JOURNAL of COMPUTER SCIENCES INSTITUTE

JCSI 6 (2018) 55-58
Wysłane: 2017-10-31
Przyjęte: 2017-11-02

# Porównanie możliwości bibliotek GTK+ oraz Qt w programowaniu gier

Kamil Drzas*, Dominik Alchimowicz*, Maciej Pańczyk
Politechnika Lubelska, Instytut Informatyki, Nadbystrzycka 36B, 20-618 Lublin, Polska

**Streszczenie**. Programowanie gier składa się z wielu elementów. Implementacja podstawowych funkcji takich jak renderowanie obrazu czy obsługa dźwięku wymaga użycia niskopoziomowych bibliotek i dużej wiedzy. Środowiska do tworzenia interfejsów graficznych aplikacji, ze względu na ich funkcjonalność, mogą być atrakcyjnym wyborem w przypadku tworzenia prostych gier. Artykuł przedstawia porównanie dwóch popularnych bibliotek: GTK+ oraz Qt.

**Słowa kluczowe:** programowanie gier; gtk+; qt

*Autor do korespondencji.

Adresy e-mail: drzas.kamil@gmail.com, dominik.alchimowicz@gmail.com

# Comparison of GTK+ and Qt libraries in game development

Kamil Drzas*, Dominik Alchimowicz*, Maciej Pańczyk
Institute of Computer Science, Lublin University of Technology, Nadbystrzycka 36B, 20-618 Lublin, Poland

**Abstract.** Game development consists of many elements. Implementation of basic features such as rendering and sound handling requires usage of low level libraries and deep knowledge. Libraries that help to create user interfaces (UI), because of their functionality, may be an attractive platform for creating basic games. This paper presents the results of comparison of the two popular libraries: GTK+ and Qt.

**Keywords:** game development; gtk+; qt

*Corresponding author.

E-mail addresses: drzas.kamil@gmail.com, dominik.alchimowicz@gmail.com

## 1. Wstęp

Programowanie gier komputerowych to bardzo skomplikowany proces. Nawet prosta gra wymaga implementacji wielu elementów, takich jak renderowanie grafiki, obsługa fizyki (wykrywanie kolizji, czy działanie grawitacji), sterowania, czy też dźwięków. W przypadku mniej skomplikowanych gier, lub prototypów, implementowanie tych funkcji staje się zbyt czasochłonne i nieopłacalne.

Silnik gry to ekosystem zawierający wyżej wymienione moduły. Duże firmy inwestują bardzo duże środki we własne, zamknięte technologie, jednak na rynku istnieją też dostępne dla niezależnych deweloperów środowiska takie jak Unity czy Unreal Engine. Silniki takie zwykle zawierają wszystkie potrzebne elementy do stworzenia kompletnego produktu.

Korzystanie z gotowych silników niesie za sobą pewne konsekwencje. Poziom ich skomplikowania może utrudnić pracę dla początkujących programistów, a mała elastyczność ogranicza swobodę w tworzeniu gry i wymaga dostosowania się do danego środowiska.

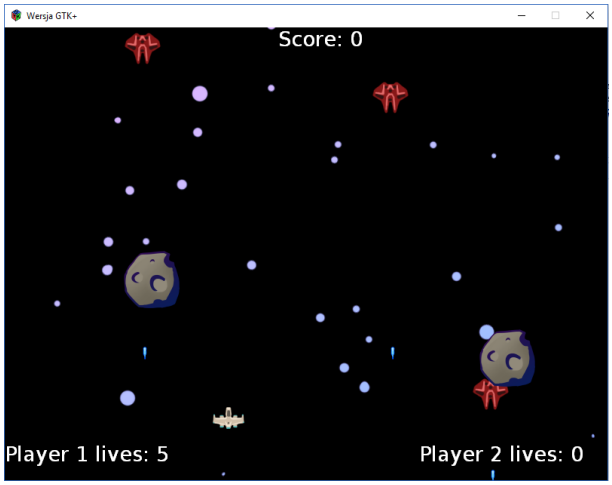
Napisanie gry od ‘zera’ nie musi wymagać samodzielniej implementacji wszystkich potrzebnych modułów. Na rynku znajduje się wiele darmowych bibliotek udostępniających konkretne funkcjonalności np. Box2D (silnik fizyki 2D) czy libgdx (audio, obsługa urządzeń wejściowych, grafika).

Celem niniejszej pracy było porównanie możliwości, w programowaniu gier, dwóch popularnych bibliotek do tworzenia interfejsów graficznych: GTK+ oraz Qt. Badana była zarówno wydajność, jak i dostępność funkcji potrzebnych przy tworzeniu gier.

## 2. Aplikacja testowa

W celu przeprowadzenia badań wydajności stworzona została prosta gra zręcznościowa w dwóch wersjach, odpowiednio dla każdej biblioteki. Przy tworzeniu aplikacji wykorzystano architekturę Entity-Component-System (ECS) [1]. Architektura ta ułatwia rozdzielenie kodu programu na pomniejsze moduły, co pozwoliło na stworzenie dwóch wersji tej samej gry różniących się jedynie podsystemami związanymi z wyświetlaniem bitmap w przestrzeni 2D przechwytywaniem klawiszy.

Celem aplikacji jest zbadanie wydajności w przypadku dużej liczby poruszających się w oknie bitmap. Użycie dostępu do biblioteki OpenGL z wewnątrz GTK+/Qt z poziomu porównywanych frameworków zostało pominięte ponieważ nie jest z nimi bezpośrednio związane. Rysunek 1 przedstawia działanie aplikacji w wersji GTK+ a Rysunek 2 w wersji Qt.




Rys. 1. Aplikacja w wersji GTK+


Rys. 2. Aplikacja w wersji Qt

Wykorzystane oprogramowanie:

- EntityX [2] - darmowa implementacja architektury ECS
- Kompilator Microsoft Visual C++ 2015
- GTK+ 3.6.4
- Microsoft Visual Studio - środowisko programistyczne do wersji GTK+
- QT 5.9.1 MSVC2015 32bit
- QtCreator - środowisko programistyczne do wersji Qt

Platforma testowa:

- Windows 10 64-bit
- Intel Core i7 4700-MQ
- 8 Gb RAM
- Intel HD Graphics 4600

## 3. Porównanie bibliotek

Obie biblioteki analizowane będą pod kątem następujących kryteriów:

- dostępności funkcji
- łatwości implementacji
- dostępu do dokumentacji
- wydajności

### 3.1. Dostępność funkcji

Zarówno biblioteka GTK+ jak i Qt posiadają podstawowe funkcje związane z ich przeznaczeniem, tj. tworzeniem interfejsów graficznych dla aplikacji. W kontekście programowania gier natomiast zapotrzebowanie na funkcje może wykraczać poza kompetencje obu bibliotek. Tabela 1 przedstawia wybrane funkcjonalności przydatne w wytwarzaniu gier.

Tabela 1. Dostępność podstawowych funkcji

| | GTK+ | QT |
|---|---|---|
| Dołączone środowisko | nie | tak |
| Wyświetlanie bitmap | tak | tak |
| Dostęp do OpenGL | tak | tak |
| Obsługa kontrolerów gier | nie | tak |
| Obsługa dźwięków | nie | tak |
| Obsługa sieci | nie | tak |

W przypadku zapotrzebowania na funkcje związane z programowaniem gier biblioteka Qt oferuje zdecydowanie więcej. Natywna obsługa dźwięków oraz kontrolerów gier dzięki modułom Qt Multimedia oraz Qt Gamepad eliminuje potrzebę korzystania z zewnętrznych bibliotek udostępniających te funkcje. W przypadku obsługi audio dodatkowym atutem jest możliwość wykorzystania pozycjonowania źródeł dźwięku, co pozwala graczowi na określenie z jakiego kierunku dany dźwięk się wydobywa [3].

Kolejną zaletą środowiska Qt jest moduł Graphical Effects [4]. Moduł ten zawiera wiele łatwych do zaimplementowania efektów graficznych takich jak:

- Displace - przesuwa pixele bitmapy na podstawie mapy
- Blur - rozmycie
- Motion Blur - rozmycie poruszających się obiektów
- Glow - efekt halo wokół obiektu
- Mask - zakrycie części obiektu

W grach komputerowych istotną rolę pełnią także efekty cząsteczkowe, które biblioteka Qt wspiera natywnie. Emitery cząsteczek w Qt pozwalają na konfigurację wielu parametrów takich jak kształt emitera czy czas życia pojedynczej cząsteczki [5].

### 3.2. Łatwość implementacji

Implementacja biblioteki GTK+ odbywa się w sposób typowy dla języka C/C++. Należy w projekcie użyć odpowiednich plików nagłówkowych, a na etapie kompilacji/linkowania dołączyć odpowiednie biblioteki. W celu określenia potrzebnych do linkowania modułów pomocne jest darmowe narzędzie pkg-config, które

automatycznie wygeneruje potrzebne komendy. Aby jeszcze bardziej ułatwić proces budowania projektu wykorzystać można środowisko Visual Studio. Po konfiguracji i dodaniu potrzebnych ścieżek do plików czy bibliotek budowanie i linkowanie odbywa się w sposób automatyczny.

W przypadku biblioteki Qt implementacja przebiega w sposób bardziej uproszczony. Dołączone do Qt środowisko Qt Creator pozwala na zautomatyzowanie procesu budowania aplikacji. Konfiguracja projektu odbywa się przez edycję pliku konfiguracyjnego [nazwa_projektu].pro, gdzie znajdują się informacje o plikach nagłówkowych, kodzie źródłowym czy też dodatkowych zewnętrznych bibliotekach [6]. Plik konfiguracyjny jest automatycznie aktualizowany w przypadku dodawania/usuwania elementów projektu.

**Tworzenie widoków gry**

Obiekty reprezentowane są w grze przy pomocy widgetów służących do wyświetlania bitmap w oknie. Zarówno GTK+ jak i Qt posiadają takie widgety. Przykład 1 pokazuje jak wyświetla się bitmapę w bibliotece GTK+.

Przykład 1. Tworzenie bitmapy w GTK+

```
GdkPixbuf *img = img_map.at(event.component-
>resourcePath);
GtkWidget *widget = gtk_image_new_from_pixbuf(img);
gtk_layout_put(GTK_LAYOUT(RenderSystem::container),
widget, 0, 0);
entity.assign<View>(widget);
gtk_widget_show(widget);
```

Do utworzenia obiektu korzysta się z funkcji gtk_image_new_from_file lub gtk_image_new_from_pixbuf. Druga opcja jest bardziej optymalna, gdyż w przypadku powtórzenia się tego samego pliku graficznego można wykorzystać już uprzednio wczytany gtk_pixbuf redukując przy tym zużycie pamięci RAM. Utworzony GtkWidget umieszcza się w kontenerze przy użyciu funkcji gtk_layout_put podając jako parametry uprzednio utworzony layout i widget. Następnie uchwyt do widgetu umieszczany jest w komponencie View.

Tworzenie bitmap w Qt odbywa się w nieco inny sposób. Istotną różnicą jest to, że widgety zdefiniowane są w plikach QML. Implementacja przedstawiona jest na Przykładzie 2.

Przykład 2. Tworznie bitmapy w Qt

```
QQmlComponent component(RenderSystem::m_engine,
QUrl::fromLocalFile(QString::fromStdString(event.component
->resourcePath)));
QObject *object = component.create();
QQmlEngine::setObjectOwnership(object,
QQmlEngine::CppOwnership);
QQuickItem *item = qobject_cast<QQuickItem*>(object);
item->setParent(RenderSystem::m_game_container);
item->setParentItem(RenderSystem::m_item_container);
entity.assign<View>(item);
```

Pierwszym krokiem jest wczytanie komponentu Qml a nastepnie utworzenie instancji obiektu przy pomocy funkcji create. Ważnym elementem jest ustawienie własności tego obiekt w funkcji setObjectOwnership. W przypadku tworzenia widgetów z poziomu c++ jest to wymagane, gdyż w innym wypadku widget ten został by usunięty przez mechanizm Garbage Collector silnika Qml.

**RenderSystem**

Za renderowanie obiektów odpowiedzialne są frameworki GTK+ oraz Qt. Celem systemu renderowania natomiast jest aktualizacja pozycji bitmap w oknie na podstawie danych ze stanu gry. Implementację przedstawia Przykład 3.

Przykład 3. Render system w wersji GTK+

```
void RenderSystem::update(entityx::EntityManager &
entities, entityx::EventManager & events, double dt)
{
View::Handle viewComponent;
Position::Handle positionComponent;

for (Entity entity :
entities.entities_with_components(viewComponent,
positionComponent))
{
//GTK+
gint gx = positionComponent->x;
gint gy = positionComponent->y;
gtk_layout_move(GTK_LAYOUT(container), viewComponent-
>view, gx, gy);
// QT
viewComponent->view->setX(positionComponent->x);
viewComponent->view->setY(positionComponent->y);
}
}
```

Funkcja Update w systemie RenderSystem uruchamiana jest co klatkę gry. System ten przetwarza jedynie obiekty z komponentami View (uchwyt do widgeta GTK+/Qt) oraz Position (pozycja w świecie gry). Przy pomocy funkcji entities_with_components biblioteki EntityX następuje iteracja po obiektach zawierających oba te komponenty, a następnie dla każdego widgeta przypisywana jest nowa pozycja na podstawie komponentu Position (gtk_layout_move dla GTK+, setX/setY dla Qt).

### 3.3. Dostęp do dokumentacji

Obie biblioteki posiadają dostęp do wyczerpującej dokumentacji, w której zawarte są m.in. opis klas i funkcji, wprowadzenie, opis poszczególnych modułów czy też poradniki i tutoriale.

Dokumentacja biblioteki Qt [7] jest jednak dużo bardziej rozbudowana i przejrzysta niż GTK+ [8]. Jest to spowodowane tym, że Qt rozwijane jest przez firmę Qt Company, która posiada duże zasoby finansowe, a biblioteka GTK+ wspierana jest przez społeczność.

### 3.4. Wydajność

Testy wydajności przeprowadzone były w warunkach normalnego działania gry, włączając w to wykrywanie kolizji. W trakcie jej działania, co określony interwał czasu, na ekranie gry pojawiał się przeciwnik, który leciał w stronę gracza, a gdy przeciwnik ten opuścił ekran, jego obiekt gry został zniszczony.

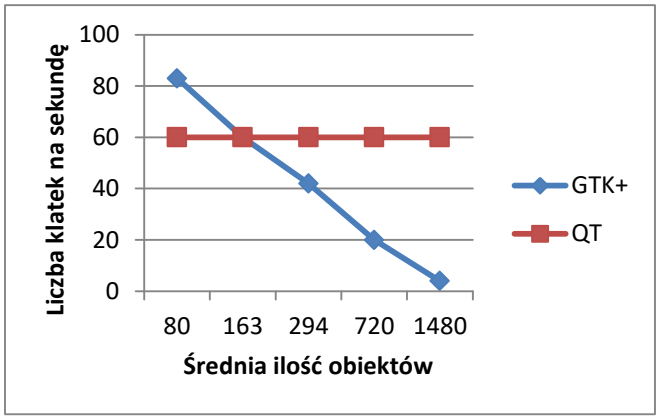
Poprzez modyfikację parametru, określającego wyżej wymieniony interwał czasu, kontrolowana była średnia ilość widocznych na ekranie obiektów. Po jej ustabilizowaniu się mierzono ilość odświeżeń pętli gry na sekundę.

Ze względu na użycie QtQuick 2.0 w przypadku biblioteki Qt, pomiar ilości klatek na sekundę odbywał się również dodatkowo w oknie Qml. Silnik Qml renderuje obiekty na oddzielnym wątku i posiada domyślnie blokadę 60 klatek na sekundę. Tabela 3 przedstawia wyniki pomiarów.

Tabela 2. Wyniki pomiarów wydajności

| Średnia ilość obiektów | GTK+ - pomiar z pętli gry | QT - pomiar z pętli gry | QT - pomiar z silnika Qml |
|---|---|---|---|
| 80 | 83 | 18000 | 60 |
| 163 | 60 | 13100 | 60 |
| 294 | 42 | 8700 | 60 |
| 720 | 20 | 2300 | 60 |
| 1480 | 4 | 355 | 60 |

Rysunek 3 przedstawia wyniki pomiarów wydajności na wykresie. Pominięto pomiar z pętli gry w przypadku Qt.




Rys. 3. Liczba klatek na sekundę w zależności od liczby obiektów

Wyniki pomiarów wskazują na znacznie większą wydajność w przypadku biblioteki Qt. QtQuick 2.0 korzysta z OpenGL(ES) 2.0, dzięki czemu wykorzystywany jest potencjał karty graficznej. Dodatkowo, dzięki zastosowaniu Scene Graph proces renderowania hierarchii obiektów jest bardziej optymalny [9]. Bardzo wysoka liczba iteracji pętli gry na sekundę względem pomiaru z silnika Qml potwierdza fakt, że silnik ten działa na oddzielnym wątku, co pozwoliło na wydajne przetwarzanie logiki gry. Biblioteka Qt nawet przy bardzo dużej ilości obiektów zachowywała stabilne 60 klatek na sekundę.

Biblioteka GTK+ cechowała się mniejszą wydajnością. Wraz ze wzrostem ilości obiektów na scenie ilość klatek na sekundę stale spadała, osiągając w najgorszym przypadku nawet 4 kl/s przy około 1500 obiektach. Tak niski wynik spowodowany był faktem, że GTK+ działa jednowątkowo i nie wykorzystuje przyspieszenia sprzętowego karty graficznej [10].

## 4. Wnioski

W pracy przebadano i porównano możliwości dwóch popularnych bibliotek do tworzenia interfejsów graficznych: GTK+ oraz Qt. Obie biblioteki umożliwiają programowanie gier w różnym stopniu.

Z przeprowadzonej analizy wynika, że biblioteka Qt cechuje się znacznie większą wydajnością, a także większą ilością funkcji przydatnych przy tworzeniu gier. Przyspieszenie sprzętowe pomaga zapewnić płynne działanie aplikacji, a dodatki takie jak efekty cząsteczkowe czy efekty graficzne mogą w łatwy sposób zwiększyć jej walory wizualne.

Biblioteka GTK+, ze względu na jej słabą wydajność przy dużej ilości obiektów, nie nadaje się do zbyt skomplikowanych projektów. Może ona sprawdzić się w grach typu szachy, czy pasjans, gdyż tego typu gry nie są zwykle wymagające graficznie ani nie wymagają dużej płynności animacji.

Biblioteka Qt, ze względu na dołączone narzędzie Qt Creator, jest znacznie łatwiejsza w implementacji. Dokumentacja biblioteki Qt jest także bardziej przejrzysta oraz przyjazna dla użytkownika, niż w przypadku GTK+, co zwiększa efektywność i wygodę pracy z tym frameworkiem.

Obie biblioteki spełniają w pełni swoje zdanie, tj. tworzenie atrakcyjnych interfejsów graficznych. Jednak w przypadku programowania gier biblioteka Qt oferuje zdecydowanie więcej możliwości.

## Literatura

[1] https://www.gamedev.net/articles/programming/general-and-gameplay-programming/understanding-component-entity-systems-r3013 [20.09.2017]

[2] https://github.com/alecthomas/entityx [20.09.2017]

[3] http://doc.qt.io/qt-5/qml-qtaudioengine-audioengine.html [20.09.2017]

[4] http://doc.qt.io/qt-5/graphicaleffects.html [20.09.2017]

[5] http://doc.qt.io/qt-5/qtquick-effects-particles.html[20.09.2017]

[6] D. Molkentin, The art of building Qt Applications, William Pollock, 2007

[7] http://doc.qt.io/ [20.09.2017]

[8] https://developer.gnome.org/gtk3/stable/ [20.09.2017]

[9] W. Wysota, L. Haas, Game Programming Using Qt, Packt Publishing, 2016.

[10] A. Krause, Foundations of GTK+ Development, Apress, 2007.