Глава 16

Прямоугольники и эллипсы, режим прозрачности: проект PNGEDIT3

Проект PNGEDIT3 продолжает серию проектов, посвященных работе с графической библиотекой Qt. В этом проекте реализуется настройка фонового цвета (класс QBrush) и рисование прямоугольников и эллипсов (методами класса QPainter) в обычном режиме и в режиме прозрачности, т. е. без заливки внутренности фигур фоновым цветом.

16.1. Настройка фонового цвета

В качестве заготовки для проекта PNGEDIT3 следует использовать ранее разработанный проект PNGEDIT2 *(см. главу 15).* Разместите в форме Form новую метку label\_5 типа ColorLabel и настройте ее свойства (листинг 16.1).

Используя кнопку **Переместить назад**, переместите левую верхнюю часть метки label\_5 под метку label\_3, как указано на рис. 16.1.

В класс Form добавьте новое поле:

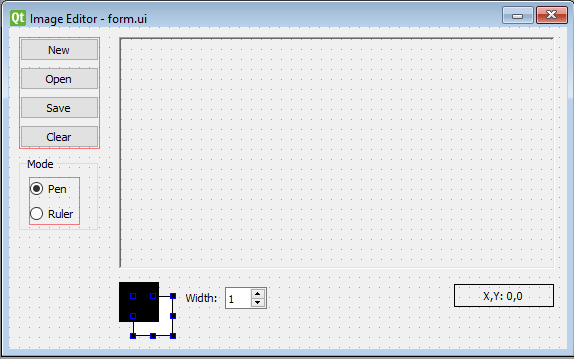
brush = QBrush(Qt.white)

Определите обработчик on\_label\_5\_backColorChanged и измените слот clear (листинг 16.2).

**Листинг 16.1. Настройка свойств**

label\_5: text = пустая строка, frameShape = **Box**, styleSheet = **background-color: white;**,

width = **40**, height = **40**



**Рис. 16.1.** Вид формы Form для проекта PNGEDIT3 на начальном этапе разработки

**Листинг 16.2. Слоты** **on\_label\_5\_backColorСhanged и** **clear**

**def on\_label\_5\_backColorChanged(self):**

**self.brush.setColor(self.label\_5.palette().color(QPalette.Background))**

def clear(self):

pix = self.label.pixmap()

**pix.fill(self.brush.color())**

self.label.setPixmap(pix)

**Результат.** Добавленная в форму метка label\_5 указывает текущий фоновый цвет, т.е. цвет инстру­мента рисования *кисть* (brush), используемого для заливки внутренней области замкнутых фигур. Этот цвет можно изменить с помощью диалогового окна **Select Color**, вызвав его на экран щелчком мыши на метке label\_5. Новое значение фонового цвета проявляется пока только при выполнении команды **Clear**: закрашивание рисунка теперь проводится не белым, а текущим фоновым цветом. При создании или загрузке из файла нового изображения текущее значение фонового цвета сохраняется.

16.2. Третий режим рисования: прямоугольники

В компоненте groupBox внутри менеджера компоновки разместите еще одну радиокнопку (radioButton\_3) и присвойте ее свойству text значение **&Figure**. Кроме того, свяжите слот setMode с ее сигналом toggled.

В класс Form добавьте метод drawFigure (листинг 16.3) и измените функцию reversibleDraw (листинг 16.4). Измените метод mouseRelease (листинг 16.5).

**Листинг 16.3. Метод** **drawFigure**

**def drawFigure(self, rect, painter):**

**painter.fillRect(rect, self.brush)**

**w = self.pen.width()**

**painter.drawRect(QRect(rect.x() + w / 2, rect.y() + w / 2,**

**rect.width() - w, rect.height() - w))**

**Листинг 16.4. Новый вариант метода** **reversibleDraw**

def reversibleDraw(self):

painter = QPainter(self.label.pixmap())

painter.setCompositionMode(QPainter.RasterOp\_NotSourceXorDestination)

**if self.mode == 1:**

painter.drawLine(self.startPt, self.movePt)

**else:**

**painter.setPen(QPen(Qt.black, 2, Qt.SolidLine))**

**painter.drawRect(QRect(self.startPt, self.movePt))**

self.label.repaint()

**Листинг 16.5. Новый вариант метода** **mouseRelease**

def mouseRelease(self, event):

if self.startPt == self.nullPt:

return

**if self.mode >= 1:**

self.label.setPixmap(self.pix)

painter = QPainter(self.label.pixmap())

painter.setPen(self.pen)

**if self.mode == 1:**

painter.drawLine(self.startPt, self.movePt)

**else:**

**self.drawFigure(QRect(self.startPt, self.movePt), painter)**

self.label.repaint()

**Результат.** В новом режиме, который устанавливается выбором радиокнопки **Figure** (Фигура) или клавиатурной комбинацией <Alt>+<F>, можно рисовать прямоугольники. Способ рисования похож на рисование прямой линии: надо установить курсор мыши в позицию одного из углов прямоугольника, нажать левую кнопку мыши и переместить курсор в позицию противоположного по диагонали угла прямоугольника (при перемещении мыши контур прямоугольника рисуется инверсным цветом). После отпускания кнопки мыши прямоугольник окончательно рисуется текущим цветом линии, а его внутренность закрашивается текущим фоновым цветом. Ширина границы, как и толщина линии в режимах **Pen** и **Ruler**, определяется значением компонента spinBox. См. также комментарии 1—3.

**Недочет.** Если толщина пера превышает 1, то полученный прямоугольник будет иметь размеры, превышающие размеры инверсного контура, рисуемого при перемещении мыши. Это связано с тем, что по умолчанию "толстый" контур замкнутых фигур рисуется по обе стороны от их границы.

**Исправление.** В конструктор класса Form добавьте оператор

**self.pen.setJoinStyle(Qt.MiterJoin)**

**Результат.** Теперь рисование контура выполняется внутри границы фигуры; таким образом, инверсный контур показывает правильные размеры фигуры независимо от текущей толщины пера (см. также комментарий 4).

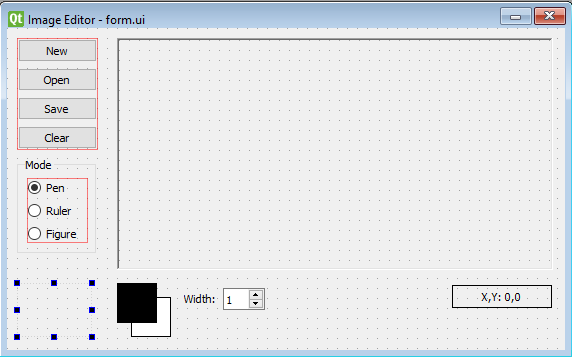
Комментарии

1. Для рисования контура прямоугольника и заливки его внутренней области предназначены два различных метода класса QPainter: drawRect и fillRect. Первым параметром этих методов является прямоугольник, на котором будет происходит рисование. Заметим, что аналогичным образом реализовано рисование контура других фигур: за это действие отвечает метод, начинающийся со слова draw, а вот для заливки всегда используется fillRect. В методах типа draw применяется перо графической канвы, для fillRect кисть требуется указывать явно. При совместном использовании этих методов первым следует вызывать контур, ранее нарисованный методом draw. В нашей программе вызов обоих методов выполняется во вспомогательном методе drawFigure (листинг 16.3).
2. При рисовании контура требуется учитывать толщину пера. Левая верхняя точка сдвигается на половину толщины пера по каждой координате, размерности прямоугольника уменьшаются на полную толщину. Это было выполнено в методе drawFigure (листинг 16.3).
3. Для рисования контура в инверсном режиме применяется перо для рисования сплошного контура ширины 2 (в конструкторе пера используется стиль пера по умолчанию Qt.SolidLine). Тонкий пунктирный контур (стиль Qt.DashLine) будет использован нами в дальнейшем при рисовании эллипсов *(см. разд. 16.3)*.
4. По умолчанию свойство joinStyle() пера имеет значение Qt.BevelJoin (перечислимый тип PenJoinStyle кроме указанных значений содержит еще варианты Qt.RoundJoin и Qt.SvgMiterJoin). При выбранном нами стиле внешние края линий расширены, в отличие от стиля по умолчанию, при использовании которого между линиями образуется и заполняется треугольная зазубрина.

16.3. Рисование эллипсов

Разместите в форме новую метку label\_6, очистите её свойство text и положите свойство alignment.Horizontal равным AlignHCenter. Метка label\_6 должна располагаться в левом нижнем углу формы (см. рис. 16.2, на котором метка label\_6 является текущим компонентом, т.е. обрамляется синими маркерами). Сохраните изменения и откройте форму в текстовом редакторе. Добавьте новый кастомный класс FigureLabel – наследник QLabel. Измените тип label\_6 на FigureLabel. Создайте класс FigureLabel в файле figurelabel.py (листинг 16.6).

В класс Form добавьте новое поле:



**Рис. 16.2.** Вид формы Form для проекта PNGEDIT3 на промежуточном этапе разработки

Измените методы drawFigure и reversibleDraw (листинг 16.7) и определите слоты on\_label\_6\_mousePressed и on\_label\_6\_painted (листинг 16.8).

Кроме того, в методы on\_spinBox\_valueChanged, on\_label\_3\_backColorChanged и on\_label\_5\_backColorChanged добавьте оператор

**self.label\_6.repaint()**

**Листинг 16.6. Класс FigureLabel**

**from PyQt5.QtCore import \***

**from PyQt5.QtGui import \***

**from PyQt5.QtWidgets import \***

**class FigureLabel(QLabel):**

**mousePressed = pyqtSignal()**

**painted = pyqtSignal()**

**def mousePressEvent(self, event):**

**self.mousePressed.emit()**

**def paintEvent(self, event):**

**self.painted.emit()**

**Листинг 16.7. Новый вариант методов** **drawFigure** и **reversibleDraw**

def drawFigure(self, rect, painter):

~~painter.fillRect(rect, self.brush)~~

w = self.pen.width()

**painter.setPen(self.pen)**

**painter.setBrush(self.brush)**

**r = QRect(rect.x() + w / 2, rect.y() + w / 2,**

**rect.width() - w + 1, rect.height() - w + 1)**

**if self.figureMode == 0:**

**painter.drawRect(r)**

**elif self.figureMode == 1:**

**painter.drawEllipse(r)**

def reversibleDraw(self):

painter = QPainter(self.label.pixmap())

painter.setCompositionMode(QPainter.RasterOp\_NotSourceXorDestination)

if self.mode == 1:

painter.drawLine(self.startPt, self.movePt)

else:

**painter.setPen(QPen(Qt.black, 2 - self.figureMode, self.figureMode + 1))**

painter.drawRect(QRect(self.startPt, self.movePt))

self.label.repaint()

**Листинг 16.8. Слоты on\_label\_6\_mousePressed и on\_label\_6\_painted**

**def on\_label\_6\_painted(self):**

**painter = QPainter()**

**painter.begin(self.label\_6)**

**r = self.label\_6.rect()**

**self.drawFigure(QRect(r.x(), r.y(), r.width(), r.height()), painter)**

**painter.end()**

**def on\_label\_6\_mousePressed(self):**

**self.radioButton\_3.setChecked(True)**

**self.figureMode = (self.figureMode + 1) % 2**

**self.label\_6.repaint()**

**Результат.** При щелчке мышью на метке-образце label\_6 изменяется нарисованная на ней фигура (прямоугольник переходит в эллипс, эллипс — в прямоугольник) и происходит автоматический переход в режим **Figure**. В этом режиме теперь рисуется та фигура, которая изображена на метке-образце (для рисования контура эллипса используется метод drawEllipse класса QPainter). Метка label\_6 также отображает текущие характеристики инструментов рисования: цвет, толщину пера и цвет кисти. При рисовании эллипсов используются тонкие штриховые инверсные контуры (в отличие от более толстых сплошных инверсных контуров, отображаемых при рисовании прямоугольников). См. также комментарии 1 и 2.

**Недочет.** Если толщина пера равна 1, то в метке-образце label5 не отображается правая и нижняя граница прямоугольника (а также правый и нижний фрагмент границы эллипса). Это объясняется имеющейся в Qt так называемой *ошибкой смещения на 1 пиксел* которая, в частно-

сти, проявляется в том, что контуры прямоугольников и эллипсов рисуются

не в прямоугольной области, указанной в качестве второго параметра методов

DrawRectangle и DrawEllipse, а в области, большей указанной на 1 пиксел (по

обоим измерениям).