Spis treści

OPROGRAMOWANIE

1. INFORMACJE OGÓLNE - JAGIELSKI

Kluczowymi elementami pracy było stworzenie oprogramowania umożliwiającego odtwarzanie nieskompresowanych sekwencji wideo oraz przeprowadzenie testów subiektywnych z użyciem autorskiego odtwarzacza.

2. WYBÓR NARZĘDZI - JAGIELSKI & ORLIŃSKI

Wybór narzędzi wiązał się z koniecznością przeglądu dostępnych zasobów ludzkich pod kątem umiejętności tworzenia oprogramowania w danym języku. Kolejną niezbędną kwestią była analiza istniejących rozwiązań w celu znalezienia jak najlepszego narzędzia do rozwiązania problemu.

Odtwarzacz nieskompresowanych sekwencji wideo musi charakteryzować się jak największą wydajnością. Poprzez wydajność rozumiane jest jak najlepsze zarządzanie zasobami w taki sposób, aby na dostępnym sprzęcie komputerowym uzyskać jak najlepsze parametry wyświetlania kolejnych klatek. Zdecydowano się zrezygnować z języków korzystających z maszyn wirtualnych na korzyść takich, które pozwalają na dużą swobodę w zarządzaniu pamięcią. Wybór padł na język C++ ze względu na doświadczenie autorów w pracy z nim, zarówno zawodowe jak i nabyte podczas studiów.

Wybranym systemem operacyjnym został Linux, dystrybucja Xubuntu 16.04.2 posiadająca jądro w wersji 4.4.0-72-generic. Decydującymi czynnikami były łatwość instalowania kolejnych pakietów bibliotek, dostęp do narzędzi konwertujących parametry filmów oraz niskie zużycie zasobów sprzętowych przez biernie działający system.

[https://xubuntu.org/]

Zdecydowano się użyć środowiska programistycznego CLion dostarczanym przez firmę JetBrains, korzystając z licencji studenckiej, która pozwala na użycie IDE (ang. *Integrated Development Environment*) w celach edukacyjnych.

[https://www.jetbrains.com/]

Analiza istniejących odtwarzaczy wideo pozwoliła wyselekcjonować bibliotekę, która dała możliwość darmowego użycia i jak największej możliwości modyfikacji, czyli posiadała licencję *open source*. Ze względu na popularność odtwarzacza VLC w systemach operacyjnych Linux, podjęto decyzję o wykorzystaniu biblioteki z użyciem której powyższy odtwarzacz został stworzony – libVLC.

[http://www.videolan.org/vlc/libvlc.html]

FFmpeg, czyli narzędzie pozwalające na edycje parametrów wzorcowego wideo zostało wybrane ze względu na łatwość użycia, swobodę w wyborze zmienianych aspektów filmu oraz licencję *open source.* Wszystkie użyte w pracy sekwencje wideo zostały wygenerowane przy pomocy FFmpeg z filmów wzorcowych.

[https://ffmpeg.org/]

QT to zestaw bibliotek dedykowanych dla m.in. języka C++ pozwalający na tworzenie zaawansowanych interfejsów użytkownika. *QT* zawiera również elementy pozwalające na obsługę procesów, sieci i grafiki trójwymiarowej a także integrację z bazami danych, posiada także narzędzia pozwalające na przeprowadzenie lokalizacji dla innych wersji językowych programu. Wykorzystane narzędzia zostały szczegółowo omówione w części pracy poświęconej graficznemu interfejsowi użytkownika. Wybór bibliotek *QT* jako narzędzia do tworzenia *GUI* (G*raphic User Interface*) był niejako oczywisty ze względu na fakt, iż VLC wykorzystuje jako QT jako jeden z podstawowych interfejsów użytkownika w odtwarzaczu podstawowym. Kluczowa była także opinia Jean-Baptiste Kempfa jednego z autorów i moderatorów biblioteki *VLC*, a także prezydenta i administratora forum jej poświęconego. Po przeanalizowaniu innych możliwości uznano, iż jest to jedyny dostępny produkt dający możliwość swobodnego tworzenia paneli np. do oceny obejrzanego filmu.

QT Creator to jedno z narzędzi udostępnianych w ramach zestawu bibliotek *QT*. Jest to kolejne użyte w pracy środowisko programistyczne pozwalające nie tylko na edycje kodu źródłowego, ale także (w narzędziu *QT Designer*) na projektowanie graficznego interfejsu użytkowania w okienkowym edytorze za pomocą widżetów z biblioteki QT. Za pomocą dodatkowych narzędzi *uic* i *moc* każda klasa korzystająca z sygnałów i slotów *QT* (reprezentująca element GUI) otrzymuje dodatkowe pliki z rozszerzeniem *cpp* i *ui* reprezentujące rozmieszczenia okien, ustawienia grafiki, standardowe zdarzenia i przypisane do nich metody.

VLC-QT darmowa biblioteka autorstwa Tadej Novaka służąca do połączenia bibliotek *QT* z biblioteką *libvlc*. Pozwala ona na stworzenie prostego odtwarzacza wraz z dowolnym interfejsem

pozwalającym na kontrolowanie odtwarzania. *VLC-QT* nakrywa metody i klasy upraszczając użycie *libvlc* w oknach *QT*. Ze względu na ograniczenia spowodowane przez użycie interfejsu *imem* do wczytywanie klatek nieskompresowanych sekwencji wideo, użycie *VLC-QT* zostało ograniczone do użycia widżetu wideo dającego większe możliwości niż standardowe *QFrame*.

3. Proces tworzenia - Jagielski

Organizacja pracy w grupie jest zależna od ilości jej członków, natury rozwiązywanego problemu oraz wielu innych czynników np. ograniczeń czasowych lub sposobu prezentacji postępów.

Obecnie w informatyce dąży się do wdrożenia tak zwanych metod zwinnych programowania. Pozwalają one zminimalizować straty w przypadku, gdy część wymogów klienta ulegnie zmianie. Wynika to z faktu, że kod źródłowy programu dostarczany jest iteracyjnie w jak najmniejszych częściach. Podejście to pozwala również zmaksymalizować czas faktycznej pracy każdego z pracowników, ponieważ eliminowane są sytuacje, w których pojawia się konieczność oczekiwania na wartość dostarczaną przez innych pracowników.

Oprogramowanie przedstawione w pracy tworzone było w zespole dwuosobowym. Jednym z pierwszych kroków podczas realizacji pracy magisterskiej było zdefiniowanie zasad współpracy między opiekunem, uczelnią a osobami odpowiedzialnymi za nią samą. W związku z koniecznością dotrzymania terminów odgórnie narzuconych przez uczelnie zdecydowano się wyznaczyć kilka terminów wraz z zakresem funkcjonalności, które na dany termin miały by być dostarczone. Pozwoliło to zsynchronizować wiedzę o projekcie między studentami, a prowadzącym oraz na skupienie się na wyznaczonych celach. Dopiero po akceptacji postępów przez opiekuna następowało przejście do kolejnego etapu. Na każdym ze spotkań projektowych przeprowadzano pokaz aktualnej wersji, na którym omawiano aktualny sposób działania, najnowsze zmiany w funkcjonowaniu programu oraz wyszukiwano ewentualne niedociągnięcia.

W czasie realizacji pracy magisterskiej obaj członkowie zespołu pracowali zawodowo oraz uczęszczali na zajęcia prowadzone na uczelni. W związku z tym proces tworzenia pracy magisterskiej musiał być dostosowany do ich obłożenia czasowego. Naturalnym wyborem było więc działanie w metodyce *Kanban* – skupiającej się na produkcie, bez dodatkowych, niepotrzebnych nakładów pracy oraz minimalizacji bezczynności. Stworzono prostą *tablicę kanbanową* czyli złożenie danych o postępie prac nad projektem przy podziale na poszczególne jego zadania. Zaletą takiego rozwiązania jest wiedza o przewidywanym terminie zakończenia prac oraz dowolność w wyborze czasu, w którym są one realizowane (oprócz terminów końcowych). Takie podejście pozwoliło powiązać pracę zawodową z rozwojem naukowym.

System kontroli wersji zastosowany w projekcie pozwolił na równoczesną pracę nad wieloma funkcjonalnościami programu, wprowadził historię zmian oraz pozwolił na bezproblemową ich synchronizację między członkami zespołu. Wykorzystano popularną platformę *GitHub*, która dostarcza gotowe rozwiązania wraz z serwerem do przechowywania danych.

[https://github.com/Kondix/UHDPlayer]

4. Analiza systemu - Jagielski

Analiza systemu została przeprowadzona przed implementacją rozwiązania. Podczas tej fazy stworzono wymagania, które będą stanowiły o tym kiedy projekt można uznać za zakończony. Ich definicja była konieczna, aby ukierunkować tok prac.

Lista wymagań projektowych:

- Odtwarzanie nieskompresowanych sekwencji wideo o wybranych parametrach
- Utrzymanie jakości filmu przez cały jego czas trwania
- Możliwość przeprowadzenia różnych scenariuszy testowych
- Możliwość wygenerowania filmów o zadanych parametrach z filmu źródłowego
- Automatyczna prezentacja filmów osobie testowanej
- Zbieranie danych o ocenie filmu osoby testowanej

Wszystkie powyższe wymagania są przedstawione z punktu widzenia nietechnicznego, co pozwala na ich zrozumienie również osobie, która nie posiada wiedzy specjalistycznej.

Testy subiektywne przeprowadzane są z użyciem nieskompresowanych sekwencji wideo, w celu wyeliminowania wpływu procesu dekodowania na otrzymany obraz, a co za tym idzie otrzymania lepszej jakości wyników samego testu. Różne implementacje tego samego sposobu dekodowania mogłyby spowodować, że przeprowadzenie teoretycznie tego samego testu na dwóch różnych, niezależnych maszynach, otrzyma wyniki dla rożnych obrazów widzianych przez osoby testujące.

Odtwarzacz takich sekwencji oraz maszyna, na której jest uruchomiony powinny być skonfigurowane tak, aby film w zadanej jakości odtwarzał się płynnie, wyświetlane były wszystkie klatki oraz ich kolejność i jakość została zachowana. Bardzo ważnym elementem analizy było ustalenie zbioru jakości, które odtwarzacz będzie mógł odtworzyć. Ograniczeniem w tej kwestii pozostaje dostępny sprzęt elektroniczny, budżet na zakup nowych części i rozwiązań oraz sama implementacja odtwarzacza. O tym czy zapewniona została odpowiednia jakość filmów decydował zestaw testów wydajnościowych, o których więcej pojawi się w dalszej części pracy.

Przeprowadzenie testów powinno odbywać się z jak najmniejszym udziałem człowieka, tj. po uprzednim przygotowaniu scenariuszy kolejne uruchomienia testu powinny być zautomatyzowane, a sama osoba testująca powinna być „prowadzona za rękę” przez cała długość jego trwania. Tester musi mieć dodatkowo możliwość interakcji z systemem w celu podawana kolejnych odpowiedzi na zadane pytania.

Kluczową dla pracy magisterskiej jest kwestia porównania między sobą wybranych scenariuszy testowych. Dlatego wymagana jest również możliwość ich przeprowadzenia, a co za tym idzie skonfigurowania. Konfiguracja i struktura plików nie powinna być dostępna dla testera ze względu na przykład na ich nazwy, które podczas generacji przez skrypt zawierają sugestie, w jakiej jakości zapisany jest dany film.

Jakości poszczególnych filmów testowych powinny być definiowane przez administratora lub osobę za to odpowiedzialną, a nie być narzucone odgórnie przez skrypt, ze względu na różnice pomiędzy koncepcjami testowymi. Niektóre scenariusze mogą zakładać użycie materiałów o nieznaczne zmienionych parametrach, inne korzystać z takich, których parametry różnią się znacznie. Generacja filmów powinna odbywać się automatycznie, korzystając z pliku źródłowego oraz informacji o pożądanej jakości wynikowego filmu. Należy zaznaczyć, że skrypt generujący został stworzony tylko na potrzeby wewnętrzne projektu w celu zaspokojenia zapotrzebowania na dane wejściowe systemu. Użytkownik końcowy powinien zapewniać jakość sekwencji wideo we własnym zakresie.

5. Szczegóły implementacyjne - Jagielski

Kolejnym krokiem po szczegółowej analizie systemu było zaplanowanie architektury kodu. Naturalnym początkiem było wydzielenie części odpowiedzialnej za przetwarzanie i wyświetlanie nieskompresowanych sekwencji wideo. Po ukończeniu odtwarzacza ruszyły prace, mające na celu stworzenie warstwy widocznej przez użytkownika, wraz z funkcją przeprowadzenia wybranych scenariuszy testowych.

Biblioteka *libVLC* zapewnia dostęp do wielu gotowych metod obsługi materiału wideo, nie posiada jednak bezpośrednich rozwiązań do odtwarzania nieskompresowanych sekwencji wideo. W przypadku sekwencji skompresowanych jednym ze standardowych sposobów odtwarzania jest wczytywanie nie całych klatek, a tylko zmian zachodzących między kolejnymi dwoma poprzez dekompresje. Takie podejście pozwala znacząco ograniczyć rozmiar wczytywanych do pamięci danych. Mniejsza ilość danych pozwala na zwiększenie maksymalnej jakości filmu odtwarzanego na tym samym komputerze względem programu ładującego każdą klatkę od nowa.

W przypadku inkrementacyjnego ładowania zmian w jednym z ostatnich kroków procesu otrzymywana jest klatka reprezentowana w ten sam sposób, co nieskompresowana. Dzięki tej obserwacji, możliwe jest wstrzyknięcie wczytanej nieskompresowanej klatki do standardowego procesu odtwarzania filmu i jej wyświetlenie z użyciem biblioteki *libVLC*. Szczegółowy opis implementacji tego sposobu zostanie przedstawiony w dalszej części rozdziału.

5.1. Podstawy działania programu - Jagielski

W tej części opisany zostanie sposób na stworzenie najprostszego programu z użyciem biblioteki *libVLC*. Program otworzy plik wideo dostępnego na dysku lokalnym, lub zasobie sieciowym a następnie odtworzy zdefiniowaną wcześniej długość filmu, np. 10 sekund. Program nie posiada wiedzy o długości

trwania dostępnego materiału, dlatego w przypadku gdy film jest krótszy od zadanej wartości program przerwie swoje działanie zwracając odpowiedni kod błędu. Później zostaną również opisane podstawowe typy dostarczane przez bibliotekę.

Pierwszym, podstawowym typem biblioteki jest *libvlc_instance_t*, struktura reprezentująca całą instancję *libVLC*. Obiekt tego typu inicjalizowany jest przez funkcję *libvlc_new*, zwracającą typ wskaźnika na wyżej wymienioną strukturę lub *NULL* gdy podczas jej wykonywania wystąpił błąd. Funkcja inicjalizująca przyjmuje dwa parametry, podobne do tych z funkcji głównej programu. Pierwszy z nich *argc* oczekuje danych w formacie całkowitoliczbowym z ilością argumentów przekazanych w postaci stałego ciągu znaków *argv*. Przekazywane argumenty są bardzo ważne z punktu widzenia pracy magisterskiej. To dzięki nim możliwe było sterowanie odtwarzanym filmem w postaci nieskompresowanej, co stanowiło kluczowy element podczas jej realizacji. Szczegółowy opis przekazanych argumentów zostanie przedstawiony wraz z interfejsem do zarządzania pamięcią w dalszej części rozdziału. Na potrzeby prostego programu przyjęto, że nie będą przekazywane żadne argumenty, a więc parametry będą ustawione kolejno na 0 i *NULL*.

Kolejnym ważnym typem dostarczanym przez bibliotekę jest *libvlc_media_t* odpowiadającym za reprezentację odtwarzanych mediów. Przez media rozumiane są dowolne, wspierane przez bibliotekę dane w odpowiednim formacie. Mogą to być przykładowo filmy znajdujące się na dysku twardym komputera lub sekwencje wideo dostępne w Internecie. W zależności od źródła danych biblioteka dostarcza różne funkcje, pozwalające na stworzenie obiektu wyżej wymienionego typu.

Lista dostępnych funkcji inicjujących obiekt typu *libvlc_media_t.* Każda z poniższych funkcji jako pierwszy argument przyjmuje wskaźnik na obiekt instancji *libVLC - llibvlc_instance_t.*

- *libvlc_media_new_location* – pozwala na obsługę mediów dostępnych zarówno w sieci jak i na lokalnej maszynie. Jako drugi parametr przyjmuje ścieżkę do żądanego zasobu (np. odnośnik protokołu http) w postaci ciągu znaków.

- *libvlc_media_new_path* – służy do obsługi mediów dostępnych w lokalnym systemie plików. Jako drugi parametr przyjmuje ścieżkę do odtwarzanego pliku w postaci ciągu znaków.

- *libvlc_media_new_fd* – umożliwia obsługę mediów z otwartego wcześniej deskryptora pliku. Jako drugi parametr przyjmuje otwarty deskryptor pliku w formacie całkowitoliczbowym.

Ostatnim z omawianych typów jest *libvlc_media_player_t* czyli typ reprezentujący odtwarzacz mediów. Pozwala on na odtworzenie w oknie wybranych mediów. W programie podstawowym nie będzie on wymagał żadnej wstępnej konfiguracji. Tworzony jest z użyciem funkcji *libvlc_media_player_new_from_media*, która jako parametr przyjmuje wskaźnik na obiekt reprezentujący media, oraz zwraca wskaźnik na nowo powstały obiekt lub *NULL* w przypadku niepowodzenia. Po wywołaniu funkcji, a więc zainicjowaniu obiektu odtwarzacza mediów, same media nie są już wymagane. Usunięcia mediów można dokonać z użyciem funkcji *libvlc_media_release*, przyjmującej jako parametr wskaźnik na usuwane media.

Kolejnym krokiem programu jest uruchomienie odtwarzacza mediów. Realizowane jest to za pomocą funkcji *libvlc_media_player_play*, która jako parametr przyjmuje wskaźnik na obiekt odtwarzacza mediów. W tym momencie następuje uruchomienie wyświetlania filmu. Program posiada podstawowe dane na temat filmu, jednak nie wie jak długo trwa. Dlatego w kolejnym kroku wołana jest funkcja *sleep*, która powoduje zatrzymanie wykonywania wątku, z którego została wywołana na określony w parametrze czas, wyrażony w sekundach. Problem pojawia się, gdy długość filmu nie przekracza wartości na którą wątek został uśpiony. Taki scenariusz powoduje nagłe zatrzymanie wykonywania programu, który kończy się zwracając odpowiedni kod błędu. Rozwiązanie zostanie przedstawione w dalszej części rozdziału, wraz z opisem bardziej zaawansowanego kodu.

Jeśli film okazał się dłuższy od zadanej wartości uśpienia, program wykonuje się dalej. Następnym etapem jest zatrzymanie odtwarzacza mediów funkcją *libvlc_media_player_stop*. Podobnie jak rozpoczęcie odtwarzania, jako parametr przyjmuje wskaźnik na odtwarzacz mediów. Funkcja *libvlc_media_player_release* zwalnia obiekt odtwarzacza mediów. Po jej wywołaniu obiekt nie nadaje się do ponownego użycia. Konieczna jest jego ponowna inicjalizacja.

Ostatnim krokiem jest zwolnienie obiektu instancji, poprzez wywołanie *libvlc_release*, przyjmującej jako parametr wskaźnika na zwalnianą instancję.

```c
#include <stdio.h>
#include <stdlib.h>
#include <vlc/vlc.h>

int main(int argc, char* argv[])
{
    libvlc_instance_t * inst;
    libvlc_media_player_t *odtwarzacz_mediow;
    libvlc_media_t *media;

    inst = libvlc_new (0, NULL);
    media = libvlc_media_new_location (inst, "http://sciezka.do.mediow/film.mov");

    odtwarzacz_mediow = libvlc_media_player_new_from_media (media);
    libvlc_media_release (media);
    libvlc_media_player_play (odtwarzacz_mediow);

    sleep (10);

    libvlc_media_player_stop (odtwarzacz_mediow);
    libvlc_media_player_release (odtwarzacz_mediow);
    libvlc_release (inst);

    return 0;
}
```

Rysunek 1 Widok ekranu z kodem źródłowym programu podstawowego.

[https://wiki.videolan.org/LibVLC_Tutorial/]
[https://www.videolan.org/developers/vlc/doc/doxygen/html/]

Na rysunku 1 zamieszczono cały kod omawianego programu. W trakcie jego pisania korzystano ze środowiska *CLion*, które opiera się na *cmake*. Jest to narzędzie skryptowe o otwartej licencji pozwalające zbudować (ang. *build*) dany program w różnych systemach operacyjnych, w tym używanego w pracy *Linuxa*. Jego działanie opiera się na sekwencyjnym wykonywaniu kolejnych dyrektyw i tworzeniu plików z regułami, wykorzystywanymi później przez kompilator. Aby zapewnić dostęp do wszystkich potrzebnych źródeł i bibliotek stworzono listę poleceń przedstawioną na rysunku 2.

```
1  cmake_minimum_required(VERSION 3.6)
2  project(Odtwarzacz)
3
4  set(CMAKE_CXX_STANDARD 11)
5
6  set(SOURCE_FILES main.cpp)
7  set(VLC_LIB /usr/lib/libvlc.so)
8  add_executable(Odtwarzacz ${SOURCE_FILES})
9  LINK_DIRECTORIES(${VLC_LIB})
10 TARGET_LINK_LIBRARIES(Odtwarzacz libvlc.so -lpthread -lm)
```

Rysunek 2 Widok ekranu z listą dyrektyw narzędzia cmake.

Polecenie *cmake_minimum_required* określa jaka najstarsza wersja *cmake* może być użyta do wykonania skryptu. Dyrektywa *set* pozwala na ustawienie zmiennej, której nazwa jest przekazywana jako pierwszy parametr, na wartość przekazywaną jako drugi. Może to być zarówno zmienna środowiskowa jak i lokalna. Komenda *project* ustawia nazwę projektu, w tym przypadku na „Odtwarzacz”. Od tego momentu nazwa jest kojarzona z konkretnym projektem i jej użycie spowoduje odniesienie się do niego. *Add_executable* dodaje pliki źródłowe do projektu w którym mają zostać zbudowane. W tym przypadku dodany jest plik *main.cpp* zawierający wcześniej opisywany program. W *cmake* aby odnieść się do wartości przechowywanej przez zmienna należy użyć składni: nazwa zmiennej w nawiasach klamrowych, poprzedzonych przez znak dolara. Polecenie *LINK_DIRECOTIES* specyfikuje ścieżkę w której konsolidator (ang. *linker*) powinien szukać bibliotek, z których powstanie plik wykonywalny. Ścieżka do biblioteki *libVLC* w przykładzie jest przekazywana przez odwołanie do wartości zmiennej *VLC_LIB*. Ostatecznie dyrektywa *TARGET_LINK_LIBRARIES* specyfikuje które konkretnie biblioteki mają być użyte przez konsolidator podczas budowania danego projektu. Jako parametr przyjmuje też flagi z jakimi proces ma zostać przeprowadzony.

5.2. Wstrzykiwanie klatek do pamięci - Jagielski

Jednym z największych problemów dotyczących tej części pracy było napisanie odtwarzacza w taki sposób, aby możliwe było odtwarzanie nieskompresowanych sekwencji wideo. Z powodu braku dostępu do materiałów wyjaśniających sposób w jaki można to osiągnąć zdecydowano się szczegółowo opisać problem oraz sposoby jego rozwiązania. W trakcie rozwoju odtwarzacza pojawiło się wiele ograniczeń które również zostaną opisane w poniższym rozdziale.

Podczas prowadzenia analizy istniejących rozwiązań dla tego typu odtwarzaczy natknięto się na wątek dotyczący przesyłania obrazu z kamery za pomocą Internetu oraz protokołu UDP. Autor pytania chciał odbierać dane i przekazywać je programowi napisanemu z użyciem *libVLC* z pomocą interfejsu *imem*. Moduł interfejsu nie cieszy się dużą popularnością, nie jest również opisany w dokumentacji dostarczonej przez autorów *libVLC*. Największym źródłem informacji o nim zdobyto podczas analizy statycznej kodu źródłowego, który na szczęście jest dostępny na zasadach otwartej licencji. Korzystano również z informacji dostarczonych przez użytkownika Arkaid z wątku na forum videolan.

[https://forum.videolan.org/viewtopic.php?t=93842]

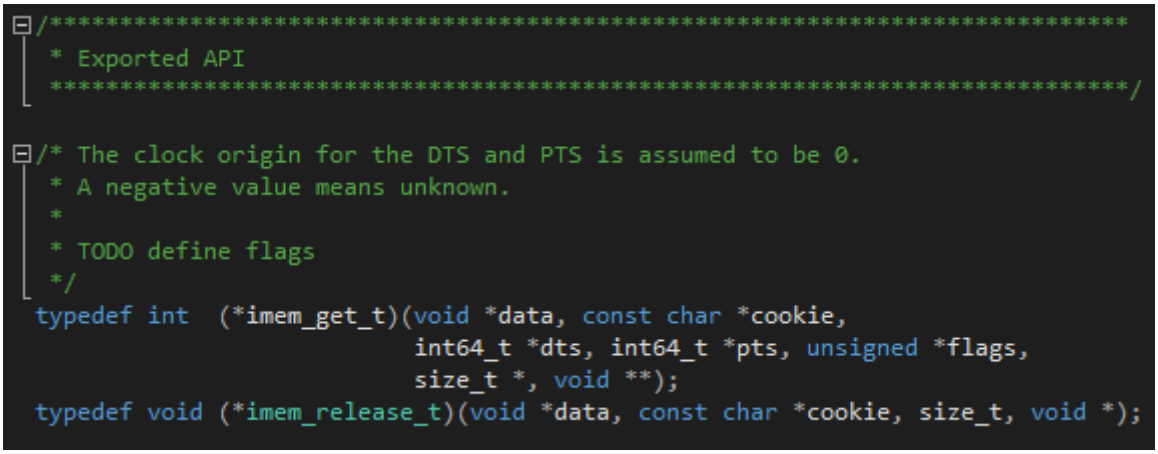
Bazując na programie podstawowym opisanym w poprzednim rozdziale stworzono kod z wykorzystaniem interfejsu *imem*. Jest to moduł pozwalający na dostęp do pamięci komputera i użycia danych przechowywanych bezpośrednio w niej. Sterowanie modułem odbywa się dzięki zestawowi komend, które zostaną omówione później.

```
Controler(std::string sFileLocation, std::vector<const char*> vcOptions)
{
    m_VLCInstance = libvlc_new(int(vcOptions.size()), vcOptions.data());
    m_pMedia = libvlc_media_new_location (m_VLCInstance, "imem://");
    m_DisplayHandler = new DisplayHandler(m_VLCInstance, m_pMedia);
};
```

Rysunek 3 Widok ekranu z konstruktorem parametrycznym klasy Controler.

Pierwszą zmianą jest ustawienie źródła mediów na omawiany interfejs. Deklaracja źródła mediów odbywa się poprzez omówioną w poprzednim rozdziale funkcję *libvlc_media_new_location*. Jako drugi parametr przekazany został ciąg znaków „imem://”. Przykład implementacji znajduje się na rysunku 3. Użycie obiektu reprezentującego media, w którym użyto modułu *imem*, w programie podstawowym zakończy się jednak niepowodzeniem. Wymagane jest wprowadzenie kilku kolejnych zmian implementacyjnych aby zmienić ten stan rzeczy.

Podstawą działania modułu *imem* są funkcje nazwane przywołaniami (ang. *callback function).* Odpowiadają one za obsługę każdej klatki filmu. Wyodrębnione zostały dwie funkcje tego typu. Pierwsza odpowiedzialna jest za alokację pamięci oraz ustawienie szeregu zmiennych odpowiedzialnych za konkretne parametry wyświetlanej klatki. Druga odpowiada za ewentualne zwolnienie zaalokowanej pamięci. Oczywiście obie funkcje można edytować do swoich własnych potrzeb.



```
/*****************************************************************************
 * Exported API
 *****************************************************************************/

/* The clock origin for the DTS and PTS is assumed to be 0.
 * A negative value means unknown.
 *
 * TODO define flags
 */
typedef int  (*imem_get_t)(void *data, const char *cookie,
                           int64_t *dts, int64_t *pts, unsigned *flags,
                           size_t *, void **);
typedef void (*imem_release_t)(void *data, const char *cookie, size_t, void *);
```

Rysunek 4 Widok ekranu z kodem źródłowym interfejsu imem. Definicje typów funkcji przywołań.

Na rysunku 4 przedstawiono definicje typów funkcji przywołań. Jest to jedno z ograniczeń, które należy uwzględnić podczas modyfikacji którejkolwiek z funkcji. Przekazane parametry funkcji muszą pokrywać się z tymi, które znajdują się w pliku imem.c dostarczanym wraz z biblioteką *libVLC*. Omówione zostaną tylko parametry przekazywane w oryginalnej, niezmodyfikowanej wersji biblioteki.

```
char imemGetArg[256];
sprintf(imemGetArg, "--imem-get=%#p", Callbacks::MyImemGetCallback);
optionsHandler.AddOption(imemGetArg);

char imemReleaseArg[256];
sprintf(imemReleaseArg, "--imem-release=%#p", Callbacks::MyImemReleaseCallback);
optionsHandler.AddOption(imemReleaseArg);
```

Rysunek 5 Widok ekranu z kodem odpowiedzialnym za ustawienie adresów funkcji przywołań.

Jak zostało już wspomniane interfejs *imem* konfigurowany jest przez zestaw komend. Komendy te przekazywane są do programu z użyciem omówionej już funkcji *libvlc_new.* Widoczna na rysunku 5 metoda *AddOption* obiektu *optionsHandler* odpowiada za agregację kolejnych argumentów, które ostatecznie przekazywane są do funkcji *libvlc_new.* Od tego momentu użycie obiektu instancji *llibvlc_instance_t* będzie wiązać się z użyciem stworzonej konfiguracji.

Argumenty „--imem-get” oraz „--imem-release” wymagają ustawienia na adresy odpowiadających im funkcji. Odbywa się to dzięki funkcji *sprintf*, która potrafi wpisać do wskazanej zmiennej ciąg znaków. Konwersja nazwy funkcji na jej adres odbywa się dzięki składni: litera ‘p’ poprzedzona znakiem kratki.

Lista parametrów alokującej funkcji przywołań:

- data – pierwszy z parametrów, odpowiadający za dane w formacie zdefiniowanym przez użytkownika. Adres przekazywany jest za pomocą argumentu „--imem-data”. Adres może zawierać dowolny typ obiektu. Dzieje się tak za sprawą typu *void** czyli wskaźnika do obiektu o nieznanym typie. Jedynie jawne rzutowanie wskaźnika na inny typ jest bezpieczne, ponieważ w innym przypadku kompilator nie wie na jaki typ naprawdę wskazuje. Przykład przedstawiony jest na rysunku 6, gdzie *data* konwertowana jest na typ *FramesHandler**, czyli wskaźnik na obiekt zdefiniowany przez użytkownika.
  [Stroustrup]

- cookie – ciasteczko (ang. *cookie)*, zdefiniowany przez użytkownika ciąg znaków. Może służyć na przykład do wyświetlania napisów w danej klatce. Adres przekazywany jest za pomocą argumentu „--imem-cookie”. W przykładzie przedstawionym na rysunku 6 ciasteczko jest pomijane ze względu na brak konieczności jego użycia w programie.

- dts - znacznik czasowy definiujący moment zdekodowania klatki (ang. *decode timestamp)*. Wyrażony w mikrosekundach obliczanych według wzoru:

$$\frac{1}{częstotliwość\ dekodowania} = dts$$

  Obliczona według wzoru wartość dodawana jest do poprzedniej. W ten sposób, dzięki inkrementacji implementacja może czerpać wiedzę o tym, kiedy dana klatka ma zostać wyświetlona .W przykładzie przedstawionym na rysunku 6 ustawiany wraz ze zmienna p*ts* na tą samą wartość, jednak może się ona różnić w zależności od użytego sposobu dekodowania. Wyświetlanie nieskompresowanych sekwencji wideo nie wymaga procesu dekompresji, co pozwala na równość dwóch współczynników.

- pts - znacznik czasowy definiujący moment wyświetlenia klatki (ang. *presentation timestamp)*. Wyrażony w mikrosekundach obliczanych według wzoru:

$$\frac{1}{ilość\ klatek\ na\ sekundę} = pts$$

  Obliczona według wzoru wartość dodawana jest do poprzedniej. W ten sposób, dzięki inkrementacji implementacja może czerpać wiedzę o tym, kiedy dana klatka ma zostać wyświetlona .
- flags - parametr typu *unsigned** czyli wskaźnik na obiekt całkowitoliczbowy bez znaku. W przykładzie przedstawionym na rysunku 6 nie został użyty.

- bufferSize - parametr typu *size_t** czyli wskaźnik na obiekt typu całkowitoliczbowego bez znaku, który może przechowywać rozmiar każdego obiektu, wyrażony w bajtach. W tym przypadku przekazywany jest rozmiar następnego parametru, czyli bufora danych. Przekazany błędnie spowoduje obcięcie wartościowych dla danej klatki danych lub wczytanie zbyt dużej ich ilości wraz ze zbędnymi, trudnymi do określenia bitami. W każdym z wyżej wymienionych przypadków klatka zostanie błędnie wyświetlona lub cały program zakończy działanie zwracając odpowiedni kod błędu.

- buffer - najważniejszy parametr odpowiedzialny za dostarczenie do interfejsu *imem* danych o klatce. Wskaźnik ustawiany jest na podany adres, a następnie brane jest tyle bajtów danych, ile wynosi zmienna bufferSize. W tym przypadku ważne jest, aby pamięć była zaalokowana w jednym bloku, to znaczy wszystkie dane klatki znajdowały się na sąsiadujących adresach w pamięci. Jeśli wywołana została funkcja *new*, aby uniknąć sytuacji gdzie zaalokowana pamięć jest nie zostaje nigdy zwolniona, w zwalniającej funkcji przywołań należy wywołać funkcję *delete*. W przykładzie przedstawionym na rysunku 6 niealokowana jest żadna pamięć, dlatego nie ma potrzeby, aby potem ją zwalniać.

```
int MyImemGetCallback (void* data, const char* , int64_t *dts, int64_t *pts, unsigned* , size_t* bufferSize, void** buffer)
{
    g_mtx.lock();
    FramesHandler* framesHandler = (FramesHandler*)data;

    *bufferSize = iFrameH*iFrameW*iFrameDepth/iBitByte;

    if (framesHandler->GetFramesCount() > 0)
    {
        *buffer = framesHandler->GetFrame(0);

        int64_t iSetUp = framesHandler->GetPts() + iFPS;

        framesHandler->SetDts(iSetUp);
        framesHandler->SetPts(iSetUp);
        *dts = *pts = iSetUp;
    }
    g_mtx.unlock();
    return 0;
}
```

Rysunek 6 Widok ekranu z kodem alokującej funkcji przywołań.

Na rysunku 6 przedstawiono implementację alokującej funkcji przywołań, którą wykonano dla potrzeb pracy magisterskiej. Funkcja *MyImemGetCallback* zaczyna się i kończy operacjami na obiekcie typu *std::mutex*. Obiekty tej klasy są używane do reprezentacji wyłącznego dostępu do jakiegoś zasobu. W tym przypadku służą do ochrony przed zjawiskiem wyścigu oraz do synchronizacji dostępu do danych, w programach wielowątkowych. Wątki reprezentują obiekty typu *std::thread*. Jako parametr konstruktor klasy *std::thread* przyjmuje nazwę funkcji którą ma wykonać. W odtwarzaczu stworzonym w ramach pracy magisterskiej wątki używane są zarówno przez wewnętrzne funkcje biblioteki *libVLC* jak i podczas ładowania kolejnych klatek filmu z dysku do pamięci ram. Wczytywanie odbywa się w konfigurowalnej przez użytkownika liczbie wątków. Aby zapewnić ich synchronizację należy skorzystać z wymienionych wcześniej obiektów klasy *std::mutex.* W tym przypadku wątki współdzielą jeden obiekt tej klasy, kolejno rezerwując sobie wyłączny dostęp do zasobów dzięki użyciu metody *lock*, a następnie zwalniając go korzystając z metody *unlock.*

Rozmiar bufora określany jest na podstawie stałych, definiowanych na początku działania algorytmu. Są to kolejno wysokość i szerokość odtwarzanej klatki wyrażona w pikselach oraz informacja ile bitów reprezentuje każdy piksel. Na potrzeby algorytmu otrzymana liczba bitów

przeliczana jest na bajty, poprzez proste dzielenie przez osiem. Wskaźnik *bufferSize* ustawiany jest na adres pamięci, z określoną wartością rozmiaru klatki. Jednym z założeń odtwarzacza jest, że wszystkie klatki w danym filmie mają ten sam rozmiar. Jeśli jakakolwiek klatka została już wczytana do kolejki, jej adres zostaje przypisany do zmiennej *buffer.* Następnie ustawiane są omówione już zmienne *pts* i *dts*.

```
int MyImemReleaseCallback (void* data, const char* , size_t , void* )
{
    g_mtx.lock();
    FramesHandler* framesHandler = (FramesHandler*)data;
    if (framesHandler->GetFramesCount() > 1)
    {
        framesHandler->ClearFirstFrame();
    }
    if(framesHandler->m_bNothingElseInFile && framesHandler->GetFramesCount() <= 1)
    {
        framesHandler->ClearFrames();
        DisplayHandler::m_bDone = true;
    }
    g_mtx.unlock();
    return 0;
}
```

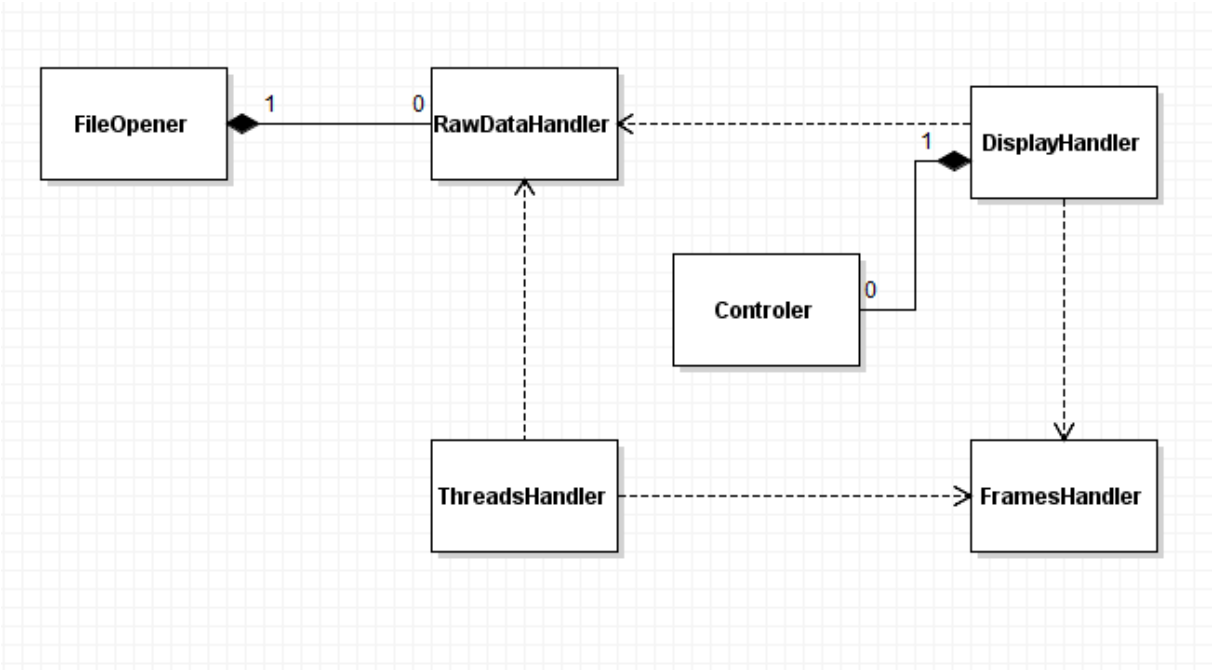
Rysunek 7 Widok ekranu z kodem zwalniającej funkcji przywołań.

Zwalniająca funkcja przywołań przyjmuje ograniczoną ilość parametrów względem funkcji alokującej. W większości nie będą one omawiane, ponieważ ich przeznaczenie jest takie samo w obu typach funkcji przywołań. Podkreślić należy znaczenie parametru *data*. Jeśli funkcja alokująca zaalokowała jakiś blok pamięci, musi zostać on zwolniony w wyżej wymienionej funkcji.

Pierwszym zadaniem funkcji jest zarezerwowanie dostępu do zasobów w omówionym już mechanizmie opierającym się na funkcjonalnościach obiektu klasy *std::mutex*. Kolejnym krokiem jest jawne rzutowanie na typ *FramesHandler** opisane wcześniej w tym rozdziale. Po nim następuje logika odpowiedzialna za usunięcie klatki filmu, która została już wyświetlona. Element usuwany jest z kolejki aby zwolnić miejsce w pamięci na nowo wczytane klatki. Następnie odbywa się sprawdzenie, czy ostatnia wyświetlona klatka nie była ostatnią z sekwencji filmowej. Jeśli tak, ustawiana jest flaga *DisplayHandler::m_bDone*, odpowiedzialna za zakończenie odtwarzania filmu w sposób bezpieczny dla działania całego odtwarzacza.

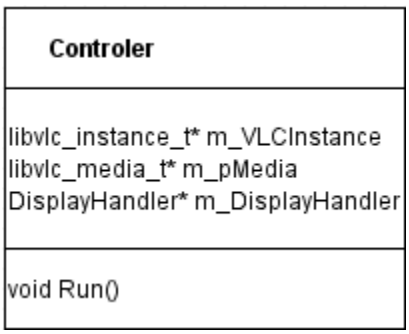
5.3. OPIS KLAS ODTWARZACZA - JAGIELSKI

W poniższym rozdziale opisane zostaną klasy użyte w implementacji odtwarzacza nieskompresowanych sekwencji wideo. Przedstawiony zostanie diagram klas UML (angielski akronim rozwijany: *Unified Modeling Language* - czyli zunifikowany język modelowania) opisujący zależności między poszczególnymi elementami implementacji.



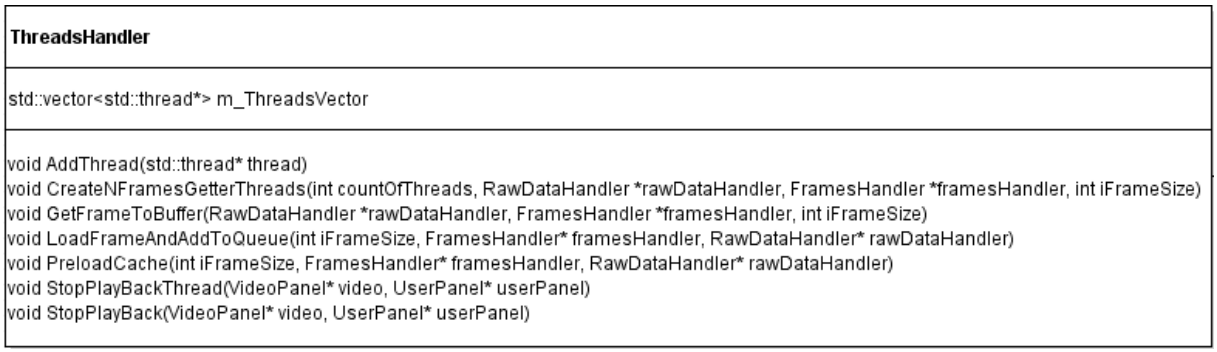
Rysunek 8 Uproszczony diagram klas odtwarzacza UML.

Na rysunku 8 przedstawiono hierarchię klas autorskiego rozwiązania dla odtwarzacza. Na diagramie pokazany jest uproszczony diagram klas. Ze względu na czytelność przedstawiono tylko nazwy klas. Szczegóły dotyczące idei stworzenia danych klas oraz opis ich funkcjonalności zostanie omówione w dalszej części rozdziału. Sercem całego systemu jest klasa *Controler*. Cztery różne typy obiektów obsługujących kolejno wątki, surowe dane, wyświetlanie oraz dane klatki zapewniają pokrycie wszystkich koniecznych funkcjonalności. W tym rozdziale omówione zostaną tylko klasy odtwarzacza. Elementy architektury graficznego interfejsu użytkownika oraz moduł testów zostaną omówione później.



Rysunek 9 Diagram UML klasy Controler.

Controler czyli najważniejsza klasa w odtwarzaczu z punktu widzenia użytkownika. Inicjuje ona w swoich konstruktorach obiekty mediów i instancji dostarczanych przez bibliotekę *libVLC* i przechowuje wskaźniki na otrzymane obiekty w swoich polach (ang. *fields).* Posiada również wskaźnik na obiekt typu *DisplayHandler*, inicjowany w konstruktorach. Jedyną metodą klasy *Controler* jest *Run*, która jako parametry przyjmuje dwie klasy pomocnicze, obsługujące klatki oraz surowe dane wczytywane z pliku. Uruchamia ona proces odtwarzania filmu. Ideą stworzenia tej klasy było uzyskanie narzędzia pozwalającego na ukrycie implementacji przed użytkownikiem oraz prosty sposób uruchomienia programu. Takie podejście pozwoliło na bardziej efektywny podział prac w zespole programistycznym oraz na zrównoleglenie rozwoju poszczególnych części funkcjonalności. Zaznaczyć trzeba, że jest to klasa kontrolująca wyświetlanie, a więc wczytywanie klatek do pamięci musi zostać wykonane przez inny element programu.



Rysunek 10 Diagram UML klasy ThreadsHandler.

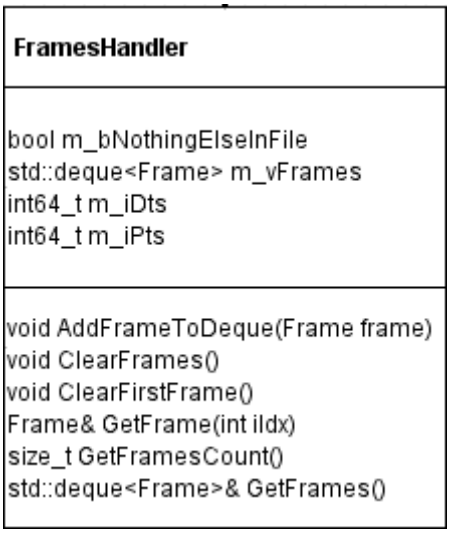
ThreadsHandler to klasa zarządzające wątkami. Jedynym jej polem jest wektor wątków, do którego dodawane są wszystkie nowo stworzone, a usuwane te już nie aktywne. Na rysunku 10 przedstawiono diagram UML tej klasy. Pierwszą z metod tej klasy jest funkcja *AddThread* dodająca nowy wątek do wektora. Powinna być wywoływana zawsze, gdy program utworzy nowy wątek, który powinien być obsługiwany przez obiekt klasy *ThreadsHandler.* Metoda *CreateNFramesGetterThreads* tworzy ilość wątków konfigurowalną przez jeden z jej parametrów. Wszystkie stworzone w metodzie wątki korzystają z innej metody - *GetFrameToBuffer*. Podlega ona zasadom przydziału zasobów za pomocą omówionych już elementów *std::mutex.* Takie połączenie zapewnia określoną, konfigurowalną liczbę klatek które znajdują się w kolejce odtwarzacza. Jeśli jakakolwiek z nich zostanie wyświetlona jeden z wątków roboczych wczyta nową z pliku, a następnie doda ją do kolejki.

Istnieje również opcja wczytania całego filmu do pamięci RAM, korzystając z metody *PreloadCache.* Podejście to ma swoje wady i zalety. Główną wadą jest konieczność posiadanie dużej ilości wolnej pamięci RAM. Nieskompresowane dane sekwencji filmowej zajmują bardzo dużo miejsca. Jedną z zalet jest możliwość ukrycia ograniczeń sprzętowych takich jak wolny odczyt z dysku. Odtwarzacz wczytuje

całą sekwencję, jednocześnie jej nie wyświetlając. Uruchomienie filmu rozpoczyna się dopiero wtedy, gdy wszystkie dane zostaną przeniesione do RAM-u. Eliminuje to ewentualne zacięcia filmu podczas jego wyświetlania, zapewnia, że wszystkie klatki zostaną wczytane oraz eliminuje zapobiega zjawiskom wyścigu. Największym argumentem za wczytywaniem filmu podczas jego odtwarzania jest możliwość odtwarzania w taki sposób sekwencji zajmujących więcej miejsca niż wynosi pojemność RAM-u. Z drugiej strony ograniczona przepustowość wczytywania z dysku nie pozwala na uruchamianie filmów wysokiej jakości w opisywany sposób.

Prawdopodobnie najlepszym rozwiązaniem byłoby połączenie obu rozwiązań w jedno. Część filmu mogłaby być ładowana jeszcze przed jego uruchomieniem, a następnie w trakcie wyświetlania wczytywane były by klatki z jego dalszej części. Należałoby stworzyć również algorytm precyzujący największą długość sekwencji przy zadanej jakości. W innym przypadku mogłyby pojawić się problemy opisane w ostatnim akapicie. Mimo wszystko dla wysokich jakości nieskompresowanych wideo najmniej wydajnym ogniwem jest sprzęt elektroniczny. Wraz z jego ulepszeniem możliwe jest osiągnięcie lepszych odtwarzanych jakości.

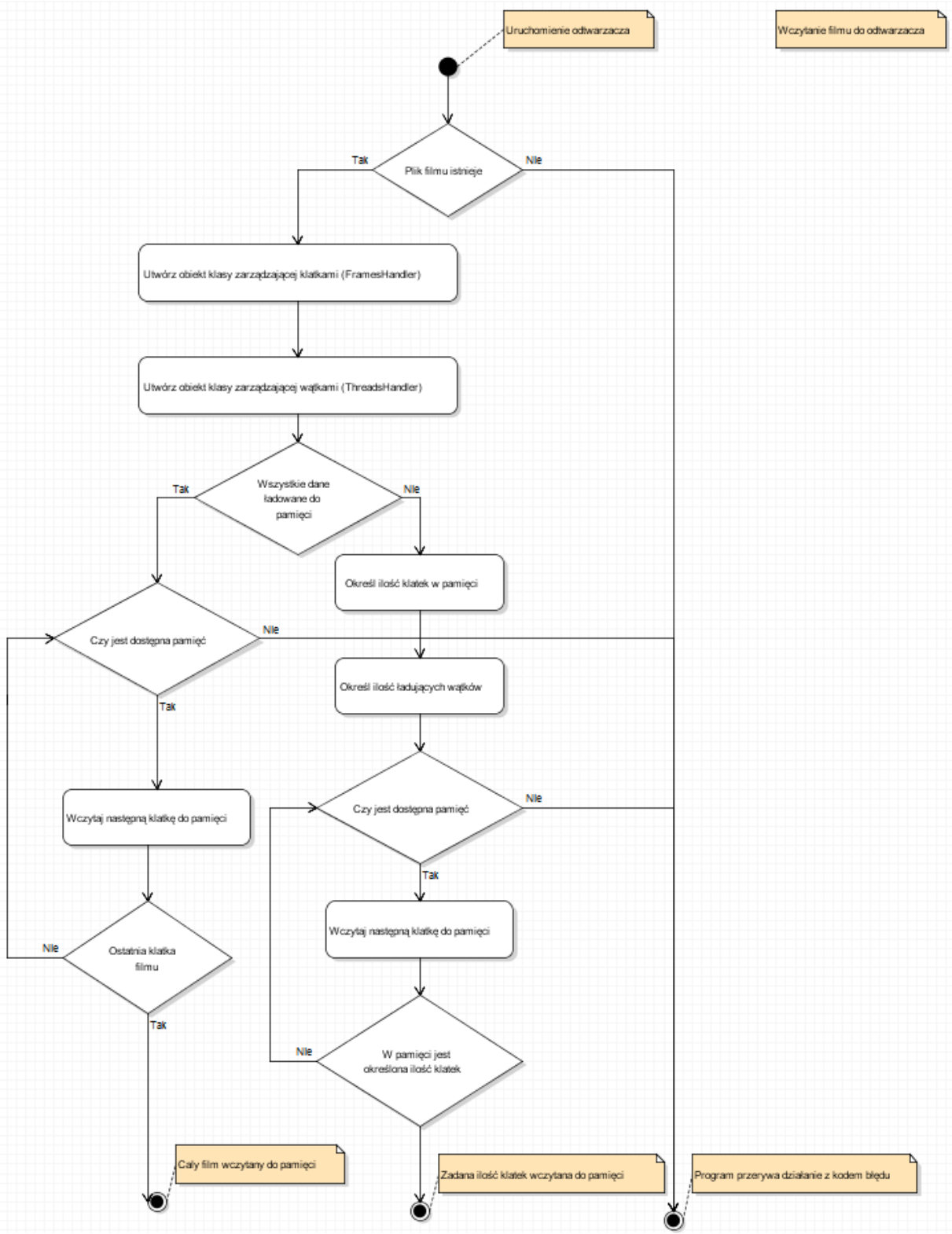
Metody *StopPlayBack* oraz *StopPlayBackThread* mają za zadanie zamknąć okna odtwarzacza po skończeniu wyświetlania filmu.



Rysunek 11 Diagram UML klasy FramesHandler.

FramesHandler to klasa stworzona z myślą o łatwym zarządzaniu zbiorem klatek wczytanych do pamięci podręcznej. Posiada kolejkę, przechowującą wszystkie klatki w odtwarzaczu. Zapewnia również zestaw operacji na kolejce, pozwalających wydobyć z niej odpowiednią klatkę, zbiór klatek lub określić ich ilość. Zdecydowano się użyć kolejki typu *std::deque* czyli takiej, która gwarantuje dostęp do pierwszego i ostatniego elementu. Należy zwrócić uwagę na metody odpowiedzialne za usuwanie klatek z kolejki, czyli *ClearFirstFrame* oraz *ClearFrames*. Są one ważnym elementem algorytmu, pozwalają zwalniać zasoby, aby nowe klatki filmu mogły zostać wczytane. Obiekty tego typu pojawiają

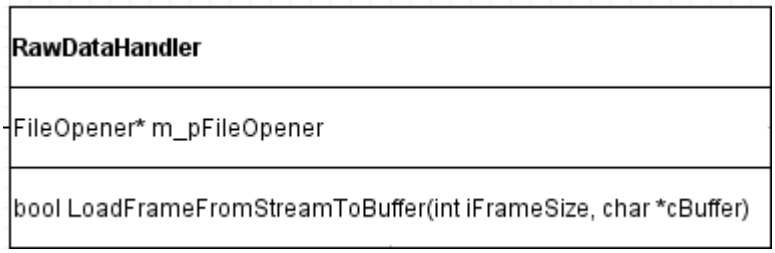
się w bardzo wielu miejscach kodu, dlatego dalsze zwiększanie funkcjonalności zawartych w tej klasie powinna być dobrze przemyślana. Wprowadzenie nadmiarowych, niepotrzebnych w klasie *FramesHandler* pól czy metod może prowadzić do nadmiernego skomplikowania kodu.



Rysunek 12 Diagram aktywności UML przedstawiający proces wczytywania filmu do odtwarzacza.

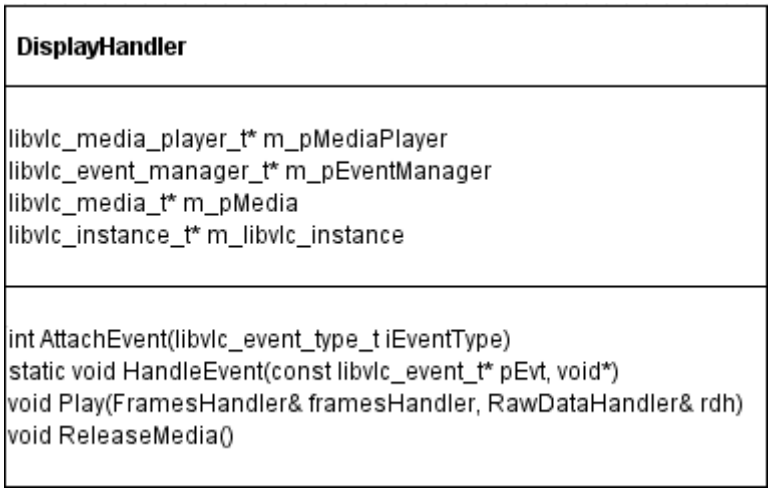
Aby lepiej zrozumieć algorytm ładowania klatek do odtwarzacza przygotowano diagram aktywności UML. Algorytm przewiduje trzy możliwe zakończenia. Pierwszym z nich jest przerwanie działania programu w wyniku jakiejś nieprawidłowości. Przykładowo może to być brak miejsca w pamięci RAM lub nieistniejący plik z danymi filmu.

Pozostałe dwa zależą od wybranego trybu ładowania klatek filmu. Przedstawiony na rysunku 12 diagram opisuje aktywności do momentu, w którym rozpoczyna się odtwarzanie filmu. Dlatego wynikami jego działania może być załadowanie całego filmu, lub tylko jego części. W przypadku gdy ładowany jest cały, podczas jego odtwarzania nie następuje już żaden odczyt filmu z dysku. Odwrotnie dzieje się gdy wybrana zostanie alternatywa. W momencie wczytania zadanej ilości klatek do kolejki następuje uruchomienie odtwarzania filmu. Zdefiniowana ilość wątków roboczych odpowiada za utrzymanie odpowiedniej ilości klatek w kolejce.



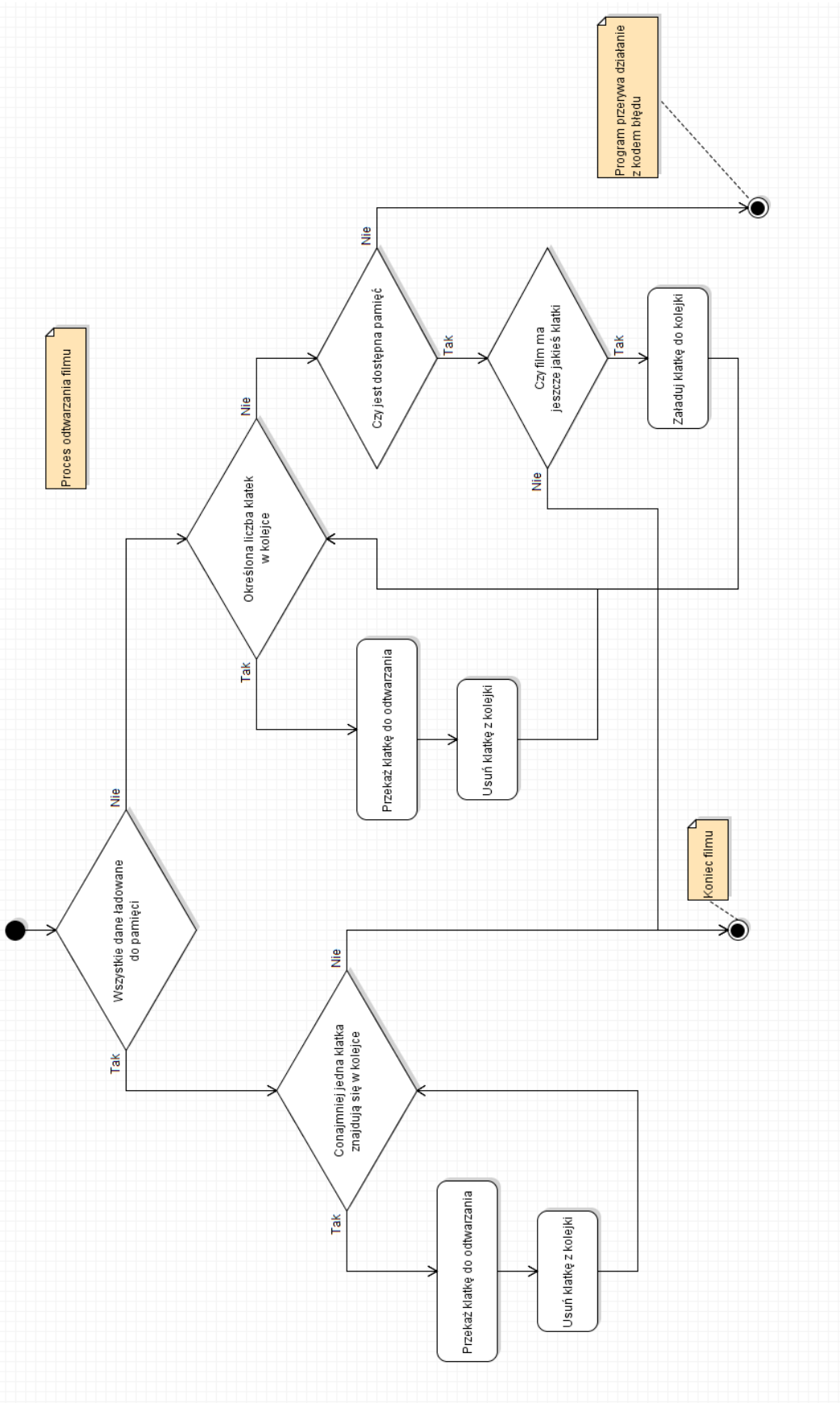
Rysunek 13 Diagram UML klasy RawDataHandler.

RawDataHandler to klasa, której zadaniem jest zarządzanie strumieniem danych z otwartego pliku. Jej diagram UML został przedstawiony na rysunku 13. Posiada informacje o wielkości klatki, dlatego jest w stanie odczytywać odpowiednie porcje danych i przekazywać je dalej. Posiada również wskaźnik na obiekt typu *FileOpener.* Odpowiedzialny jest on za sprawdzenie, czy plik istnieje, a następnie jego otwarcie. Przyjmuje tylko jeden parametr - ścieżkę do żądanego pliku. Głównym użytym narzędziem w omawianych dwóch klasach jest *std::fstream.* Jest to klasa dostępna w ramach biblioteki standardowej pozwalająca na czytanie oraz pisanie z/do pliku.



Rysunek 14 Diagram UML klasy DisplayHandler.

Ostatnią omawianą klasą jest *DisplayHandler.* Jej diagram UML przedstawiony jest na rysunku 14. Podobieństwo do omówionej już klasy *Controler* jest nie przypadkowe. To właśnie klasę *Controler* przysłania. *DisplayHandler* oprócz znanych już pól posiada wskaźnik na obiekt typu *libvlc_event_manager_t*. Jest to klasa zarządzająca zdarzeniami, które zostaną omówione w następnym rozdziale. Głowną metodą jest *Play,* odpowiedzialna za uruchomienie odtwarzania filmu.



Rysunek 15 Diagram aktywności UML przedstawiający proces odtwarzania filmu.

Diagram na rysunku 15 prezentuje proces odtwarzania filmu. Niezależnie od podjętych decyzji bazuje na operacjach wykonywanych wewnątrz omówionych już funkcji wywołań. Widoczny jest podział na dwie główne ścieżki. Decyzja zależy od tego, czy wybrany jest tryb wczytywania całego filmu do pamięci, czy jego ładowania podczas odtwarzania filmu. Jeśli coś pójdzie nie tak podczas wykonywania programu, zakończy się on z odpowiednim kodem błędu. Na diagramie przedstawiono taką sytuację, gdy brakuje pamięci RAM, aby wczytać do niego nowe dane. Pojawia się ona tylko w jednej ścieżce, ponieważ podczas wczytywania całego filmu do pamięci taka sytuacja eliminowana jest już wcześniej (Rys 12.).

5.4. System zdarzeń - Jagielski

Biblioteka *libVLC* dostarcza również obsługę zestawu zdarzeń (ang. *events*) mogących mieć miejsce podczas odtwarzania filmu. Zdarzenia obsługiwane są asynchronicznie i do zarządzania nimi potrzebny jest obiekt klasy *libvlc_event_manager_t*, czyli menadżer zdarzeń. W odtwarzaczu wskaźnik na obiekt tego typu posiadają obiekty klasy *DisplayHandler*.

```
int DisplayHandler::AttachEvent(libvlc_event_type_t iEventType)
{
    return libvlc_event_attach(m_pEventManager, iEventType, HandleEvent, nullptr);
}

void DisplayHandler::HandleEvent(const libvlc_event_t* pEvt, void*)
{
    switch(pEvt->type)
    {
        case libvlc_MediaPlayerEndReached:
            std::cout << "MediaPlayerEndReached" << std::endl;
            m_bDone = true;
            break;
        default:
            std::cout << libvlc_event_type_name(pEvt->type) << std::endl;
    }
}
```

Rysunek 16 Widok ekranu z przykładowym kodem obsługującym zdarzenie.

Pierwszym krokiem obsługi zdarzenia jest użycie funkcji *libvlc_event_attach*, odpowiadającej za dodanie wybranego zdarzenia do menadżera zdarzeń. Funkcja ta przyjmuje również jako parametr nazwę funkcji lub metody odpowiedzialnej za obsługę tego zdarzenia. W przypadku oprogramowania odtwarzacza jest to *DisplayHandler::HandleEvent* widoczna na rysunku 16.

Zdarzenia reprezentowane są w bibliotece *libVLC* jako typ wyliczeniowy (ang. *enum*). Ich dokładna lista dostępna jest w dokumentacji biblioteki. W tym rozdziale zostaną omówione tylko wybrane z nich. Na rysunku 16 przedstawiony jest fragment kodu w którym wyszczególniony jest tylko jeden typ zdarzenia - *libvlc_MediaPlayerEndReached*. Zdarzenie to uaktywniane jest tylko w przypadku, gdy odtwarzany film się zakończy. Jeśli takie zdarzenie będzie miało miejsce, flaga *m_bDone* zostanie ustawiona i rozpocznie się bezpieczne zamykanie odtwarzacza.

Każde inne zdarzenie zostanie wypisane na standardowym wyjściu jako ciąg znaków. Proste logowanie, osiągnięte w ten sposób pozwoli na sprawniejsze znalezienie problemu w razie, gdyby ten wystąpił.

[https://www.videolan.org/developers/vlc/doc/doxygen/html/group__libvlc__event.html#ga284c010ecde8abca7d3f262392f62fc6]

5.5. Dane wejściowe systemu - Jagielski

Na potrzeby testów wydajności i niezawodności odtwarzacza stworzono prosty skrypt zdolny do generacji filmów w zadanych jakościach. Został użyty również w części badawczej, dostarczając nieskompresowanych sekwencji wideo. Jego zadaniem było w prosty sposób, korzystając z filmu wzorcowego wygenerować filmy o innych niż wzorcowy jakościach. Oczywistym jest fakt, że jakość filmu wzorcowego musi być lepsza niż najlepszy z oczekiwanych plików wynikowych.

```
def main():
    bit_rates = ['64k', '32k', '16k', '8k']

    for bit_rate in bit_rates:
        cut_bit_rate(bit_rate)


def cut_bit_rate(bit_rate):
    if not os.path.exists(output_folder):
        os.makedirs(output_folder)

    path_with_postfix = output_folder+'/'+file_name+'_'+str(bit_rate)
    bit_rate_cut_command = 'ffmpeg -ss '+time_start + \
                           ' -i '+input_file_location + \
                           ' -t '+time_end + \
                           ' -vcodec h264 ' + \
                           '-strict -2 ' + \
                           '-b:v '+str(bit_rate)+' ' + \
                           path_with_postfix+'.mp4'

    subprocess.check_output([bit_rate_cut_command], shell=True)

    convert_to_raw_command = 'ffmpeg ' + \
                             '-i '+path_with_postfix+'.mp4 ' + \
                             '-f rawvideo ' + \
                             '-vcodec rawvideo ' + \
                             path_with_postfix+'.raw'

    subprocess.check_output([convert_to_raw_command], shell=True)

    os.remove(path_with_postfix+'.mp4')


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

Rysunek 17 Widok ekranu z fragmentem kodu źródłowego generatora filmów o zdanych jakościach.

Możliwe jest jednak stworzenie filmu o parametrach lepszych niż wzorcowy. Otrzymana w ten sposób sekwencja nie będzie różnić się wizualnie od oryginału, pod warunkiem, że sprzęt jest na tyle wydajny żeby obsłużyć obie z nich. Przeznaczeniem takich sekwencji może być zbadanie zachowania systemu przy odtwarzaniu filmów o lepszej jakości, gdy przykładowo te lepsze nie są dostępne.

Skrypt przedstawiony na rysunku 17 napisany został w języku *Python*. Opiera się on na programie *ffmpeg* i jego możliwościach konwersji filmów. Filmy w formacie *mp4* zapisywane są ponownie do tego samego formatu, jednak limitowana jest ich przepływność bitowa. Następnie następuje konwersja do nieskompresowanej sekwencji wideo. Należy podkreślić, że skrypt używany jest tylko na potrzeby testów i nie powinien stanowić dla użytkownika końcowego źródła filmów do

badań. Nieskompresowane sekwencje wideo z zagwarantowaną jakością powinny być dostarczane z zaufanych źródeł.

Kolejne wywołania wspomnianego programu jako podproces z różnymi parametrami jako wynik tworzy serie filmów o zadanych wartościach. *FFmpeg* pozwala ustalić maksymalną przepływność bitową filmu wynikowego. Aby tego dokonać, potrzeba przekazać jako parametr „-b:v" a następnie wartość limitującą. Program pozwala także na wybór fragmentu filmu wzorcowego, który ma zostać skonwertowany. Początek pożądanego okresu ustawiany jest za pomocą parametru „-ss", a jego koniec „-t". Parametr „-f" odpowiedzialny jest za wymuszenie formatu na wyjściu, w przykładzie przedstawionym na rysunku 17 jest to *rawvideo*, czyli nieskompresowana sekwencja wideo. Z kolei „-vcodec" ustawiaja kodek wideo. Nieskompresowana sekwencja wideo wymaga podania *rawvideo* jako wspomniany parametr.

Aby odtwarzacz poprawnie interpretował dane które otrzymuje konieczne jest zdefiniowanie sposób w jaki reprezentowane są dane piksele. Na każdy z nich może przypadać różna ilość bitów opisujących luminancję lub głębie barw. Różnice mogą wynikać z kolejności ułożenia bitów na przykład *big/little endian*. Innym źródłem różnic jest podpróbkowanie chrominancji, omówione we wstępie teoretycznym. Aby otrzymać film wynikowy, w którym wszystkie piksele reprezentowane są w żądany sposób należy użyć parametru „-pix_fmt", a następnie podać żądany format. Opis dostępnych reprezentacji został zamieszczony w dokumentacji programu *FFmpeg*.

[https://ffmpeg.org/doxygen/2.7/pixfmt_8h.html#a9a8e335cf3be472042bc9f0cf80cd4c5]

Sterowanie reprezentacją piksela w odtwarzaczu odbywa się za pomocą opisanego wcześniej interfejsu *imem*. Za pomocą parametrów „--imem-width", „--imem-height" oraz „--imem-channels" ustawiane są kolejno szerokość, wysokość oraz głębia kolorów odtwarzanego filmu.

```
/* Video codec */
#define VLC_CODEC_MPGV          VLC_FOURCC('m','p','g','v')
#define VLC_CODEC_MP4V          VLC_FOURCC('m','p','4','v')
#define VLC_CODEC_DIV1          VLC_FOURCC('D','I','V','1')
#define VLC_CODEC_DIV2          VLC_FOURCC('D','I','V','2')
#define VLC_CODEC_DIV3          VLC_FOURCC('D','I','V','3')
#define VLC_CODEC_SVQ1          VLC_FOURCC('S','V','Q','1')
#define VLC_CODEC_SVQ3          VLC_FOURCC('S','V','Q','3')
#define VLC_CODEC_H264          VLC_FOURCC('h','2','6','4')
#define VLC_CODEC_H263          VLC_FOURCC('h','2','6','3')
```

Rysunek 18 Widok ekranu z fragmentem pliku vlc_fourcc.h z biblioteki libVLC.

Na rysunku 18 przedstawiono sposób w jaki definiowane są kolejne kodeki wideo w bibliotece *libVLC.* W ten sam sposób definiowane są reprezentacje pikseli w zależności od podpóbkowania chrominancji i głębi kolorów. Interfejs *imem* przyjmuje parametr „--imem-codec", który

odpowiedzialny jest za ustawienie kodeka w przypadku sekwencji skompresowanych lub sposobu zapisu piksela w sekwencjach nieskompresowanych. Lista wspieranych kodeków i reprezentacji dostępna jest w kodzie źródłowym biblioteki *libVLC*.

[https://github.com/RSATom/libvlc-sdk/blob/master/include/vlc/plugins/vlc_fourcc.h]

[https://ffmpeg.org/ffmpeg.html]

[https://wiki.videolan.org/YUV/]

6. KONFIGURACJA - ORLIŃSKI

Oprogramowanie do przeprowadzania testów subiektywnych musi dysponować możliwością wprowadzania bazy filmów, które należy odtworzyć. Konieczny jest również wybór scenariusza testowego, czy określenie podstawowych cech każdego z filmów do odtworzenia np. rozdzielczości. Jest to konieczne ze względu na potrzebę zdefiniowania wielkości wczytywanej ramki. Nieskompresowane wideo jest wczytywane jako strumień bitów. Program musi rozróżniać kolejne klatki do wyświetlenia jako bloki o zadanej wielkości. Administrator powinien mieć także wpływ na kolejność odtwarzanych filmów oraz np. ich nazwy (szczególnie przy menu wyboru).

W odpowiedzi na konieczność dostarczenia konfiguracji zdecydowano się umożliwić administratorowi testów tworzenie plików tekstowych zawierających bloki o charakterystycznej składni reprezentujących każdy kolejny film.

```
<video pieski
<path /home/barti/CLionProjects/UHDPlayer/sampleVideos/puppies.raw
<width 4096
<height 2304
<fps 40000
<depth 12
<codec I420
```

Rysunek 19 Blok konfiguracji odpowadający jednemu filmowi.

Powyższy fragment konfiguracji przedstawia blok reprezentujący film z pliku „puppies.raw" wczytywany z folderu o ścieżce podanej w linii rozpoczynającej się od słowa *path*. Kolejne dwie linie są wczytywane jako szerokość i wysokość obrazu. Następnie podano odstęp czasowy pomiędzy kolejnymi klatkami (w mikrosekundach) oraz ilość bitów reprezentujących piksel lub kodek.

Wczytywanie konfiguracji odbywa się z panelu administracyjnego jako wczytanie pliku tekstowego i utworzenie obiektu klasy *PlayerConfigurationsHandler* przy pomocy konstruktora parametrycznego przyjmującego jako argument ścieżkę do pliku z konfiguracją. Zawartość pliku jest następnie przesyłana w postaci dwóch zmiennych typu string reprezentujących nazwę parametru i jego wartość. Nazwy są porównywane z wzorcami i jeżeli w pobranym ciągu znaków zostaje odnaleziony wzorzec wartość zostaje zapisana do odpowiedniego elementu. Klasa *PlayerConfigurationsHandler* posiada 3 pola. Pierwsze z nich to pole zawierające numer aktywnej konfiguracji, kolejnym jest pole typu logicznego oznaczające poprawnie wczytaną konfigurację. Jest ono sprawdzane przy pomocy metody *CheckConfiguration* z tej samej klasy. Najważniejszym polem klasy jest jednak wektor wskaźników na obiekty typu *MovieProperties* zawierające ustawienia charakterystyczne dla każdego z filmów, ustawiane na podstawie wczytanej konfiguracji.

Każdy kolejny blok zawierający film (rozpoczynający się od „<video”) jest oznaką, że konstruktor klasy *PlayerConfigurationsHandler* powinien utworzyć kolejny obiekt typu *MovieProperties* i dodać wskaźnik do niego do wspomnianego wektora, zapisywany jest również (w zmiennej lokalnej) numer aktualnie utworzonego obiektu wektora. Klasa *MovieProperties* posiada takie parametry filmu jak ścieżka w której film się znajduje, nadana mu w konfiguracji nazwa, szerokość i wysokość wyrażona w pikselach, odstęp czasowy miedzy klatkami, ilość bitów na piksel lub kodek czy wystawiona przez testera ocena domyślnie ustawiana na zero. Parametry tekstowe są bezpośrednio wpisywane z pliku konfiguracyjnego, natomiast liczbowe zostały uprzednio skonwertowane przy pomocy funkcji *std::stoi*. Klasa *PlayerConfigurationsHandler* posiada szereg metod zwracających każdy z parametrów dla filmu o danym numerze z wektora wskaźników na obiekty typu *MovieProperties*. Są one wykorzystywane w funkcjonalności programu gdy zostają wczytane kolejne filmy.

Przechowywanie ustawień i ścieżek jest konieczne, ponieważ sam film w formacie RAW nie przechowuje żadnych danych dotyczących rozdzielczości czy głębokości barw. Nie jest również możliwe przechowywanie w pamięci wszystkich sekwencji wideo potrzebnych do wykonania testu ze względu na ogromną ilość przestrzeni dyskowej jaką sekwencje zajmują. Stworzona w ten sposób konfiguracja jest odpowiedzią na te problemy, a także kompromisem pomiędzy łatwością implementacji i obsługi oprogramowania.

7. Graficzny interfejs użytkownika - Orliński

7.1. Wstęp - Orliński

Przeprowadzanie jakichkolwiek testów z udziałem losowych osób wymaga, aby testerzy byli prowadzeni przez cały test niejako za rękę. Wiążę się to z przemyśleniem i zaprojektowaniem interfejsu użytkownika w taki sposób, aby zapewnić możliwie jak największą czytelność i ergonomie procesu testowania. Interfejs użytkownika musi pozwalać na oglądanie filmów i wystawianie ocen. Do interfejsu można wprowadzić dodatkowe panele pozwalające na informowanie testera, czy też pobieranie od niego dodatkowych informacji.

7.2. Konfiguracja środowiska - Orliński

Przed przystąpieniem do pracy z bibliotekami *QT* należy skonfigurować środowisko programistyczne. Ponieważ projekt został stworzony w środowisku *CLion* opartym na *cmake* należało zmodyfikować plik z dyrektywami, dodając do linkera ścieżki do uprzednio zainstalowanej biblioteki *QT*. Dla poprawnego działania narzędzi biblioteki, a także do automatycznej generacji koniecznych do kompilacji plików *.ui oraz ui_*.h należy w skrypcie *CMakeLists* uruchomić następujące parametry:

```
9
10  set(CMAKE_AUTOMOC ON)
11  set(CMAKE_AUTOUIC ON)
12  set(CMAKE_INCLUDE_CURRENT_DIR ON)
13
```

Rysunek 20 Parametry cmake konieczne dla QT

7.3. Podstawowe połączenie VLC z QT - Orliński

Odtwarzacz wideo *libvlc_media_player* posiada domyślny interfejs (w zależności od ustawień może być oparty o QT) jednakże interfejsu tego nie można w żaden sposób modyfikować. Dlatego też do modyfikowania interfejsu odtwarzacza należy stworzyć własny widżet (ang. *widget*) wideo. Aby to uczynić w najbardziej prosty sposób należy utworzyć klasę dziedziczącą z *QWidget*, a następnie utworzyć w niej obiekt typu *QFrame*, który następnie należy połączyć z odtwarzaczem za pomocą metody *libvlc_media_player_set_xwindow*, która za argumenty przyjmuje utworzoną wcześniej instancje *libvlc_media_player*, a także identyfikator okna, który uzyskujemy wołając metodę *winId* na obiekcie okna. Następnie można uruchamiać i kończyć wideo tak samo jak robiono to dotychczas bez dodatkowego interfejsu, pamiętając o tym, że nie wolno zniszczyć obiektu okna, do którego przypisany jest odtwarzacz wideo. Standardowy obiekt *QFrame* możemy modyfikować nadając mu zależne od nas wymiary, czy też umieszczając go w layoucie w wybranym przez siebie miejscu otaczając dowolnymi obiektami dodając np. paski do sterowania głośnością. Kolejnym krokiem powinno być wywołanie w głównej funkcji *main* programu utworzenie obiektu klasy *QApplication*, którego konstruktor przyjmuje

standardowe parametry programu. *QApplication* to klasa służąca do zarządzania aplikacją okienkową, a także podstawowymi opcjami takimi jak np. rozmiary okna podstawowego czy ustawienie trybu pełnoekranowego. Ważną rolą jest także możliwość przenoszenia argumentów i sygnałów sterujących do dalszych paneli programu. Następnym krokiem jest powołanie do życia obiektu klasy bazowej za pomocą jej konstruktora. Standardowy konstruktor klasy dziedziczącej z *QWidget* nie posiada dodatkowych parametrów, jednakże warto go przeciążyć i przekazać mu argumenty programu, aby następnie przekazać je do konstruktora instancji *VLC*, jeżeli to konieczne. Stworzony widżet należy wyświetlić metodą *show*. Ze względu na konieczność otwierania i zamykanie widżetu wideo, a także stworzenia paneli do oceniania i prowadzenia testera zastosowano bardziej zaawansowane podejście.

7.4. Zaproponowany Interfejs - Orliński

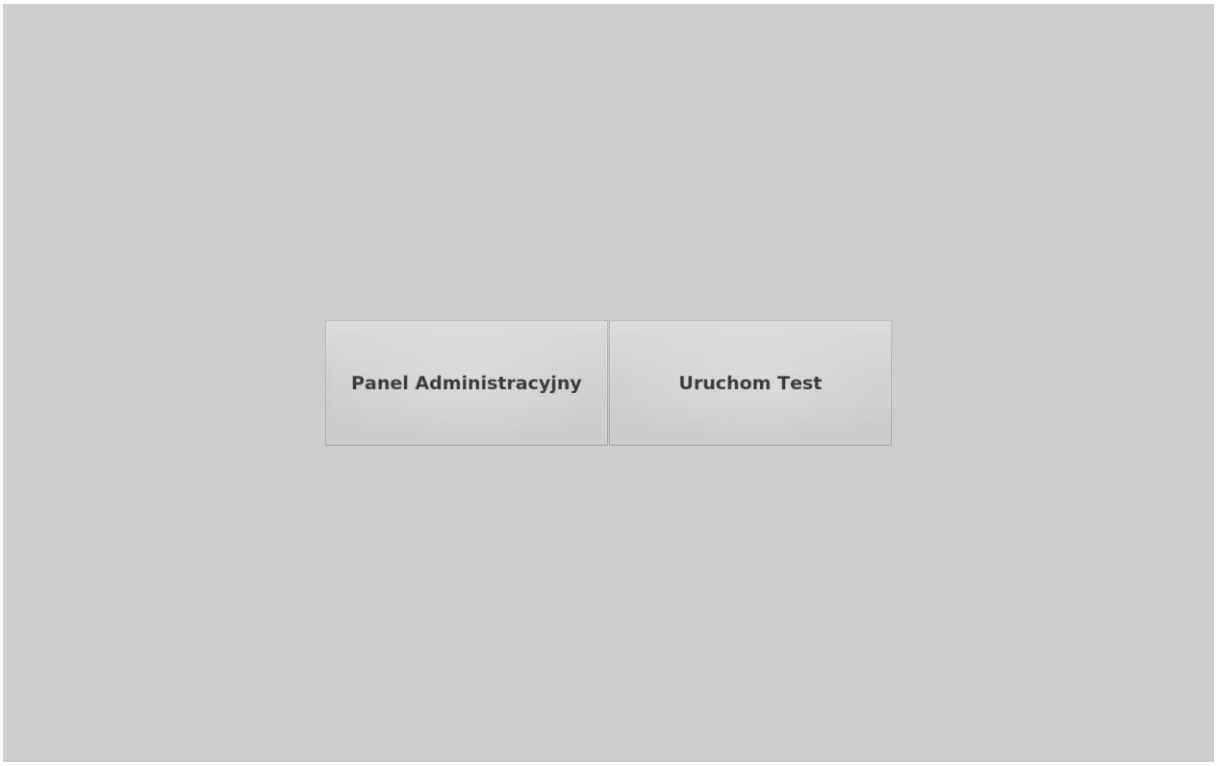
7.4.1. Wstęp - Orliński

Ze względu na silny związek interfejsu użytkownika ze scenariuszami testowymi (końcowa wersja powinna korespondować z opracowanymi scenariuszami pozwalając na ich przeprowadzenie) postanowiono zaprojektować interfejsy dedykowane do konkretnych scenariuszy dopiero po ich określeniu. Ponieważ zdecydowano się na standardowe metody *ACR, PC* oraz menu z wyborem filmów postanowiono opracować *GUI* oparte o modułowe panele podmieniane automatycznie w zależności od wybranej konfiguracji testów.

7.4.2. Stworzony Interfejs - Orliński

Interfejs użytkownika wykonano za pomocą środowiska *QT Creator* w narzędziu *QT Designer* pozwalającym na wstawianie komponentów GUI w trybie okienkowym. Ułatwiło to edycje i umiejscowienie elementów dokładnie w tych miejscach w których zamierzano. Całość została ponownie zaimportowana do środowiska *CLion* głównie ze względu na wygodę i przyzwyczajenie do tego środowiska, dającego większe możliwości debugowania, zwłaszcza przy zaimportowanej bibliotece *libvlc*.

Projektując interfejs kierowano się jego funkcjonalnością. Ze względu na konieczność użytkowania aplikacji zarówno przez administratora testów jak i przez testera należało stworzyć rozdzielne panele testerski i administracyjny. Użytkownik staje przed wyborem czy zamierza konfigurować testy czy też jest testerem i chce je uruchomić.

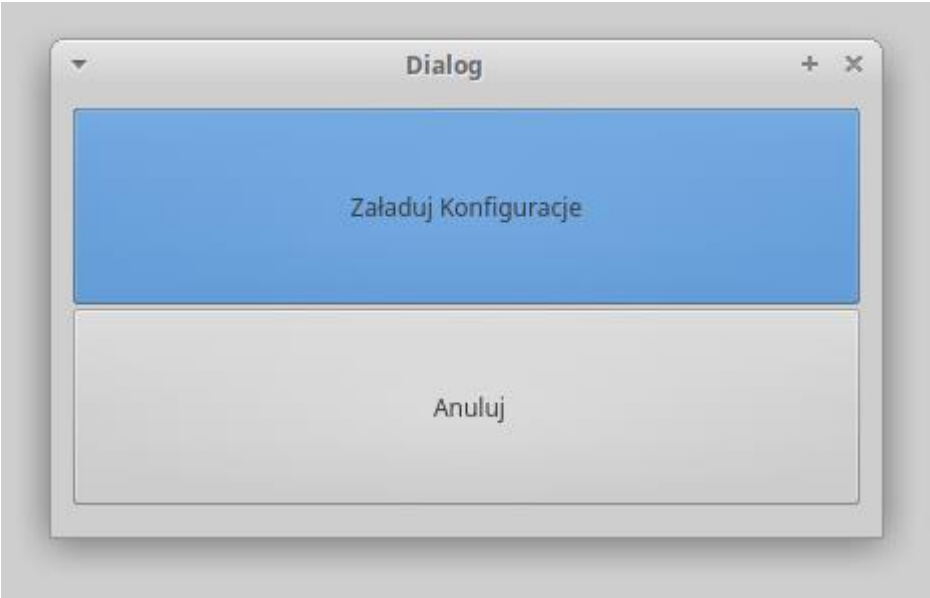


Rysunek 21 Widok panelu MainWindow.

Panel, który widzimy na powyższym widoku ekranu jest obiektem typu *MainWindow*. *MainWindow* to klasa stworzona jako główne okno aplikacji, klasa dziedziczy z *QMainWindow*, klasy QT wersji 5 udostępniającej szereg metod to sterowania aplikacją, dziedziczącą z klasy QWid*get*. Ponieważ obiekt *mainWindow* jest tworzony w głównej funkcji programu, w konstruktorze dodano parametry wysokości i szerokości okna pobrane uprzednio z ustawień ekranu systemowego. Pobieranie rozdzielczości ekranu odbywa się za pomocą dostarczanych w bibliotece QT klas *QScreen* i *QRect.* Pierwsza z nich to typ wskaźnikowy, który za pomocą metody *primaryScreen* z klasy *QApplicatio*n, zostaje ustawiony na obiekt reprezentujący podstawowy ekran w systemie. Następnie za pomocą metody *availableGeometry* zostają pobrane jego wysokość i szerokość, następnie przekazane do *mainWindow*. Konstruktor *MainWindow* wymusza maksymalizacje okna na ekranie za pomocą metody *showFullScreen,* a także tworzy przy pomocy słowa kluczowego *new* pusty obiekt typu *PlayerConfigurationsHandler,* który został omówiony przy okazji obsługi konfiguracji. Na zrzucie ekranu widoczne są dwa przyciski czyli obiekty typu *QPushButton*. Zostały one umieszczone na środku ekranu przy pomocy linii zakotwiczających przyciski w szablonie, co pozwoliło utrzymać je na swoim miejscu dla różnych rozdzielczości.

Przyciski typu *QPushButton* to standardowe przyciski z biblioteki *QT*, poza szerokimi możliwościami związanymi z geometrią, czyli położeniem i wymiarami dostarczają szereg metod pozwalających na interakcje z użytkownikiem. Jedną z tych metod jest metoda *on_pushButton_clicked,* która jest wyzwalana w momencie naciśnięcia na aktywny obszar przycisku. W przypadku *MainWindow* zaimplementowano dwie takie metody (do każdego z przycisków osobną). Pierwsza z nich uruchamia panel administracyjny odbywa się to w następujący sposób. Tworzony jest nowy obiekt typu *AdminPanel,* następnie na ty obiekcie wołane są kolejno metody *show, activateWindow* oraz *topLevelWidget,* pozwala to na wyciągnięcie powstałego panelu administracyjnego na wierzch oraz nadanie mu aktywności. Konstruktor klasy *AdminPanel* przyjmuje jako parametr wskaźnik na

wspomniany obiekt typu *PlayerConfigurationsHandler*, celem nadpisania pustego obiektu, obiektem przechowującym aktualnie wczytaną konfigurację.



Rysunek 22 Panel administracyjny.

Panel administracyjny widoczny na powyższym zrzucie ekranu posiada dwa przyciski z których jeden służy do zamykania go (poprzez zniszczenie obiektu za pomocą *delete*), a drugi uruchamia okno systemowe okno dialogowe za pomocą którego można wybrać plik konfiguracyjny. Następuje to za pomocą metody *getOpenFileName* która zwraca ścieżkę do wybranego filmu w formacie *QString. QString* ze względu na brak konieczności użycia jest konwertowany do *std::string* metodą *toStdString.* Następnie ścieżka do pliku konfiguracyjnego jest przekazywana jako parametr w konstruktorze tworzonego obiektu *PlayerConfigurationsHandler*, konfiguracja zostaje wczytana, przekazany z głównego okna wskaźnik zostaje ustawiony na nowo powstały obiekt, a panel administracyjny zostaje zamknięty.

Drugi z przycisków *MainWindow* jest aktywny tylko w momencie, gdy wczytana jest konfiguracja. Sprawdzanie odbywa się za pomocą wywołania metody *CheckConfiguration* z obiektu na który wskazuje *playerConfigurationsHandler*. Metoda została opisana w podrozdziale dotyczącym konfiguracji. Naciśnięcie na aktywny przycisk tworzy i uruchamia kolejny panel, ustawiając jego geometrie na taką samą jaka została wpisana w *MainWindow*, skalując go do całości ekranu. Za pomocą szeregu metod okno staje się aktywnym i pełnoekranowym. Nowo powstałe okno jest obiektem typu *UserPanel*.

UserPanel przyjmuje w konstruktorze parametry dotyczące wielkości ekranu (przekazywane z *MainWindow*) oraz wskaźnik na obiekt typu *PlayerConfigurationsHandler* przechowujący aktualną konfigurację. Klasa UserPan*el* poza przekazanymi parametrami posiada również kilka istotnych pól. Jednym z nich jest wskaźnik *mTimer* na obiekt typu *QTimer*, który jest używany w konfiguracji numer

trzy. Klasa *QTimer* dostarcza liczniki pozwalające na odmierzanie chwil czasowych do odświeżania interfejsów bądź uruchamiania zdarzeń w czasie. Wykorzystanie w praktyce zostanie omówione przy konkretnej konfiguracji. Należy jednak nadmienić, że w konstruktorze klasy *UserPanel* zostają wybrane konkretne ustawienia obiektu typu *QTimer*. Typ licznika zostaje ustawiony na *singleShot*, oznacza to, że przypisane zdarzenie jest uruchamiane tylko raz po upłynięciu określonej chwili. W konfiguracji, w której licznik jest wykorzystywany przypisana zostaje metoda *StartPlayback*, omówiona w dalszej części tego podrozdziału. Kolejnymi polami są dwie tablice typu logicznego reprezentujące stany pól wyboru służących do oceny filmów. Ich wykorzystanie zostanie omówione przy opisie systemu oceniania filmów. Klasa *UserPanel* zawiera również numer aktualnie odtwarzanego filmu, oraz pole typu string przechowujące wyniki testów gotowe do zapisania w pliku tekstowym z wynikami. Klasa zawiera także cztery widżety: *startWidget, ratingWidget*, *ratingWidget_2* oraz *chooseMovieWidget.*

Rysunek 23 Panel startowy testów.

Widżet startowy służy podaniu imienia testera mające na celu identyfikacje, czy też odróżnienie od siebie kolejnych osób. Prośba o podanie imienia ma również na celu przywiązanie testera do administratorów, zbudowanie swoistej relacji, daje poczucie większej odpowiedzialności za swoje wyniki niż w przypadku zupełniej aminowości. Po naciśnięciu na przycisk z napisem „Rozpocznij test" wywołana zostaje metoda *on_pushButton_clicked* rozpoczynająca test. Po wywołaniu metody zostaje odczytana zawartość pola typu *TextEdit*, będącego miejscem na wpisanie imienia przez użytkownika. Zawartość zostaje wpisana do pola *testsOutputToFileString* klasy *UserPanel* przechowującego zawartość do wpisania w plik wynikowy. Następnie w zależności od numeru konfiguracji zostaje wywołana metoda *StartPlayback* uruchamiająca film, bądź zostaje otwarty kolejny widżet. Pozostałe widżety zostaną omówione w dalszej części pracy.

StartPlayback to metoda odpowiedzialna za główną funkcjonalność programu, czyli odtwarzanie nieskompresowanych sekwencji wideo. Filmy są uruchamiane według wartości pola *iActualPlayedMovie* które jest iterowane z każdym wywołaniem metody lub w przypadku konfiguracji oznaczonej numerem 2 ustawiane poprzez wybranie odpowiedniej pozycji na liście. W metodzie tworzone są obiekty typu *RawDataHandler, FramesHandler* oraz *ThreadsHandler*, oraz *OptionsHandler*. Funkcjonalność poszczególnych obiektów została opisana w rozdziale dotyczącym programu odtwarzacza. Należy zwrócić uwagę na ustawienia opcji instancji *LibVLC*. Opcje zostają ustawione tak samo jak w przypadku odtwarzania wideo bez własnego interfejsu jednak parametry takie jak rozdzielczość czy ilość klatek na sekundę zostają wczytane z obiektu typu *PlayerConfigurationsHandler* do którego *UserPanel* posiada referencje. W metodzie tworzony jest również obiekt typu *Controler*, którego zastosowanie również zostało omówione w poprzednich podrozdziałach. Na obiekt ten zostaje ustawiony wskaźnik współdzielony klasy *std::shared_ptr*. Jest to konieczne, aby zachować dostęp do obiektu w przypadku programu wielowątkowego. Kolejnym krokiem jest stworzenie kolejnego panelu tym razem typu *VideoPanel,* który zostanie opisany w kolejnym akapicie. Metoda *StartPlayback* posiada również instrukcje warunkowe wynikające z różnych konfiguracji zgodne z przepływem związanego z nią scenariusza testowego. Wywołana zostaje również metoda *StopPlayBackThread* na obiekcie *threadsHandler*. Jest ona odpowiedzialna za zniszczenie okna *VideoPanel* w momencie zakończenia odtwarzania filmu, co zostało już wspomniane we wcześniejszych podrozdziałach.

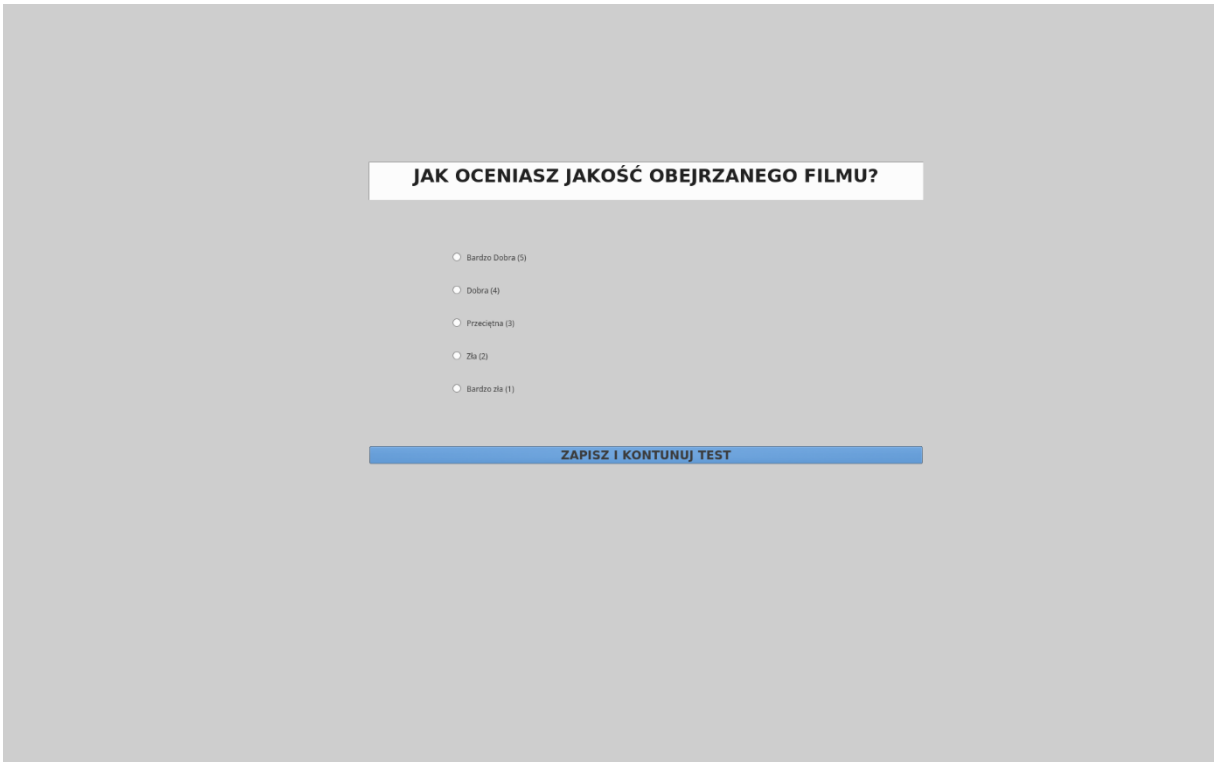
VideoPanel to klasa odpowiedzialna za powiązanie *libvlc_media_player* z oknem odtwarzacza, odbywa się to w konstruktorze tej klasy zgodnie z opisem przedstawionym w przypadku podstawowego połączenia odtwarzacza z *QFrame*, jednakże zamiast *QFrame* użyto obiektu klasy *VlcWidgetVideo*, aby uzależnić go od opcji instancji *libvlc_instance*. Konstruktor wraz z metodą przypisania odtwarzacza do widżetu został pokazany na poniższym rysunku.

```
4   VideoPanel::VideoPanel(int w, int h, std::shared_ptr<Controler> controler, QWidget *parent) :
5       QDialog(parent),
6       ui(new Ui::VideoPanel)
7   {
8
9       ui->setupUi(this);
10      ui->centralWidget->setGeometry(0, 0, w, h);
11      ui->centralWidget->showFullScreen();
12
13      ui->video->showFullScreen();
14      ui->video->setGeometry(0, 0, w, h);
15
16      libvlc_media_player_t* mp = controler->m_DisplayHandler->m_pMediaPlayer;
17      int windid = ui->video->winId();
18      libvlc_media_player_set_xwindow (mp, windid);
19
20
21  }
```

Rysunek 24 Widok ekranu z konstruktorem panelu VideoPanel z przypisaniem odtwarzacza.

Jak zostało wspomniane we wcześniejszym fragmencie tego podrozdziału w zależności od wybranej konfiguracji uruchamiane są różne widżety do oceniania: *ratingPanel* i *ratingPanel2*. Zostały one stworzone jako osobne widoki, bez tworzenia klas, ze względu na brak konieczności posiadania osobnej logiki. Oba widżety posiadają niemal identyczną konstrukcję. W górnej części znajduje się pasek z pytaniem zadawanym do testera, a w środkowej znajdują się przyciski opcji (typu *QRadioButton*). Przyciski zostały umieszone w jednym obszarze interfejsu co pozwala na stworzenie grupy. Grupa przycisków automatycznie łączy je, nadając im parametr ekskluzywności. Oznacza to że tylko jeden z nich może być zaznaczony w danej chwili, a zaznaczenie innego powoduje wyłączenie poprzedni zaznaczonego. Parametr ten można zmieniać przy pomocy metody *setAutoExclusive,* która przyjmuje argument typu logicznego oznaczający stan docelowy parametru. Metoda ta została użyta w metodzie *UncheckToggles* w klasie *UserPanel*, która została omówiona poniżej. Każdy z przycisków opcji posiada własną metodę *on_buttonNumber_toggled,* która zostaje wywołana przy każdorazowej zmianie stanu przycisku. Wywołanie metody ustawia element tablicy pola *states* (lub *states2* w scenariuszu numer 3) o numerze przycisku w zależności od stanu. W dolnej części widżetów do oceniania znajdują się przyciski służące do kontynuacji przepływu testu. Przycisk kontynuacji testu posiada różne zachowania zależne od etapu testu i konfiguracji. Jego zachowanie w konkretnych sytuacjach zostanie omówione w dalszej części rozdziału. Istotną w każdym scenariuszu rolą jest jednak zbieranie po kliknięciu informacji o wybranej przez użytkownika ocenie filmu. Odbywa się to przy pomocy pętli która przegląda odpowiednią tablice *states* w poszukiwaniu elementu ustawionego na wartość *„true*". Numer zaznaczonego przycisku opcji oznacza wystawioną przez testera ocenę. Wystawione oceny zostają rzutowane na stringa i zapisane do pliku w metodzie *WriteToFile.*

UncheckToggles to metoda pomocnicza służąca odświeżaniu przycisków typu *QRadioButton* na widżetach do oceny. Funkcjonalność tej metody jest standardową funkcjonalnością wyłączającą wszystkie przyciski opcji. Wyłącznie aktywnego przycisku opcji nie ogranicza się jednak do ustawienia stanu *QRadioButton* na nieaktywny, ponieważ w przypadku ustawionej opcji ekskluzywności odznaczenia jednego z przycisków jest możliwe tylko w przypadku zaznaczenia innego, dlatego też w metodzie *UncheckToggles* konieczne jest wyłączenie automatycznej ekskluzywności wszystkich przycisków w grupie. Jest to możliwe przy pomocy wspomnianej już metody *setAutoExclusive* z klasy *QRadioButton*. Następnie na każdym z przycisków opcji należy wywołać metodę *setChecked* z parametrem „*false*" przełączając wszystkie opcje w stan wyłączony. Na koniec pozostaje ponowne włączenie automatycznej ekskluzywności.

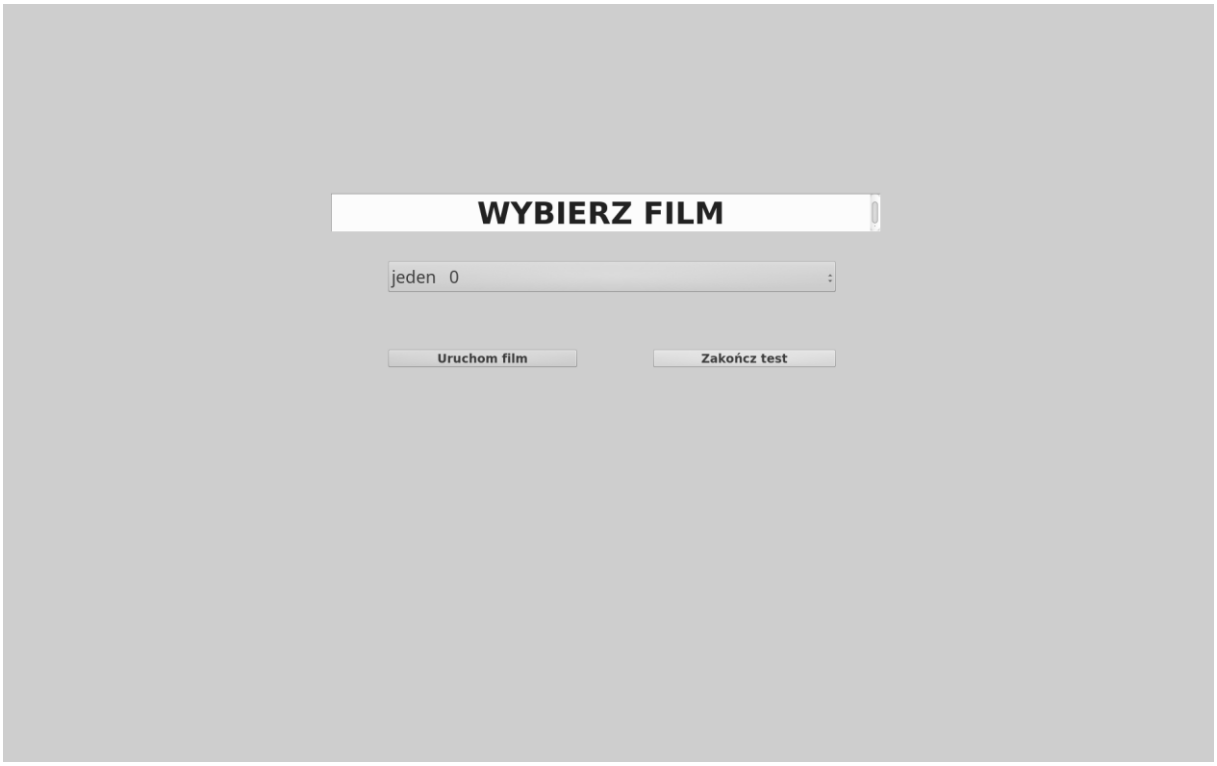


Rysunek 25 Widok ekranu oceny filmu.

Widoczny na powyższym zrzucie ekranu panel to panel typu *UserPanel* z włączonym widżetem *ratingWidget*. Na widżecie widzimy pięć zgrupowanych przycisków opcji – przyciski posiadają opisy tekstowe będące reprezentacją wystawianych ocen, oraz wspomniane przycisk kontynuacji testu. Widżet ten jest widoczny po zakończeniu się każdego z odtwarzanych filmów w scenariuszach oznaczonych w konfiguracji jako 1 i 2 (czyli *ACR* i opcji wyboru filmu z listy).

W przypadku konfiguracji pierwszej po naciśnięciu przycisku oznaczonego jako *„ZAPISZ I KONTYNUUJ TEST*" następuje uruchomienie kolejnego filmu, czyli wywołanie wcześniej opisanej metody *StartPlayback* oraz wyczyszczenie stanów przycisków opcji za pomocą metody *UncheckToggles.* Kolejne filmy są wybierane są według kolejności z konfiguracji. Po zakończeniu filmu użytkownik ponownie zostaje postawiony przed widżetem *ratingWidget* aby ocenić jakość kolejnego nagrania, aż do momentu gdy zostaną odtworzone wszystkie filmy z konfiguracji. Po odtworzeniu ostatniej z wyznaczonych sekwencji wideo napis na przycisku zostaje zmieniony na *„ZAKOŃCZ TEST".* Zmieniając tym samym swoją funkcjonalność. Po kliknięciu wywołana zostaje metoda *WriteToFile,* a okno zostaje zniszczone pozwalając na powrót do głównego panelu.

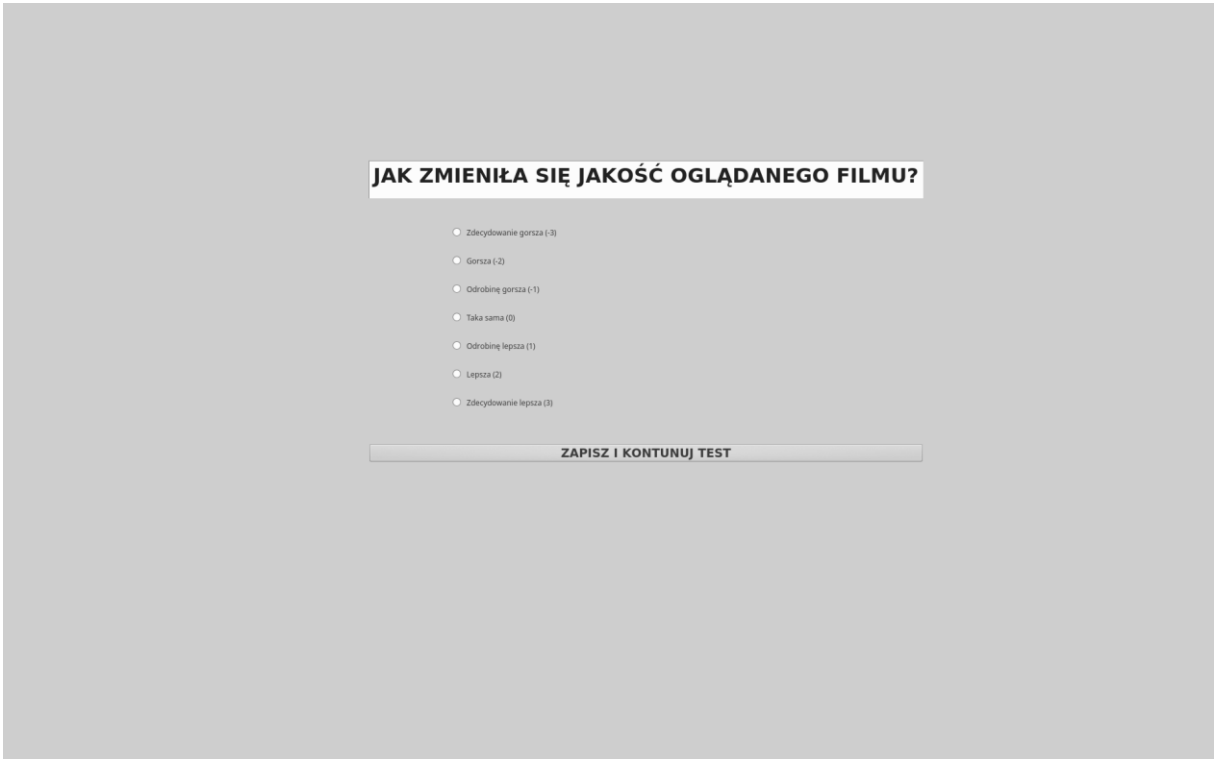
Dla konfiguracji oznaczonej numerem dwa przebieg testu jest zupełnie inny, co wymusza inne działanie interfejsu użytkownika. W tej konfiguracji tester rozpoczyna test od menu pozwalającego mu na wybór filmu z listy. Odbywa się to za pomocą widżetu *chooseVideoWidget* przedstawionego na poniższym zrzucie ekranu.



Rysunek 26 Widok menu wyboru filmów.

Widżet składa się z napisu, rozwijalnej listy (pole wyboru) a także dwóch przycisków oznaczonych w sposób widoczny na powyższym zrzucie ekranu. Przycisk zakończenia testu posiada funkcjonalność zbliżoną do funkcjonalności zakończenia testu o pierwszym numerze konfiguracji. Różnica polega na sposobie zapisu ocen wystawionych przez użytkownika. Oceny są przypisywane do filmów poprzez klasę *MovieProperties* i jej pole *rate*, które zostały opisane w części dotyczącej konfiguracji. Następnie z wektora obiektów typu *MovieProperties* wartości są odczytywane w pętli i zapisywane do pliku tekstowego, a samo okno podobnie jak w poprzednio jest niszczone. Drugi z przycisków znajdujących się na widżecie służy do uruchamiania wybranego z listy filmu poprzez metodę *StartPlayback.* Wybór filmu odbywa się poprzez wybranie pozycji z pola wyboru. Pole wyboru jest reprezentowane przez obiekt *QComboBox,* który udostępnia między innymi metodę *on_activated* wywoływaną przy każdym rozwinięciu pola. Metoda ta ustawia parametr index na wartość wybraną z listy. Parametr ten zostaje użyty do ustawienia pola *iActualPlayedMovie* reprezentującego aktualnie wybranego filmu. Wywołując w następnym kroku metodę *StartPlayback* uruchamiany zostaje film o tym numerze. Pozycje w polu wyboru reprezentują kolejne filmy identyfikując je po nazwach. Obok nazwy w polu znajduję się aktualnie wybrana ocena nadana przez użytkownika danej sekwencji. Jeżeli żadna ocena nie została wybrana w polu widoczne jest zero. Użytkownik zgodnie z koncepcją scenariusza może według uznania obejrzeć każdy filmu kilkukrotnie zmieniając bądź podtrzymując ocenę.

Trzecia opcja konfiguracyjna pozwala na realizacje scenariusza opartego o porównanie dwóch sekwencji np. *DCR* czy *PC.* Zrealizowano scenariusz *PC*, dla którego ze względu na konieczność zmiany skali oceniania należało dodać kolejny widżet. Było to wygodniejsze niż przebudowywanie widżetu *ratingWidget* z kodu w trakcie działania aplikacji. Stworzono widżet *ratingWidget2*.



Rysunek 27 Widok oceny porównawczej pary filmów.

Widżet *ratingWidget2* widoczny na powyższym widoku ekranu został zaprojektowany w identyczny sposób co widżet *ratingWidget*. Najważniejsza zmiana w widocznej części interfejsu to zmiana ilości przycisków opcji oraz tekstów identyfikujących możliwe do nadania oceny. Ponieważ test ma charakter porównawczy, testerowi zadano inne pytanie w polu tekstowym. Ze względu na fakt, iż w skali występują oceny ujemne, wystąpiła konieczność przesunięcia wartości ocen, które dotychczas były równe numerom przycisków w dół, aby uzyskać konieczne wartości. Istotne zmiany w tej konfiguracji następują w metodzie *StartPlayback,* a także w konstruktorze całego panelu *UserPanel,* co zostało już wspomniane. Konfiguracja trzecia wymusza konieczność uruchamiania drugiego filmu od razu po pierwszym. Ponieważ zmiana odtwarzanego filmu wymusza zmianę instancji *libvlc_media,* co z kolei zmusza do zmiany instancji *libvlc_media_player* uznano, iż łatwiejszym rozwiązaniem będzie zniszczenie starego panelu *VideoPanel* i utworzenie kolejnego, dlatego też funkcja *StartPlayback* zostaje wywołana ponownie po czasie odtwarzania poprzedniej sekwencji poprzez zdarzenie oparte na liczniku czasu typu *QTimer*. Dopiero po dwóch sekwencjach widoczny jest widżet pozwalający na ocenę. Ocena jest przekazywana do zapisu do pliku w ten sam sposób co w przypadku konfiguracji pierwszej.

Stworzony w ten sposób interfejs pozwala na przeprowadzanie testów w trzech wybranych konfiguracjach. Interfejs można dodatkowo rozszerzyć o lokalizację za pomocą udostępnianego w ramach biblioteki *QT* narzędzia *QT Linguist* pozwalającego na tłumaczenie. Narzędzie to można również wykorzystać do podmiany pół tekstowych, co pomogłoby skonstruować inne, nowe scenariusze testowe. Całość interfejsu została utrzymana w szarych, stonowanych kolorach niepowodujących

rozproszenia uwagi testera, jednakże korzystano z podstawowych, szablonowych grafik, dlatego też istotnym krokiem w rozwoju aplikacji byłaby wymiana grafik na dedykowane.