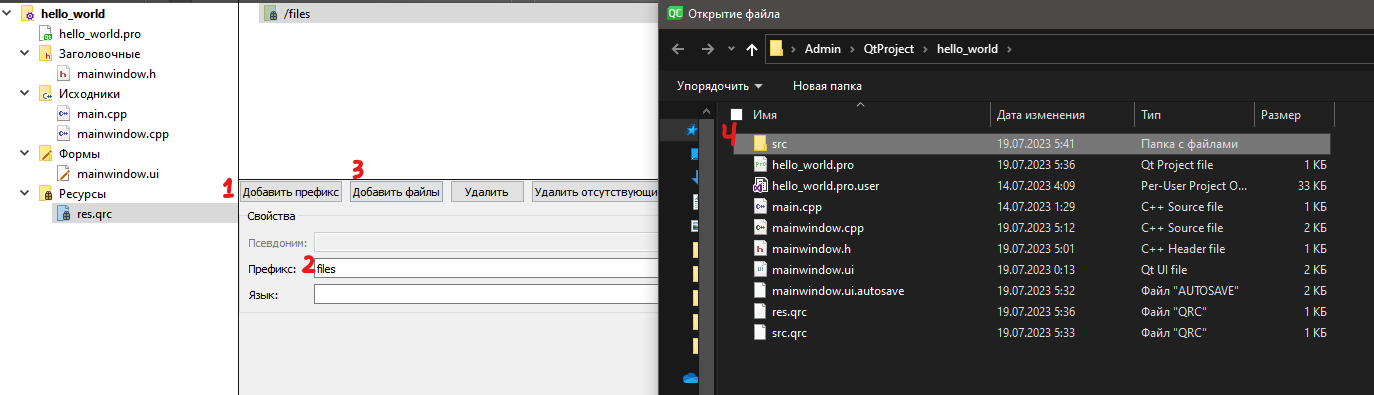
В древе проекта создастся файл фармата .qrc:



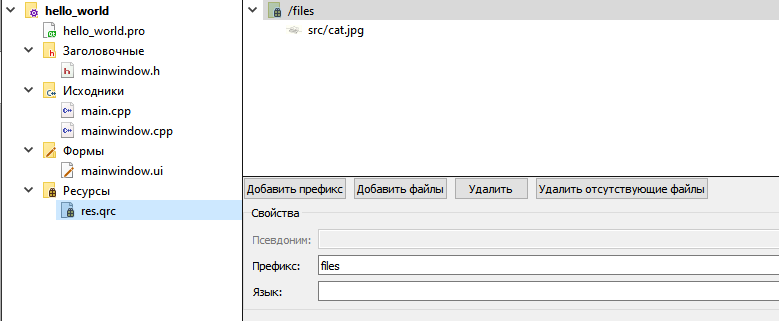
Далее нужно создать папку в директории проекта, в которой будут храниться ресурсы программы и перенести туда изображение:



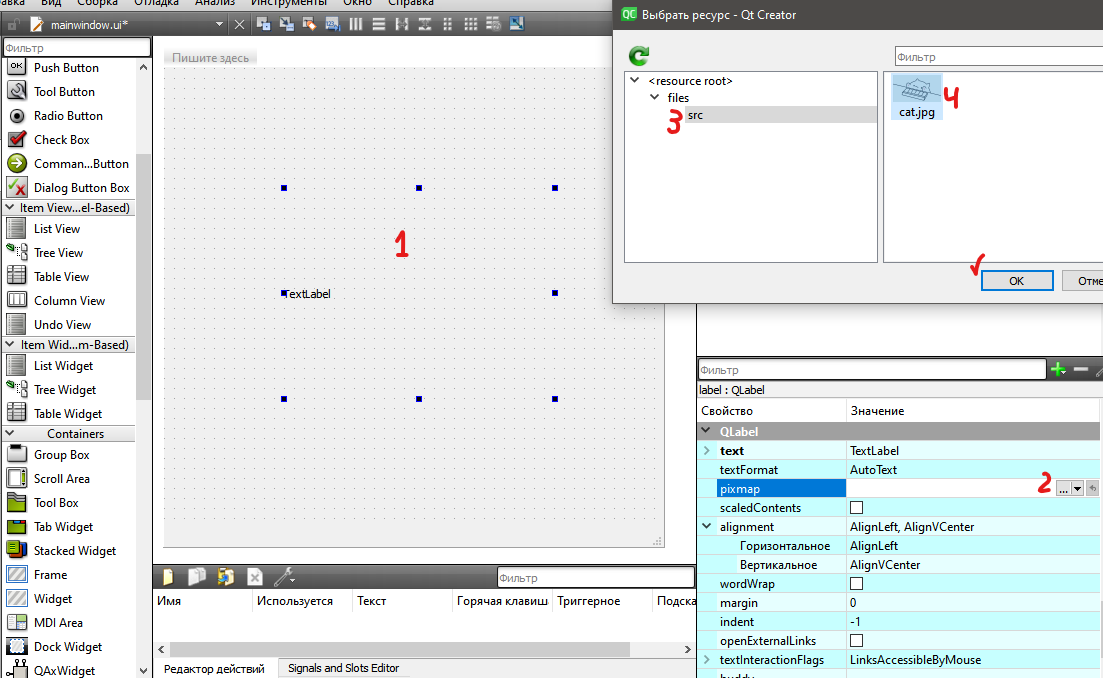
Теперь осталось подключить изображение к файлу ресурсов. Это можно сделать через добавление префикса во вкладке файла ресурсов:



Итоговый вид файла ресурсов будет следующим:



В Qt не существует специального компонента для отображения картинок. Воспользоваться можно простым текстовым полем Label. Подключение изображения производится через свойство pixmap.



Если изображение не помещается в контейнер, то следует установить свойство scaledContents в состояние true.

Результат – добавленная в приложение картинка:

