# UNIDADE 4

## CREACIÓN DE COMPOÑENTES VISUAIS

**PARTE 1**

DESENVOLVEMENTO DE INTERFACES

CS DESENVOLVEMENTO DE APLICACIÓNS MULTIPLATAFORMA

Autor: Manuel Pacior Pérez

Índice

**1 INTRODUCIÓN....3**

**2 A APLICACIÓN PRINCIPAL QT....3**

2.1 EXEMPLO: DEFININDO A APLICACIÓN PRINCIPAL....4

2.1.1 Análise do código....4

**3 SIGNALS (SINAIS) E SLOTS (FENDAS)....5**

3.1 SIGNALS (SINAIS)....5

3.2 SLOTS (FENDAS)....6

3.3 EXEMPLO 1: BOTÓN SIMPLE....6

3.3.1 Análise do código....7

3.4 EXEMPLO 2: BOTÓN DEFININDO DOUS SLOTS....7

3.4.1 Análise do código....8

3.5 EXEMPLO 3: COMPOÑENTES GRÁFICOS CON SINAIS E SLOTS VINCULADOS....9

3.5.1 Análise do código....11

**4 EVENTOS....12**

4.1 EVENTOS DO RATO....13

4.2 MÉTODOS DE QMOUSEEVENT....13

4.3 EXEMPLO: IMPLEMENTAR EVENTOS DO RATO....14

4.3.1 Análise do código....15

# 1 INTRODUCIÓN

Nesta unidade imos ver como crear compoñentes personalizados partindo doutros compoñentes e modificando ou ampliando o seu comportamento.

Para poder realizar isto, primeiro veremos como se xestiona a comunicación entre os diferentes compoñentes do sistema, e que accións permiten os diferentes elementos Qt que nos proporciona a libraría Pyside2.

Comezaremos por definir que é a xestión de eventos:

- ✔ **Algo sucede**: facemos clic nun botón co rato, colocamos o punteiro do rato sobre un elemento, prememos unha tecla determinada, .... Isto representa un evento que emite unha sinal (signal) ou acción.
- ✔ **Algo pasa**: Cando se produce unha acción, como as descritas anteriormente, pódense desencadear unha ou máis reaccións.

No marco Qt, todo isto coñécese como *mecanismo de Signals (sinais) e Slots (fendas)*, que é o xeito de comunicación entre obxectos Qt.

Basicamente este é o seu funcionamento:

- Emítese un sinal, sinal, cando se produce un evento específico.
- Os widgets Qt veñen con moitos sinais predefinidos.
- Un slot é unha función ou método que se executará en resposta a un sinal concreto.
- Como ocorre cos sinais, os widgets Qt poden ter slots predefinidos.
- Normalmente, subclasificamos e reescribimos sinais e slots de widgets para que se axusten mellor ao que necesitamos que fagan.

# 2 A APLICACIÓN PRINCIPAL QT

Un gran número de funcionalidades Qt requiren a existencia dun obxecto especial que denominamos como a aplicación princial Qt. Entre os seus princpais obxectivos destaca a xestión dos bucles de eventos, manter un gran número de variables globais (como pode ser o idioma), etc.

Temos tres tipos de clases que permiten definir obxectos con estas características:

- `QCoreApplication`: Para tipos de aplicacións de consola.

- QGuiApplication: Para aplicacións con interfaces gráfica.
- QApplication: Do mesmo xeito que na clase anterior, úsase para definir aplicacións con interface gráfica, aínda que compre dicir que é máis habitual usar esta, pois estende a QGuiApplication.


## 2.1 EXEMPLO: DEFININDO A APLICACIÓN PRINCIPAL

Aínda que na Unidade 2 vimos como inicializar unha aplicación Qt, agora imos ver en detalle que fai cada un dos obxectos que se definen, para elo imos partir do seguinte exemplo:

```
if __name__ == "__main__":
    # Agora que temos definida a nosa xanela principal. Comezamos definindo a
    instancia
    app = QApplication(sys.argv)
    # da aplicación principal Qt e pasamos a lista de argumentos do sistema.
    # Definimos un obxecto da clase definida para xerar a xanela principal da
    aplicación
    window = MainWindow()
    # Amosamos a nosa xanela principal
    window.show()
    # Inicia o ciclo de eventos até que se pecha a aplicación
    sys.exit(app.exec_())
```

### 2.1.1 Análise do código

Sobre o exemplo do código anterior para inicializar unha aplicación debemos saber:

- A primeira liña fai unha comprobación que pode parecer estrana e que detallamos a continuación.

  ```
  if __name__ == "__main__":
  ```

  Isto está ligado á forma de traballar do intérprete de Python:

  - Cando o intérprete le un ficheiro de código, executa todo o código global nel. Isto implica a creación de obxectos para calquera función ou clase definida e variables globais.
  - Cada módulo (ficheiro de código) en Python, ten un atributo especial chamado __name__, que define o espazo de nomes no que se está executando. Úsase para identificar dun xeito único un módulo no sistema de importación. Poden darse dous contextos:

    - Pola súa banda "__main__" é o nome do ámbito no que se executa o código de nivel superior (o seu programa principal), é dicir cando en lugar de importarse como módulo, se executa como aplicación principal do seguinte xeito "python my_modulo.py".
    - Se o módulo non se chama como o programa principal, pero se importa doutro módulo, o atributo `__name__` contén o nome do propio ficheiro.

- Na variable `app` gardamos unha instancia `QApplication` que define a nosa aplicación Qt con interface gráfica.

```
app = QApplication(sys.argv)
```

- Temos unha fiestra principal que definimos e asignamos á variable window e acto seguido amosamos.

```
window = MainWindow()
window.show()
```

- Cando chamamos ao método `.exec_()` da instancia de `QApplication` que asignamos á variable `app`.

```
sys.exit(app.exec_())
```

  Isto fai que se inicie o bucle de eventos e a nosa aplicación comeza a "escoitar". Se non se produce ningún evento, segue esperando. Pero se ocorre un evento, reacciona.

    - Se o evento non chama ao método `exit` da instancia de `QApplication`, procesarao como corresponda e volverá a poñerse en "escoita" agardando por novos eventos.
    - Se o evento chama ao método `exit` da instancia `QApplication`, pechará a aplicación e devolverá a mensaxe co estado (normalmente 0 se todo foi correctamente).

# 3 SIGNALS (SINAIS) E SLOTS (FENDAS)

Cando se crean aplicacións gráficas temos que ter un xeito de conectar as accións cunha serie de eventos para provocar que se executen as accións definidas. Aquí é onde entran en xogo os seguinste conceptos: 'signals', 'slots'.

## 3.1 SIGNALS (SINAIS)

Os sinais son notificacións emitidas polos widgets cando ocorre algo. Iso algo pode ser unha cantidade de cousas, desde premer un botón ata o texto dunha caixa de entrada cambiando, ao texto da xanela que cambia. Moitos sinais son iniciados pola acción do usuario, pero esta non é

unha regra.

Ademais de avisar sobre algo que ocorre, tamén se poden enviar sinais datos para proporcionar un contexto adicional sobre o sucedido.

### 3.2 SLOTS (FENDAS)

Slots (fendas) é o nome que usa Qt para os receptores de signals (sinais). En Python calquera función (ou método) na súa aplicación, pódese usar coma un slot, simplemente conectándose o signal.

Se o signal envía datos, entón a función receptora recíbeos. Moitos widgets Qt tamén teñen os seus propios slots incorporados, é dicir, podes conectar directamente os Widgets Qt.

A continuación imos ver, a través de diferentes exemplos, os conceptos básicos dos signals (sinais) en Qt, e como se poden vincular aos widgets para que ocorran cousas nas aplicacións.

### 3.3 EXEMPLO 1: BOTÓN SIMPLE

No seguinte exemplo temos unha sinxela aplicación que ten un `QMainWindow` cun botón de tipo `QPushButton` configurado como o widget central. Imos comezar conectando este botón a un método Python personalizado.

Aquí creamos un sinxelo slot (fenda) personalizado chamado `o_boton_foi_premido` que acepta o signal (sinal) lanzado ao premer o `QPushButton`.

```
from PySide2.QtWidgets import QApplication, QMainWindow, QPushButton
from PySide2.QtCore import Qt
import sys

class MainWindow(QMainWindow):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("A miña aplicación")
        boton = QPushButton("Preme!")
        boton.setCheckable(True)
        boton.clicked.connect(self.o_boton_foi_premido)
        # Configuramos 'boton' como o Widget central de MainWindow
        self.setCentralWidget(boton)

    def o_boton_foi_premido(self):
        print("O botón foi premido!")

if __name__ == "__main__":
    # Agora que temos definida a nosa xanela principal. Comezamos definindoa
    instancia
    app = QApplication(sys.argv)
    # da aplicación principal Qt e pasamos a lista de argumentos do sistema.
    # Definimos un obxecto da clase definida para xerar a xanela principal da
```

```
aplicación
window = MainWindow()
# Amosamos a nosa xanela principal
window.show()
# Inicia o ciclo de eventos até que se pecha a aplicación
sys.exit(app.exec_())
```

### 3.3.1 Análise do código

Estamos recibindo datos, iso xa é un bo comezo. Observamos que os sinais tamén poden enviar datos para proporcionar máis información sobre o que acaba de acontecer.

```
boton.clicked.connect(self.o_boton_foi_premido)
```

O signal (sinal) `.clicked` non é unha excepción:, tamén proporciona un estado `checked` para o botón. Para os botóns por defecto do sistema, isto sempre é `False`, salvo que se modifique o seu comportamento a través da súa propiedade `checkable` que modificados a `True`, a través do método `setCheckable`.

```
boton.setCheckable(True)
```

A partires dese momento, cada vez que prememos o botón, este actualiza o valor deste atributo a (`True` ou `False`), máis neste primeiro exemplo non se está facendo nada con ese dato.

## 3.4 EXEMPLO 2: BOTÓN DEFININDO DOUS SLOTS

No seguinte exemplo temos unha sinxela aplicación moi semellante á anterior, ten un `QMainWindow` que cun botón de tipo `QPushButton` configurado como o widget central, máis neste caso engadimos un segundo slot que vinculamos ao sinal `.clicked`

```
import sys
from PySide2.QtCore import Qt
from PySide2.QtWidgets import QApplication, QMainWindow, QPushButton

class MainWindow(QMainWindow):

    def __init__(self, boton: QPushButton):
        super().__init__()
        self.setWindowTitle("A miña App")
        boton.setCheckable(True)
        boton.clicked.connect(self.o_boton_foi_premido_1)
        boton.clicked.connect(self.o_boton_foi_premido_2)
        # Configuramos 'boton' como o Widget central de MainWindow
        self.setCentralWidget(boton)

    def o_boton_foi_premido_1(self):
        print("Premido!")

    def o_boton_foi_premido_2(self, premido):
        print("Esta premido: ", premido)
```

```
if __name__ == "__main__":
    # Agora que temos definida a nosa xanela principal. Comezamos definindoa
    instancia
    app = QApplication(sys.argv)
    # da aplicación principal Qt e pasamos a lista de argumentos do sistema.
    # Definimos un obxecto da clase definida para xerar a xanela principal da
    aplicación
    boton = QPushButton("Preme!")
    window = MainWindow(boton)
    # Amosamos a nosa xanela principal
    window.show()
    # Inicia o ciclo de eventos até que se pecha a aplicación
    sys.exit(app.exec_())
```

### 3.4.1 Análise do código

A continuación destacamos as modificacións realizadas:

- Facemos unha nova variant, modificando o constructor `__init__` de `MainWindow` engadindo un parámetro `boton` de tipo `QPushButton`. A propiedade `checked` para de `boton` podemos consultala a través do método `isChecked()`. O valor desta propiedade, inxéctase no parámetro `premido` do slot `o_boton_foi_premido_2` (isto ocorre xusto despois de que se realice a acción de premer o botón).

```
...
def __init__(self, boton: QPushButton):
    super().__init__()
    self.setWindowTitle("A miña App")
...
```

- Neste segundo exemplo, ao igual que no anterior, fixemos que o noso botón sexa `checkable`

```
...
boton.setCheckable(True)
...
```

- Engadimos ao signal `.clicked` un primeiro slot co nome `o_boton_foi_premido_1`. Na documentación aparece para cada un dos compoeñentes os :signals, slots e event que ten cada un e as súas características. No caso do Widget `QPushButton`, o signal `.clicked` vén herdado a través da clase `QAbstractButton` á que estende (doc. na seguinte ligazón).

```
...
boton.clicked.connect(self.o_boton_foi_premido_1)
...
```

- Para ver o efecto que produce, engadimos ao signal `.clicked` un segundo slot (fenda) co nome `o_boton_foi_premido_2`, que imprime por consola o estado do botón, que a propia

sinal carga a través do parámetro `premido`, que ven definido polo valor da propiedade `checked`.

```
...
boton.clicked.connect(self.o_boton_foi_premido_2)
...
def o_boton_foi_premido_2(self, premido):
      print("Esta premido: ", premido)
```

Na fenda definida a través do método `o_boton_foi_premido_2` o parámetro `premido` pode chamarse de calquera xeito, decidín este porque pareceume que representaba ben a lóxica. O sinal `.clicked` é o encargo de inxectar a ese segundo parámetro, o valor que teña `checked` nese intre.

- Os slots ou fendas, son executados cando sucede o evento vinculado ao sinal, e fano na mesma orde na que foron asignados ao mesmo.

## 3.5 EXEMPLO 3: COMPOÑENTES GRÁFICOS CON SINAIS E SLOTS VINCULADOS

Neste tereceiro exemplo, realizamos unha pequena aplicación moi semellante ás anteriores, conta cunha xanela principal de tipo `QMainWindow` que contén un botón de tipo `QPushButton` configurado coma o widget central.

```
from PySide2.QtWidgets import QApplication, QMainWindow, QPushButton
from PySide2.QtCore import Qt
import sys
import time
from random import choice

# Definimos unha lista que contén todos os posibles
# valores que pode ter como título a xanela principal
titulos_da_xanela = [
      "A miña aplicación",
      "Aínda seguimos na aplicación",
      "Lóstregos!",
      "Isto é sorprendente",
      "Continuamos por acó",
      "Pode continuar",
      "Avante",
      "Algo fallou",
]

class MainWindow(QMainWindow):
      '''Esta Xanela amosa un botón como elemento central, que contén unha mensaxe
      como título da xanela. Existe unha lista de posibles títulos definida
      anteriormente titulos_da_xanela, e cada vez que o usuario preme o botón, o
      sistema escolle un mensaxe aleatorio, de todas as definidas nunha lista, e saca
      a mensaxe por consola. Ademais, cando esa mensaxe coincide co valor que ten a
      última da lista, entón o botón bloquéase.'''
```

```
    def __init__(self):
        ''' Constructor da aplicación'''
        # Chamamos ao constructor da clase pai
        super().__init__()
        # Inicializamos o título por defecto co primeiro elemento da lista
        self.n_times_clicked = 0
        # Inicializamos o título por defecto co primeiro elemento da lista
        self.setWindowTitle(titulos_da_xanela[0])
        self.boton = QPushButton("Preme!")
        self.boton.clicked.connect(self.o_boton_foi_premido)
        self.windowTitleChanged.connect(self.o_titulo_da_fiestra_foi_modificado)
        # Configuramos o Widget central da MainWindow
        self.setCentralWidget(self.boton)

    def o_boton_foi_premido(self):
        '''Método que define o Slot para o signal .clicked de boton'''
        # Aumentamos o contador do número de veces que se preme o botón en 1
        self.n_times_clicked += 1
        print("Premido!")
        # Escollemos ao azar entre os elementos da lista 'titulos_da_xanela'
        novo_titulo = choice(titulos_da_xanela)
        # Amosamos por consola o número de veces que se premeu o botón, e título
        da xanela
        print("O botón premeuse %s veces" % self.n_times_clicked)
        print("Vaise a modificar o título por: %s" % novo_titulo)
        self.setWindowTitle(novo_titulo)

    def o_titulo_da_fiestra_foi_modificado(self, titulo):
        '''Método que define o Slot para o signal .clicked de boton'''
        # Amosamos por consola o novo título que xa ten a xanela principal
        print("O título da xanela foi modificado a: %s" % titulo)
        # Non era preciso declarar esta variable, pero facilita a lectura do
        código. Nela gardamos
        # o último valor da lista de 'titulos_da_xanela', que será o valor por
        defecto de saída
        last_title = titulos_da_xanela[len(titulos_da_xanela) - 1]
        if titulo == last_title:
        self.boton.setDisabled(True)
        self.pechar_aplicacion()

    def pechar_aplicacion(self):
        ''' Método que serve para pechar a aplicación '''
        # Amosamos unha mensaxe por consola informando que se vai pechar a
        aplicación
        print("A aplicación precharase en 5 segundos")
        # Conxélase durante 5 segundos
        time.sleep(5)
        # Pechamos a aplicación
        sys.exit()

if __name__ == "__main__":
    # Agora que temos definida a nosa xanela principal. Comezamos definindoa
    instancia
    app = QApplication(sys.argv)
    # da aplicación principal Qt e pasamos a lista de argumentos do sistema.
    # Definimos un obxecto da clase definida para xerar a xanela principal da
```

```
aplicación
window = MainWindow()
# Amosamos a nosa xanela principal
window.show()
# Inicia o ciclo de eventos até que se pecha a aplicación
sys.exit(app.exec_())
```

### 3.5.1 Análise do código

Preséntase unha xanela na que se amosa un botón como elemento central, no que aparece unha mensaxe. Existe unha lista de posibles títulos definida chamada `titulos_da_xanela`, que contén unha serie de probables nomes para a xanela.

Cada vez que o usuario preme o botón, o sistema escolle un mensaxe aleatorio dos posibles que asigna como novo título da xanela, e ademais saca a información pola consola. Cando ese mensaxe coincide co valor que ten na derradeira posición da lista `titulos_da_xanela`, entón o botón bloquéas e a aplicación finaliza.

A continuación detallamos que as partes do código máis destacadas:

- Modificamos o constructor `__init__` de `MainWindow` engadindo o seguinte:
  - Inicializamos o contador para gardar as veces que se preme o botón.

    ```
    self.n_times_clicked = 0
    ```

  - Engadimos un primeiro slot co nome `o_boton_foi_premido` á sinal `.clicked` de `boton`. Cando o botón se prema, entón invócase ao método do slot.

    ```
    self.boton.clicked.connect(self.o_boton_foi_premido)
    ```

  - Engadimos á sinal `.windowTitleChanged` da propia xanela principal (`self`), un primeiro slot co nome `o_titulo_da_fiestra_foi_modificado`. Cando o título da fiestra cambie, entón invócase ao método do slot.

    ```
    self.windowTitleChanged.connect(self.o_titulo_da_fiestra_foi_modificad
    o)
    ```

- Imos detallar que fai o método `o_boton_foi_premido` definido para o slot. Este método está vinculado á sinal `.clicked`, e que polo tanto execútase cando se produce o evento de premer o botón.
  - Aumentamos en 1 o contador de veces que se premeu o botón

    ```
    # Aumentamos o contador do número de veces que se preme o botón en 1
    self.n_times_clicked += 1
    ```

  - Escollemos un novo título ao azar entre os elementos da lista 'titulos_da_xanela'

```
novo_titulo = choice(titulos_da_xanela
```

- Amosamos por consola o número de veces que se premeu o botón, e o novo título que asignaremos á xanela principal.

```
print("Premido!")
print("O botón premeuse %s veces" % self.n_times_clicked)
print("Vaise a modificar o título por: %s" % novo_titulo)
```

- Modificamos o nome do título da xanela polo novo. Debemos ter en conta que cando facemos isto, lánzase de xeito inmediato a sinal `.windowTitleChanged` que ten definido o slot `o_titulo_da_fiestra_foi_modificado`, polo que se executa ese método.

```
self.setWindowTitle(novo_titulo)
```

- Imos detallar que fai o método `o_titulo_da_fiestra_foi_modificado` definido para o slot vinculado á sinal `.windowTitleChanged` , e que polo tanto execútase cando se cambia o título da xanela principal.
  - Indicamos nunha mensaxe por consola que o título da xanela foi modificado e amosamos o novo nome.

```
print("O título da xanela foi modificado a: %s" % titulo)
```

  - Busco cal é o último elemento da lista `titulos_da_xanela`. É certo que non era preciso ter declarado esa variable `last_title`, pero fíxeno así para facilitar a lectura do código.

```
last_title = titulos_da_xanela[len(titulos_da_xanela)-1]
```

  - Se a mensaxe asignada á xanela principal coincide coa mensaxe situada na última posición da lista que calculamos anteriormente, entón deshabilitamos o botón e a aplicación finaliza.

```
if titulo == last_title:
      self.boton.setDisabled(True)
```

# 4 EVENTOS

Cada interacción que o usuario ten cunha aplicación Qt, é un evento. Hai moitos tipos de eventos e cada un deles debido a un tipo de interacción. Qt representa estes eventos usando obxectos de evento que conteñen información sobre o que pasou. Estes eventos pásanse aos controladores manipuladores de eventos específicos no widget onde se produciu a interacción.

Ao definir controladores de eventos personalizados, ou estender o funcionamento dos que xa existen, pode alterar o xeito no que os widgets responden a eses eventos.

Os controladores de eventos defínense como calquera outro método, pero o nome é específico para o tipo de evento que xestionan. A maioría dos tipos de eventos teñen clases especiais, por exemplo: `QResizeEvent`, `QPaintEvent`, `QMouseEvent`, `QKeyEvent` e `QCloseEvent`. Implementan á subclase `QEvent` e, cada unha delas, engade funcións específicas para eventos. Por exemplo, `QResizeEvent` engade `size()` e `oldSize()`, para permitir aos widgets calcular como se modificaron as súas dimensións.

## 4.1 EVENTOS DO RATO

A continuación imos describir un dos principais eventos que reciben os widgets, trátase de `QMouseEvent`, que son os creados para cada movemento e cada tecla que se realizan co rato sobre un widget. Os controladores que están dispoñibles para a xestión de eventos do rato son os seguintes.

| CONTROLADOR DE ENVENTOS (EVENT HANDLER) | TIPO DE EVENTO |
|---|---|
| mouseMoveEvent | O rato móvese |
| mousePressEvent | O botón do rato prémese |
| mouseReleaseEvent | O botón do rato deixa de premerse |
| mouseDoubleClickEvent | Detectouse un dobre clic |

Por exemplo, facer clic nun widget fará que se envíe un `QMouseEvent` ao controlador de eventos `.mousePressEvent` dese widget. Este controlador pode usar o obxecto do evento para descubrir información sobre o que pasou, como por exemplo, que provocou o evento e onde se produciu específicamente.

## 4.2 MÉTODOS DE QMOUSEEVENT

Todos os eventos do rato en Qt, producen un obxecto de tipo QMouseEvent que contén a información sobre o evento. Podemos acceder a ela a través dos seguintes métodos:

| MÉTODO | ACCIÓN |
|---|---|
| .button() | Devolve o botón específico que desencadea este evento |

| | |
|---|---|
| .buttons() | Contén o estado de todos os botóns do rato |
| .globalPos() | Devolve a posición global da aplicación como un `QPoint` |
| .globalX() | Devolve a posición global horizontal (x) da aplicación. |
| .globalY() | Devolve a posición global vertical (y) da aplicación. |
| .pos() | Devolve a posición relativa ao widget como un `QPoint` que representa as coordenadas como (X,Y) como valores enteiros (int) |
| .posF() | Devolve a posición relativa ao widget como un `QPoint` que representa as coordenadas como (X,Y) como valores en coma flotante (float) |

Podemos usar estes métodos dentro dun controlador de eventos, para responder de xeito particular ante diferentes eventos, ou ignoralos por completo.

- Os **métodos posicionais** proporcionan:
  - Información de posición global
  - Información de posición local (relativa ao widget)
  - Obxectos QPoint.
- Os **botóns infórmanse** empregando os tipos de botóns do rato desde o espazo de nomes `Qt`.

## 4.3 EXEMPLO: IMPLEMENTAR EVENTOS DO RATO

O seguinte exemplo consiste nunha aplicación que define unha xanela principal de tipo `QMainWindow` que implementa os diferentes eventos de contol do rato, e ademais leva un control do tipo de botón (esquerdo, central, dreito) que se preme no dispositivo.

```
import sys
from PySide2.QtCore import Qt
from PySide2.QtGui import QMouseEvent
from PySide2.QtWidgets import QApplication, QLabel, QLineEdit, QMainWindow

class MainWindow(QMainWindow):
      def __init__(self):
            super().__init__()
            self.setMinimumWidth(350)
            self.label = QLabel("Preme nesta xanela co rato")
            self.line_edit = QLineEdit()
            self.setMouseTracking(True)
            self.label.setMouseTracking(True)
            self.setCentralWidget(self.label)
```

```
    def mouseMoveEvent(self, e):
        '''Evento que salta cando, mentres temos o botón premido, movemos o rato
        sobre o Label'''
        text_label = "O rato móvese x: {0} | y: {1}".format(e.pos().x(),
        e.pos().y())
        self.label.setText(text_label)

    def mousePressEvent(self, e):
        '''Evento que salta cando se fai un click co rato sobre o Label'''
        self.label.setText("Evento: mousePressEvent, Botón: " +
        self.devolver_nome_do_boton_premido(e))

    def mouseReleaseEvent(self, e):
        '''Evento que salta cando despois de premer o botón do rato, deixamos de
        premelo'''
        self.label.setText("Evento: mouseReleaseEvent, Botón: " +
        self.devolver_nome_do_boton_premido(e))

    def mouseDoubleClickEvent(self, e):
        '''Evento que salta cando se fai un duplo click co rato sobre o Label'''
        self.label.setText("Evento: mouseDoubleClickEvent, Botón: " +
        self.devolver_nome_do_boton_premido(e))

    def devolver_nome_do_boton_premido(self, e: QMouseEvent):
        ''' Método que devolve o nome do botón do rato premido '''
        if e.button() == Qt.LeftButton:
            # O evento ten información correspondente ao botón esquerdo
            return "Esquerdo"
        elif e.button() == Qt.MiddleButton:
            # O evento ten información correspondente ao botón central
            return "Central"
        elif e.button() == Qt.RightButton:
            # O evento ten información correspondente ao botón direito
            return "Direito"

if __name__ == "__main__":
    # Agora que temos definida a nosa xanela principal. Comezamos definindoa
    instancia
    app = QApplication(sys.argv)
    # da aplicación principal Qt e pasamos a lista de argumentos do sistema.
    # Definimos un obxecto da clase definida para xerar a xanela principal da
    aplicación
    window = MainWindow()
    # Amosamos a nosa xanela principal
    window.show()
    # Inicia o ciclo de eventos até que se pecha a aplicación
    sys.exit(app.exec_())
```

### 4.3.1 Análise do código

Este código non ten nada moi diferente ao que vimos no resto de exemplos, se ben imos destacar os aspectos destacables:

- O método `setMouseTracking` dos compoñentes Qt, serve para activar/desactivar que os eventos do rato funcionen sen ter que presionar o botón. Neste caso o valor po defecto é `FALSE` máis no noso caso queremos que tan pronto entre o rato na nosa aplicación, xa comece a recibir os eventos do rato sen que se precise que teña un botón premido.

```
self.setMouseTracking(True)
self.label.setMouseTracking(True)
```

- O método `devolver_nome_do_boton_premido` recibe como parámetro un parámetro de tipo `QMouseEvent`. Desde a implementación dos eventos invocamos a función para que nos devolva o tipo de botón que se premeu. Para comprobar o tipo usamos o espazo de nomes `Qt`

```
def devolver_nome_do_boton_premido(self, e: QMouseEvent):
    ''' Método que devolve o nome do botón do rato premido '''
    if e.button() == Qt.LeftButton:
        # O evento ten información correspondente ao botón esquerdo
        return "Esquerdo"
    elif e.button() == Qt.MiddleButton:
        # O evento ten información correspondente ao botón central
        return "Central"
    elif e.button() == Qt.RightButton:
        # O evento ten información correspondente ao botón direito
        return "Direito"
```