Глава 15

Цветное перо и прямые линии: проект PNGEDIT2

Проект PNGEDIT2 продолжает серию проектов, посвященных работе с графической библиотекой Qt. В этом проекте реализуется рисование цветным пером произвольной ширины (классы QPen и QColorDialog) и рисование прямых линий с возможностью изображения текущего положения рисуемой линии инверсным цветом.

15.1. Рисование цветным пером

В качестве заготовки для проекта PNGEDIT2 следует использовать ранее разработанный проект PNGEDIT1 *(см. главу 14)*. Скопируйте проект PNGEDIT1 в новый каталог PNGEDIT2 и выполните действия, необходимые для переименования проекта. (см. разд. 1.1). Разместите в форме Form две новые метки (label\_3 и label\_4), компонент типа QSpinBox (spinBox). Закройте Qt Designer и откройте файл формы в текстовом редакторе. Добавьте кастомный виджет ColorLabel (наследник класса QLabel) и измените тип метки label\_3 на ColorLabel. Сохраните изменения и вновь запустите Qt Designer. Настройте свойства добавленных визуальных компонентов (листинг 15.1) и расположите их в соответствии с рис. 15.1. Создайте класс ColorLabel в файле colorlabel.py и подключите его в файле form.py (листинг 15.2). Добавьте в класс Form реализацию двух слотов (благодаря префиксу их не нужно подключать в конструкторе): on\_label\_3\_backColorСhanged и on\_spinBox\_valueChanged (листинг 15.3).

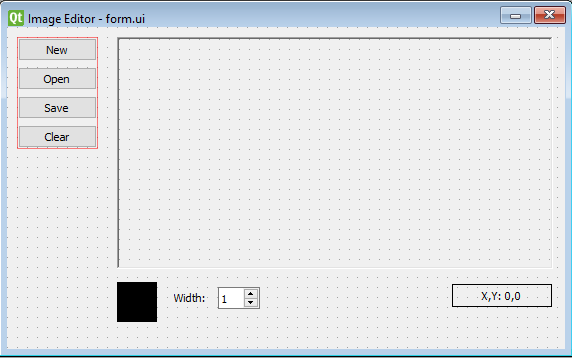
**Листинг 15.1. Настройка свойств**

label\_3: text = пустая строка, frameShape = **Box**, styleSheet = **background-color: black;**,

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

label\_4: text = **Width:**

spinBox: maximum = **22**, minimum = **1**, value = **1**, singleStep = **3**



**Рис. 15.1.** Вид формы Form для проекта PNGEDIT2 на начальном этапе разработки

**Листинг 15.2. Класс ColorLabel**

**from PyQt5.QtCore import \***

**from PyQt5.QtGui import \***

**from PyQt5.QtWidgets import \***

**class ColorLabel(QLabel):**

**backColorChanged = pyqtSignal()**

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

**color = QColorDialog.getColor()**

**if color.isValid():**

**self.setStyleSheet('background-color: rgb({},{},{})'**

**.format(color.red(), color.green(), color.blue()))**

**self.backColorChanged.emit()**

**Листинг 15.3. Слоты** **on\_label\_3\_backColorСhanged** **и** **on\_spinBox\_valueChanged**

**def on\_label\_3\_backColorChanged(self):**

**self.pen.setColor(self.label\_3.palette().color(QPalette.Background))**

**def on\_spinBox\_valueChanged(self, value):**

**self.pen.setWidth(int(value))**

**Результат.** При рисовании можно выбирать:

* *цвет пера*, щелкая мышью на метке label\_3, фон которой соответствует текущему цвету пера; в результате на экране появляется диалоговое окно Цвет, позволяющее выбрать требуемый цвет (при закрытии данного окна кнопкой OK фон метки label\_3 закрашивается выбранным цветом);
* *толщину пера*, устанавливая ее значение с помощью компонента spinBox (допустимыми являются значения от 1 до 22).

Установив большую толщину и белый цвет линии, можно стирать элементы изображения. При создании или загрузке нового изображения настройки пера сохраняются. См. также комментарий.



**Рис. 15.2.** Пример линии большой толщины до исправления недочета *(слева)* и после исправления *(справа)*

**Недочет.** При рисовании линий большой толщины результат оказывается неудовлетворительным (рис. 15.2, *слева*).

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

**self.pen.setCapStyle(Qt.RoundCap)**

**Результат.** Теперь толстые линии рисуются надлежащим образом (рис. 15.2, справа). См. также комментарий 2.

Комментарии

1. Обратите внимание на «разделение обязанностей» обработчиков для метки **label\_3**: обработчик события mousePressEvent меткиColorLabel отвечает за выбор цвета с помощью диалогового окна и изменение фона метки в соответствии с выбранным цветом, а обработчик on\_label\_3\_backColorChanged отвечает за изменение цвета пера при изменении цвета фона метки. В дальнейшем нам может потребоваться выполнять дополнительные действия, связанные с изменением текущего цвета пера; эти действия будет достаточно указать в обработчике on\_label\_3\_backColorChanged. Кроме того, цвет пера может быть изменен не только с помощью диалогового окна **Select Color**, а, например, с помощью инструмента «пипетка» (см. разд. 17.3). В этом случае нам будет достаточно присвоить новый цвет через свойство styleSheet, и это автоматически приведет к выполнению всех действий, связанных с изменением цвета пера.
2. Отмеченный недочет был вызван неудачной формой концов линий. По умолчанию концы линий делаются прямоугольными (Qt.SquareCap), поэтому при любом изменении направления рисования неизбежно возникают изломы. В нашей ситуации естественнее использовать закругленные концы линий; для этого достаточно в конструкторе формы установить новое значение Qt.RoundCap для свойства capStyle. Значения принадлежат перечислению Qt.PenCapStyle.

15.2. Второй режим рисования: прямые линии

Разместите в форме компонент-контейнер типа QGroupBox (groupBox) и присвойте его свойству text значение **Mode**. В компоненте groupBox разместите две радиокнопки (radioButton и radioButton\_2), скомпонуйте их по вертикали и присвойте их свойствам text значения **&Pen** и **&Ruler** соответственно (рис. 15.3). Кроме того, поставьте галочку в свойстве checked радиокнопки radioButton.

В класс Form добавьте три новых поля и подключите модуль sys:

**mode = 0**

**movePt = QPoint()**

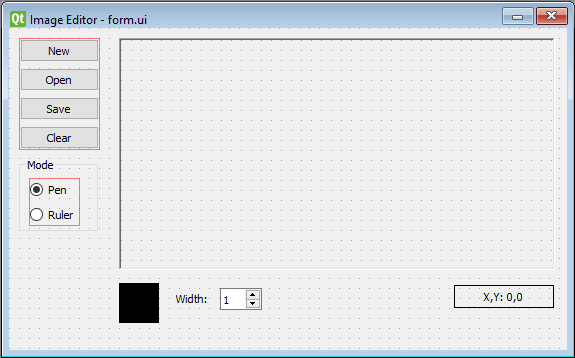
**nullPt = QPoint(sys.maxsize, 0)**

Определите в классе Form новый вспомогательный метод reversibleDraw (листинг 15.4) и обработчик сигнала toggled setMode (листинг 15.5) радиокнопок radioButton и radioButton\_2, после чего свяжите созданный слот с сигналом toggled в конструкторе.

Определите сигнал mouseReleased и обработчик события mouseReleaseEvent для класса ImageLabel (листинг 15.6), слот mouseRelease для формы Form (свяжите данный слот с сигналом mouseReleased в конструкторе формы) и измените слоты mousePress и mouseMove (листинг 15.7).

Наконец, в начало слотов open и save добавьте оператор

**self.startPt = self.nullPt**



**Рис. 15.3.** Окончательный вид формы Form для проекта PNGEDIT2

**Листинг 15.4. Метод reversibleDraw класса Form**

**def reversibleDraw(self):**

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

**painter.setCompositionMode(QPainter.RasterOp\_NotSourceXorDestination)**

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

**self.label.repaint()**

**Листинг 15.5. Слот setMode**

**def setMode(self):**

**rb = self.sender()**

**if not rb.isChecked():**

**return**

**self.mode = self.verticalLayout\_2.indexOf(rb)**

**Листинг 15.6. Слот 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)**

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

**self.startPt = event.pos()**

**self.label.repaint()**

**Листинг 15.7. Новый вариант методов mousePress и mouseMove**

**с**

def mousePress(self, event):

self.pix = QPixmap(self.label.pixmap())

**self.movePt =** self.startPt = event.pos()

def mouseMove(self, event):

self.label\_2.setText('X,Y: {},{}'.format(event.x(), event.y()))

**if self.startPt == self.nullPt:**

**return**

if event.buttons() == Qt.LeftButton:

**if self.mode == 0:**

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

painter.setPen(self.pen)

painter.drawLine(self.startPt, event.pos())

self.startPt = event.pos()

self.label.repaint()

**else:**

**self.label.setPixmap(self.pix)**

**self.movePt = event.pos()**

**self.reversibleDraw()**

**Результат.** Теперь рисование можно выполнять в двух режимах: в режиме **Pen** (Перо), как раньше, рисуются линии произвольной формы, а в новом режиме **Ruler** (Линейка) рисуются прямые линии. Рисование прямых линий производится следующим образом: левая кнопка мыши нажимается в начальной точкелинии и затем, при нажатой кнопке, мышь перемещается к конечной точке, где кнопка мыши отпускается. При перемещении мыши на рисунке изображаетсятекущее положение линии (для этого используется вспомогательная линия толщины 1, цвет которой является *инверсным* по отношению к цвету под линией).Окончательно (текущим цветом и текущей толщиной) линия рисуется в момент отпускания кнопки мыши. Для рисования ломаной линии надо после отпускания кнопки мыши сразу нажать ее еще раз, и повторять этот процесс длякаждого звена ломаной. Переключение между режимами рисования осуществляется радиокнопками radioButton и radioButton\_2: радиокнопка, соответствующая текущему режиму, является выбранной. Для переключения междурежимами рисования можно использовать клавиши-ускорители: <Alt>+<P> (перо) и <Alt>+<R> (линейка). См. также комментарии 1-3.

Комментарии

1. Номер текущего режима рисования содержится в поле mode и определяется c помощью функции indexOf вертикального менеджера компоновки.
2. При перемещении мыши в режиме **Ruler** необходимо постоянно перерисовывать текущую линию, стирая ее прежний вариант (соответствовавший предыдущему положению курсора мыши) и рисуя новый. Для того чтобы при стирании прежнего варианта восстанавливалось исходное изображение, для линии обычно применяется *инверсный цвет*, обладающий той особенностью, что повторное рисование линии на том же месте автоматически приводит к ее удалению и восстановлению исходного изображения. Для получения инверсного цвета требуется изменить свойство compositionMode, по умолчанию равное QPainter.CompositionMode\_SourceOver, на QPainter.RasterOp\_NotSourceXorDestination.
3. При задании особого значения для точки nullPt используется поле sys.maxsize. Подобное присваивание гарантирует, что поле nullPt не будет соответствовать никакой реальной точке на изображении.