

Lista 7 – trochę więcej Qt

Pod adresem <https://github.com/zkoza/cpp-issp/tree/main/cpp/qt-lilac-chaser> znajduje się mój program napisany w Qt w postaci, jaką udało mi się osiągnąć na wykładzie (w 2021 r.).

Państwa zadaniem będzie jego dalsze udoskonalenie. W tym celu proszę najpierw zapoznać się z początkiem filmu na youtube <https://www.youtube.com/watch?v=78T848QuaME>

(15 Mind Blowing Optical illusions and Strange Visual Phenomena),

skąd zaczerpnąłem inspirację. Państwa zadaniem jest upodobnienie animacji do tego, co widać na filmie (minuty 0:14 do 1:17, „Lilac Chaser”).



1. Proste modyfikacje:

- zmień liczbę dysków na 12,
- zmniejsz ich promień,
- dodaj jasnoszare tło (por. https://wiki.qt.io/Colors_and_Font_Guidelines i/lub <https://www.w3.org/TR/SVG11/types.html#ColorKeywords>),
- narysuj w środku koła czarny krzyżyk (na tym krzyżyku widz animacji powinien skupić wzrok),
- zmień kolor dysków na podobny do tego z filmu.

2. W kodzie obsługi polecenia menu „Koniec (ctrl-Q)”, czyli w funkcji `void MainWindow:::koniec_programu()`, dodaj wywołanie okienka z zapytaniem o to, czy użytkownik naprawdę chce zakończyć program i dwoma przyciskami: Yes/No (lub Tak/Nie, ale to nieco trudniejsze). Tylko potwierdzenie zamiaru przyciskiem „Yes” powinno kończyć program. Proponuję zastosować `QMessageBox:::question`, ale do tego, jak tego użyć, trzeba dojść samodzielnie (dokumentacja, źródła w internecie etc.).

Wskazówka: Qt został zaprojektowany tak, by tego rodzaju zadania dało się wykonać niewielkim nakładem pracy.

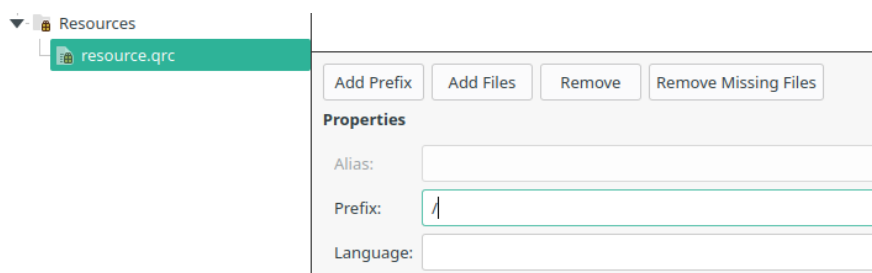
3. Zamiast pełnych, zwyczajnych kółek wyświetl rozmazane chmurki, jak na filmie.

Wskazówka 1: Proponuję zapoznać się z klasą `QRadialGradient`. Jej dokumentacja jest skąpa, ale Internet pełen jest samouczków, np.: https://www.bogotobogo.com/Qt/Qt5/QLinear_QRadial_QConical_QGradient.php. Zawarte tam przykłady powinny pomóc zrozumieć, jakie jest znaczenie parametrów konstruktora tego gradientu i jak się go używa w praktyce.

Wskazówka 2: gradient można przekazać jako argument konstruktora pędzla (`QBrush`). Alternatywne rozwiązanie: zamiast `painter.drawEllipse` można do rysowania kółek użyć, jak w zalinkowanym powyżej samouczku, funkcji `fillRect`. Jeśli kółka na siebie nie nachodzą, to efekt końcowy będzie identyczny.

4. Dodaj kod, dzięki któremu po kliknięciu przycisku włączającego lub zatrzymującego animację, stosownie do sytuacji wyświetlała się na nim będzie ikonka ► lub ||. W tym celu:

- Utwórz w katalogu źródłowym katalog `img` (lub o innej, czytelnej dla Ciebie nazwie) i umieść w nim załączone pliki z ikonami (*.png)
- Dodaj do projektu (*.pro) plik z zasobami
 - W QtCreator: Ctrl-N. Następnie w oknie dialogowym New File wybierz w lewym panelu: Qt, w środkowym: Qt Resource file
 - lub bezpośrednio w pliku *.pro dodaj wiersz `RESOURCES += resource.qrc`
- Otwórz Resource editor (może się przydać prawy klawisz myszki), kliknij „Add prefix”, wybierz prefix, np. „/”:



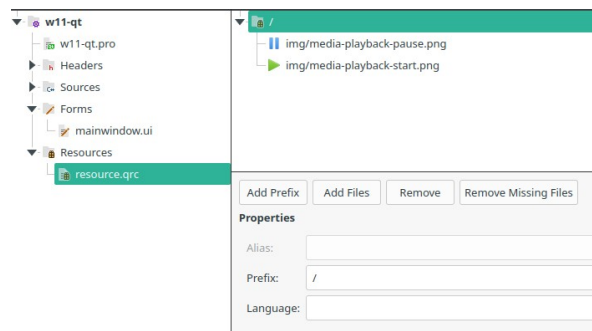
```
SOURCES += \
    main.cpp \
    mainwindow.cpp \
    my_widget.cpp

HEADERS += \
    mainwindow.h \
    my_widget.h

FORMS += \
    mainwindow.ui

RESOURCES += \
    resource.qrc
```

- Kliknij **Add Files** i dodaj pliki z ikonami ze swojego podkatalogu **img**



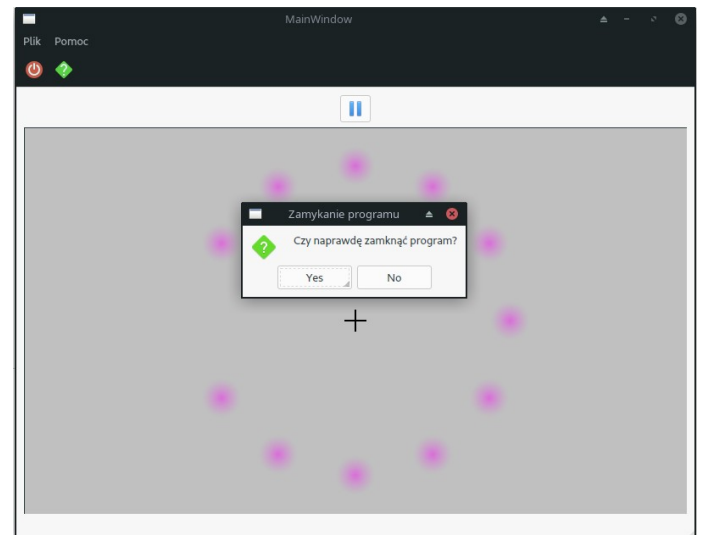
- Dodaj w programie kod, który odpowiednio do sytuacji zmienia ikonę wyświetlaną na przycisku. Uwaga. Aby ona się wyświetliła, przycisk nie może zawierać tekstu. Ścieżka do plików pobieranych z zasobów (zdefiniowanych w `*.qrc`) powinna się rozpoczynać od dwukropka, np.

```
ui->przycisk->setIcon(QIcon(":/img/media-playback-start.png"));
```

Przy okazji sprawdź, jak teraz wygląda zawartość pliku z definicją zasobów (`*.qrc`).

5. Na pasku narzędzi widać „nudne” przyciski z tekstem zamiast grafiki (Koniec, O programie). Zrób coś, żeby tam pojawiły się ikonki.

Rysunek obok przedstawia zrzut ekranu z mojego programu po wybraniu z menu opcji „Koniec”.



Uwagi

Powyższe zadania są łatwe dla osób, które miały choć przelotną styczność z Qt, a zapewne trudne dla pozostałych. Generalnie, programowanie polega na posługiwaniu się cudzym kodem. Zdarza się, że biblioteki w ogóle nie mają dokumentacji (wtedy za dokumentację służy kod źródłowy i jakiś proste `Read.me`). Częściej za całą dokumentację służy kilka przykładowych programów. Tutaj muszą państwo dać sobie radę z programem zapisanym w kilku plikach, napisanym przez kogoś innego, posługującym się nieznaną państwu technologią, której trzeba się uczyć wraz z pisanie własnego kodu, i której trzeba użyć w nieznanym sobie środowisku (np. QtCreator). Pytanie, co z powyższego było na zajęciach, a czego nie – jest niestosowne. Programowanie właśnie tak wygląda, tyle że zwykle dokumentacja jest znacznie, znacznie uboższa.