Programowanie
po 9 wykładzie

Andrzej Giniewicz

10.05.2024

W tej porcji materiału zajmujemy się interfejsami graficznymi w bibliotece PyQt.

1 Qt, PyQt i PySide — co do czego i odrobina historii

Biblioteka i zbiór narzędzi Qt służy do tworzenia interfejsów graficznych. Jest jedną z bibliotek dających taką możliwość, działającą na wielu platformach, nie tylko na systemach Windows, macOS oraz Linux, ale również na telefonach. Do innych wieloplatformowych bibliotek do tworzenia interfejsów graficznych należy między innymi GTK oraz wxWidgets. Podczas zajęć jako biblioteka do GUI wybrana została Qt, ze względu na dobre wsparcie dla Pythona na wszystkich systemach operacyjnych.

Pomysł na bibliotekę Qt powstał w 1990 roku (Haavard Nord i Eirik Chambe-Eng siedzący na ławce w parku w Trondheim). Od 1992 roku zaczęli implementować bibliotekę a w 1994 roku założyli firmę Trolltech, której celem była sprzedaż komercyjnych licencji na bibliotekę Qt. W 1995 roku wydali pierwszą wersję beta, natomiast w 1996 roku pierwszą wersję stabilną. W tym samym roku mieli pierwszego klienta biznesowego — Europejską Agencję Kosmiczną. W 2008 roku firmę Trolltech kupiła Nokia, która rozwijała Qt do 2012 roku, kiedy prawa do biblioteki Qt wykupiła firma Digia. Od 2014 roku rozwojem biblioteki zajmuje się jej część — „The Qt Company". Od pierwszych wersji biblioteki system dystrybucji był dwulicencyjny — darmowa wersja oraz płatna wersja. Historyczna notka jest o tyle istotna, że podczas zmian właściciela praw do Qt, zmieniały się również licencje. Biblioteka do wersji Qt 4.5 rozwijana głównie przez Trolltech, miała licencję komercyjną i GPL, natomiast od wersji Qt 4.5 (za czasów rozwoju w firmie Nokia), licencja to licencja komercyjna i LGPL na większość modułów oraz GPL na niektóre[1].

Licencja GPL jest dość skomplikowanym dokumentem prawnym[2], który komplikował użytkowanie biblioteki Qt w wielu projektach. Licencja ta definiuje cztery podstawowe wolności:

1. wolność do uruchamiania programu w dowolnym celu, czyli na przykład można tworzyć dzieła komercyjne za pomocą darmowego programu na licencji GPL,

[1] Lista modułów na licencji GPL w najnowszej wersji Qt, patrz `https://doc.qt.io/qt-6/qtmodules.html#gpl-licensed-addons`.

[2] Pełny tekst dostępny na stronie `https://www.gnu.org/licenses/gpl-3.0-standalone.html`.

2. wolność analizowania i modyfikowania kodu programu, czyli musimy mieć dostęp do kodu, choć licencja nie deklaruje, w jaki sposób — możemy go otrzymać po listownej prośbie do twórców oprogramowania

3. wolność do rozpowszechniania niezmodyfikowanego oprogramowania, czyli nikt nie może nam zabronić przesyłania go dalej,

4. wolność do udoskonalania i rozpowszechniania modyfikacji programu, ale przy założeniu, że modyfikacje będą na tej samej licencji.

Ostatni punkt licencji powoduje, że kod na licencji GPL nie może być użyty w programie na innej licencji. Z tego powodu mówimy o licencji GPL, że jest „licencją wirusową".

Licencja LGPL powstała po to, aby osłabić wirusowość licencji GPL w sytuacji bibliotek. Licencja LGPL dodaje wyjątek mówiący, że

- możemy **korzystać** z niezmodyfikowanych bibliotek w aplikacjach o dowolnej licencji,

- jakiekolwiek **modyfikacje** biblioteki, muszą być udostępnione na licencji LGPL.

Oznacza to, że o ile nie modyfikujemy samej biblioteki, tylko jesteśmy jej użytkownikami, czyli używamy jej w naszym programie, nie mamy narzuconej żadnej licencji. Jest to sytuacja o wiele bardziej wygodna dla biblioteki takiej jak Qt, ponieważ programiści zwykle wykorzystują ją niezmienioną do tworzenia aplikacji z interfejsem graficznym, a nie modyfikują jej samej. Z tego powodu biblioteka Qt od wersji 4.5 może być szerzej używana, bez ryzyka naruszenia warunków licencji i bez konieczności zakupienia licencji komercyjnej, która od tego czasu różni się od licencji darmowej głównie tym, że zawiera dodatkowe wsparcie techniczne.

Biblioteka Qt jest napisana w języku C++ i działa dla języka C++. Aby wykorzystać ją w innych językach, potrzebne są odpowiednie łączenia (ang. *bindings*). Do łączenia Qt z Pythonem służy biblioteka PyQt lub PySide. Biblioteka PyQt jest rozwijana przez firmę Riverbank Computing od 1998 roku i posiada (do dziś) identyczny model licencji jak ten używany przez firmę Trolltech, czyli licencję komercyjną plus licencję GPL. PySide powstała w firmie Nokia jako alternatywne połączenie z Qt dla Pythona, oparte na licencji LGPL. Obecnie biblioteka PySide jest rozwijana przez firmę „The Qt Company" i jest zalecanym sposobem łączenia bibliotek. Składnia bibliotek PyQt oraz PySide jest bardzo zbliżona, różnice są na tyle kosmetyczne, że powstała biblioteka QtPy[3], która pozwala napisać identyczny kod i wykorzystać taką bibliotekę, jaka jest dostępna na komputerze użytkownika — a jeśli jest ich kilka, wybrać jedną z nich za pomocą zmiennej środowiskowej.

Ważnym dodatkowym aspektem jest to, że różne wersje biblioteki Qt nie są kompatybilne. Oznacza to, że kod napisany dla Qt4, nie musi działać dla Qt5 lub najnowszej wersji — Qt6. Z tego powodu, różne biblioteki korzystające z Qt muszą „nadgonić". I tak na przykład ważna dla nas biblioteka matplotlib ma wsparcie dla Qt w wersji 6 dopiero od wydania 3.6[4]. Również dystrybucja Anaconda, zalecana na zajęciach, nie ma jeszcze

[3] Patrz `https://github.com/spyder-ide/qtpy`.

[4] Patrz `https://github.com/matplotlib/matplotlib/pull/19255`.

preinstalowanej wersji PyQt 6 lub PySide 6. Z tego powodu na niniejsze zajęcia zalecana jest najnowsza dostępna wersja Qt 5. Jeśli chodzi o rozgraniczenie pomiędzy PyQt lub PySide, nie ma to istotnej różnicy, którą bibliotekę wykorzystamy, ponieważ nasze programy nie będą publikowane, tak więc różnice licencji nie są istotne w środowisku edukacyjnym. W dystrybucji Anaconda domyślnie dostępne jest PyQt[5], zatem w przykładach posłużymy się biblioteką **PyQt 5**.

Różnice i nazwy bibliotek do łączenia z Pythonem, są dostępne w tabeli poniżej

wersja Qt	licencja Qt	wersja PyQt**	wersja PySide**
<4.5	komercyjna lub GPL	PyQt 4	—
⩾4.5	komercyjna lub LGPL*	PyQt 4	PySide
5	komercyjna lub LGPL*	PyQt 5	PySide 2
6	komercyjna lub LGPL*	PyQt 6	PySide 6

* — oprócz niektórych modułów, które są na licencji GPL.
** — wszystkie wersje PyQt są na licencji GPL lub komercyjnej, wszystkie wersje PySide są na licencji LGPL.

Podczas zajęć można wykorzystać inny wariant biblioteki, jednak należy wtedy we własnym zakresie zadbać o jej wybór, instalację oraz modyfikacje kodu z przykładów.

2 Aplikacje graficzne w Pythonie

Aplikacje w Pythonie zamykamy zwykle w skrypt. Ogólnym schematem skryptu[6] dla aplikacji jest

```
#!/usr/bin/env python

def main():
    ...

if __name__ == "__main__":
    main()
```

W kodzie tym najpierw informujemy BASH'a w jakim języku napisany jest skrypt, następnie mamy funkcję, która docelowo będzie zawierała kod wykonywalny aplikacji. Warunek `if` zapisany w sposób powyższy uruchamia tę funkcję, gdy plik uruchomimy jako skrypt, natomiast nie uruchomi funkcji, gdy plik uruchomimy jako bibliotekę, na przykład importując z niego funkcję `main`.

Aplikacja napisana w PyQt trzyma się tego schematu, przy czym minimalna jej wersja wygląda następująco (nie uruchamiajmy jej jeszcze, nic w niej nie ma, więc po prostu będzie „wisieć")

```
#!/usr/bin/env python
from PyQt5.QtWidgets import QApplication
```

[5] PySide można łatwo doinstalować, patrz `https://anaconda.org/conda-forge/pyside2`.
[6] Materiał można sobie przypomnieć z notatek po 14 wykładzie z poprzedniego semestru.

```
def main():
    app = QApplication([])

    # tutaj zbudujemy aplikację

    app.exec()

if __name__ == "__main__":
    main()
```

Aby było cokolwiek widać, rozbudujemy wersję o okienko. Załóżmy, że zbudujemy grę w kółko i krzyżyk.

```
#!/usr/bin/env python
from PyQt5.QtWidgets import QApplication, QWidget

def main():
    app = QApplication([])
    okno = QWidget()
    okno.resize(400, 600)
    okno.move(50, 50)
    okno.setWindowTitle("Kółko i krzyżyk")

    # tutaj rozbudujemy aplikację

    okno.show()
    app.exec()

if __name__ == "__main__":
    main()
```

Aplikacje w Qt automatycznie obsługują też wiele argumentów[7], aby móc je obsłużyć z poziomu skryptu PyQt, wystarczy je przekazać w konstruktorze aplikacji.

```
#!/usr/bin/env python
import sys

from PyQt5.QtWidgets import QApplication, QWidget

def main():
    app = QApplication(sys.argv)
    okno = QWidget()
    okno.resize(400, 600)
```

[7] Patrz https://doc.qt.io/qt-5/qapplication.html#QApplication.

```
    okno.move(50, 50)
    okno.setWindowTitle("Kółko i krzyżyk")

    # tutaj rozbudujemy aplikację

    okno.show()
    app.exec()

if __name__ == "__main__":
    main()
```

Taką aplikację możemy już swobodnie uruchomić, choć na razie jedyne co możemy z nią zrobić, to przeskalować okienko, zminimalizować je, przywrócić lub zamknąć.

Wszystko co widzimy na ekranie w PyQt jest „Widgetem". `QWidget` jest klasą bazową dla wszystkich innych elementów interfejsu[8], takich jak okna, przyciski, listy rozwijane, napisy, obrazki, tabelki i temu podobne. Aby nie tworzyć jednej długiej funkcji `main`, tylko rozbić ją na więcej elementów, skorzystamy z dziedziczenia.

```
#!/usr/bin/env python
import sys

from PyQt5.QtWidgets import QApplication, QWidget

class Gra(QWidget):
    def __init__(self):
        # najpierw uruchamiamy konstruktor klasy QWidget
        super().__init__()

        # ustawiamy okno
        self.resize(400, 600)
        self.move(50, 50)
        self.setWindowTitle("Kółko i krzyżyk")

        # tutaj rozbudujemy aplikację

def main():
    app = QApplication(sys.argv)
    gra = Gra()
    gra.show()
    app.exec()

if __name__ == "__main__":
    main()
```

[8] Patrz https://www.riverbankcomputing.com/static/Docs/PyQt5/api/qtwidgets/qwidget.html.

Od teraz w dalszych fragmentów kodu, będziemy pomijać funkcję `main` oraz jej uruchamianie. Będziemy skupiać się na metodach klasy `Gra`, niekiedy dodając jakiś `import` na początku pliku.

Dodamy teraz interfejs. Kółko i krzyżyk wymaga 9 pól gry. Zaimplementujemy je za pomocą dziewięciu przycisków, trzy rzędy po trzy przyciski. Jako przycisk wykorzystamy obiekty klasy `QPushButton`[9]. Stworzymy metodę `zbuduj_planszę`, której zadaniem będzie przygotowanie odpowiednich przycisków i ułożenie ich w siatce[10] 3×3.

```
from PyQt5.QtWidgets import QPushButton, QLabel, QGridLayout
from PyQt5.QtGui import QFont
from PyQt5.QtCore import Qt

# ...

    def __init__(self):
        # ...
        self.zbuduj_planszę()

    def zbuduj_planszę(self):
        # budujemy układ w siatkę
        layout = QGridLayout(self)
        # tworzymy i zapamiętujemy przyciski, aby móc się do nich odwołać
        self.przyciski = []
        for y in range(3):
            rząd = []
            for x in range(3):
                przycisk = QPushButton()
                layout.addWidget(przycisk, y, x)
                # wyliczamy pozycję i ustalamy rozmiary
                przycisk.setFixedSize(110, 110)
                # font to Arial, 30pt
                przycisk.setFont(QFont("Arial", 30))
                rząd.append(przycisk)
            self.przyciski.append(rząd)
        # miejsce na napis, w którym umieścimy wyniki
        self.wyniki = QLabel()
        layout.addWidget(self.wyniki, 4, 0, 1, 3)
        # tekst wyśrodkowany, Arial 18pt
        self.wyniki.setAlignment(Qt.AlignCenter)
        self.wyniki.setFont(QFont("Arial", 18))
        self.wyniki.setText("Ruch gracza X")
```

[9] Patrz https://www.riverbankcomputing.com/static/Docs/PyQt5/api/qtwidgets/qpushbutton.html oraz https://doc.qt.io/qt-5/qpushbutton.html.

[10] Patrz https://www.riverbankcomputing.com/static/Docs/PyQt5/api/qtwidgets/qgridlayout.html.

Zwróćmy uwagę, że tworząc układ, podajemy `self` jako argument, dzięki temu „widget" ten będzie częścią definiowanego właśnie okienka. Pomiędzy elementami graficznymi jest hierarchia rodzic-dziecko, w tym przypadku układ jest dzieckiem okienka a każdy z przycisków i zdefiniowane również miejsce na wyniki są pod układem. Układ siatkowy automatycznie rozmieszcza elementy, więc nie musimy wyznaczać ich położenia.

Zanim uruchomisz kod, spróbuj naszkicować na kartce, jak może wyglądać zbudowany przez nas interfejs. Następnie uruchom kod i zweryfikuj, czy uzyskałeś identyczny wynik.

Niestety obecnie możemy klikać, ale nic się nie dzieje. Aby coś się działo, musimy podłączyć akcję pod przyciski. W tym celu obsłużymy sygnał `clicked`. Sygnały to różne zdarzenia, które mogą mieć w interfejsie. Sygnał `clicked` występuje, gdy klikniemy obiekt graficzny. Sygnały łączymy ze slotami. Sloty to funkcje lub metody, które powinny zostać wywołane, gdy zaistnieje sygnał. W kodzie poniżej slotem jest metoda `akcja`.

```
    def zbuduj_planszę(self):
        # ...
                przycisk.setFont(QFont("Arial", 30))

                # tu dodajemy obsługę sygnału
                przycisk.clicked.connect(self.akcja)

                rząd.append(przycisk)
        # ...

    def akcja(self):
        ...  # tu dodamy obsługę klikania
```

Zajmijmy się teraz obsługą przycisku. Do konstruktora dodamy numer ruchu, aby wiedzieć, kto teraz oddaje ruch. Jeśli do tej pory była parzysta liczba ruchów, w tym zero, następny gracz to krzyżyk. Jeśli liczba ruchów była nieparzysta, gracz to kółko. Po kliknięciu wyłączymy przycisk, aby nie można było go nacisnąć drugi raz.

```
    def __init__(self):
        # ...

        # liczba ruchów
        self.ruchy = 0

    # ...

    def akcja(self):
        # w co klikamy?
        przycisk = self.sender()

        # wyłączamy klikanie
```

```
        przycisk.setEnabled(False)

        if self.ruchy % 2 == 0:
            przycisk.setText("X")
            self.wyniki.setText("Ruch gracza O")
        else:
            przycisk.setText("O")
            self.wyniki.setText("Ruch gracza X")

        self.ruchy += 1
```

Teraz już coś mamy. Zwróćmy uwagę, jak dzięki metodzie sender, możemy dostać się do obiektu, który wygenerował sygnał. W obecnej wersji brakuje nam jeszcze sprawdzania, kto wygrał. Dodamy to na końcu metody z obsługą zdarzeń, ponieważ po każdym kliknięciu chcemy sprawdzać zwycięzcę. Aby sprawdzić zwycięzcę, musimy dostać się do wpisanych do przycisków wartości i sprawdzić, czy są trzy identyczne w rzędzie, kolumnie lub na przekątnej. Po wygranej wszystkie przyciski dezaktywujemy, aby nie dało się dalej „grać".

```
    def akcja(self):
        # ...

        # czy koniec gry?
        koniec = False
        # sprawdzamy po trzy
        for n in range(3):
            # wygrana w wierszu
            if self.przyciski[n][0].text() == self.przyciski[n][1].text() == \
                    self.przyciski[n][2].text() != '':
                self.wyniki.setText(f"Wygrywa gracz {self.przyciski[n][0].text()}")
                koniec = True
                break
            # wygrana w kolumnie
            if self.przyciski[0][n].text() == self.przyciski[1][n].text() == \
                    self.przyciski[2][n].text() != '':
                self.wyniki.setText(f"Wygrywa gracz {self.przyciski[0][n].text()}")
                koniec = True
                break
        # wygrana po przekątnej "\"
        if self.przyciski[0][0].text() == self.przyciski[1][1].text() == \
                self.przyciski[2][2].text() != '':
            self.wyniki.setText(f"Wygrywa gracz {self.przyciski[0][0].text()}")
            koniec = True
        # wygrana po przekątnej "/"
        elif self.przyciski[0][2].text() == self.przyciski[1][1].text() == \
```

```
            self.przyciski[2][0].text() != '':
        self.wyniki.setText(f"Wygrywa gracz {self.przyciski[0][2].text()}")
        koniec = True
    # jeśli nikt nie wygrał, ale 9 ruchów wykorzystano, to remis
    if not koniec and self.ruchy == 9:
        self.wyniki.setText("Remis")
    # jeśli koniec gry, dezaktywujemy wszystkie przyciski
    elif koniec:
        for x in range(3):
            for y in range(3):
                self.przyciski[y][x].setEnabled(False)
```

Gra, którą mamy, jest dość minimalistyczna. Zachęcam do tego, aby ją zmodyfikować. Przykładowo można rozważyć dodanie przycisku, który pozwoli na zagranie ponownie po końcu gry. Można też pomyśleć nad wersją 5×5 lub innym wariantem.

3 Podsumowanie

Za pomocą omówionego prostego przykładu gry w kółko i krzyżyk przedstawione zostały najważniejsze elementy biblioteki PyQt. Do najważniejszych pojęć należą:

- elementy interfejsu (Widget), które tworzą drzewa rodzic-dziecko,
- sloty i sygnały (Slot, Signals), które pozwalają łączyć zdarzenia z kodem je obsługującymi,
- układy (Layout), które pozwalają na automatyczne rozmieszczanie elementów w okienku.

To, co pozostaje, by stworzyć dowolnie skomplikowany interfejs, to użycie dokumentacji, aby móc odnaleźć właściwy element interfejsu lub układ, który chcemy umieścić.

Interfejsy graficzne można zaprogramować lub zaprojektować w aplikacji Qt Designer. O współpracy Qt Designera z Pythonem można przeczytać w sieci, na przykład w `https://realpython.com/qt-designer-python/`. Lista książek, które pomogą zgłębić temat bardziej, dostępna jest na stronie `https://wiki.python.org/moin/PyQt/Books`. Gotowy kod z przykładów z niniejszych notatek dostępny jest pod adresem `http://prac.im.pwr.edu.pl/~giniew/tictactoe.py`.