SFML – Budowanie aplikacji (sheadovas/poradniki/howto/sfml-budowanie-aplikacji/)

Mar 31, 2017 / howTo (sheadovas/category/poradniki/howto/)

Budowanie aplikacji dla „klienta” oraz o linkowaniu statycznym i dynamicznym.

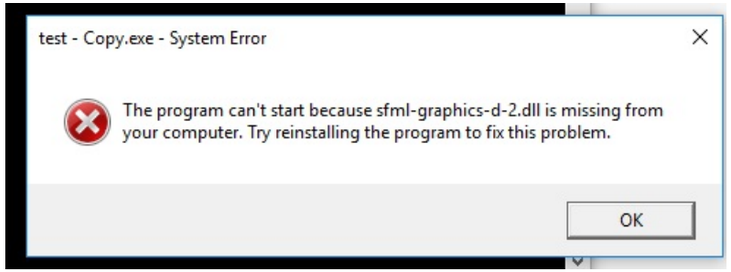
Hej, dzisiaj chciałbym się zająć pozornie trudnym tematem, który jak się okaże pod koniec wpisu jest trywialnie prostu.

> Uwaga! Cały wpis poświęcam budowaniu pod kątem Windowsa, w przypadku Linuksa jest prościej ;)

Tło fabularne

Jakiś czas temu na [forum Pasji Informatyki (http://forum.pasja-informatyki.pl)] zauważyłem, że całkiem sporo osób ma problem z finalnym zbudowaniem swojej gry, tak aby można było ją uruchomić poza lokalnym środowiskiem IDE, czy też na komputerze osób nieposiadających biblioteki SFML.

Próba uruchomienia aplikacji poza IDE kończy się następująco:



(https://i0.wp.com/www.shead.ayz.pl/wp-content/uploads/2017/03/build-0.png)

Dzisiaj pokażę jak poradzić sobie z tą sytuacją.

Jak zbudować aplikację?

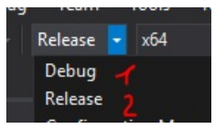
Sam przepis jest względnie uniwersalny i może dotyczyć każdego projektu. Jednakże w tym wpisie zajmiemy się konfiguracją specyficzną dla SFML, w gwoli ścisłości całość pokażę krok po kroku dla Visual Studio (kroki dla innych środowisk są analogiczne).

Do tego poradnika będzie nam potrzebne SFML, ja korzystam z wersji 64bitowej. Warto też mieć otwartą przed sobą [dokumentację (https://www.sfml-dev.org/tutorials/2.4/start-vc.php)], bo będziemy z niej korzystać.

Debug vs Release

Pierwsza rzecz jaką należy zauważyć przy wydawaniu aplikacji jest to, że środowiska zazwyczaj dają możliwość skompilować ją w dwóch konfiguracjach:

- *Debug*
- *Release*



(https://i1.wp.com/www.shead.ayz.pl/wp-content/uploads/2017/03/build-1.png)

W skrócie:

- *Debug* służy do kompilowania aplikacji na czas testów, kompilator dodaje tutaj dodatkowe symbole pomagające przy debugowaniu;
- *Release* jest wersją „dla klienta”, tzn nie posiada jemu niepotrzebnych symboli, ani dodatkowych komunikatów (dzięki czemu waży też mniej).

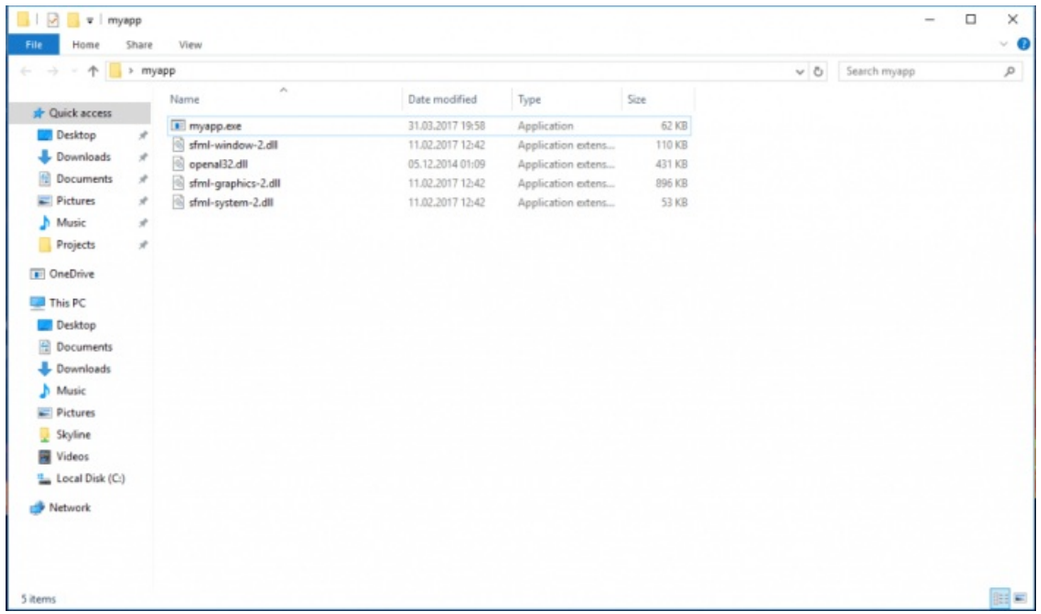
Jeżeli chcecie dostarczyć wasz produkt „klientowi”, to powinniście skorzystać z konfiguracji *Release*. Jeżeli podążaliście tutorialem z linku powyżej to powinniście mieć skonfigurowany SFML dla wszystkich konfiguracji.

Linkowanie dynamiczne vs statyczne

Kolejnym krokiem jest wybranie sposobu linkowania bibliotek. Każda z opcji ma swoje wady i zalety i pokrótce je tutaj przedstawię.

Linkowanie dynamiczne

Linkowanie dynamiczne cechuje się tym (dla klienta), że obok pliku wykonywalnego (.exe) są pliki bibliotek (.dll), które są wymagane do poprawnego działania aplikacji.



(https://i1.wp.com/www.shead.ayz.pl/wp-content/uploads/2017/03/build-2.png)

Co to oznacza oprócz powyższego faktu w praktyce?

- \+ plik wykonywalny waży mniej,
- \+ z tych samych plików bibliotek może korzystać wiele aplikacji,
- – odrobinę wolniejsze uruchamianie.

W przypadku SFML aby zbudować aplikację dynamicznie należy:

1. dodać makro „SFML_DYNAMIC” w *Properties -> Configuration properties -> C/C++ -> Preprocessor*
2. w *Properties -> Configuration properties -> Input -> Additional Dependencies* należy dodać biblioteki zbudowane pod linkowanie dynamiczne, a więc np.: sfml-graphics.lib, sfml-system.lib, sfml-window.lib (bez żadnych post-fixów).

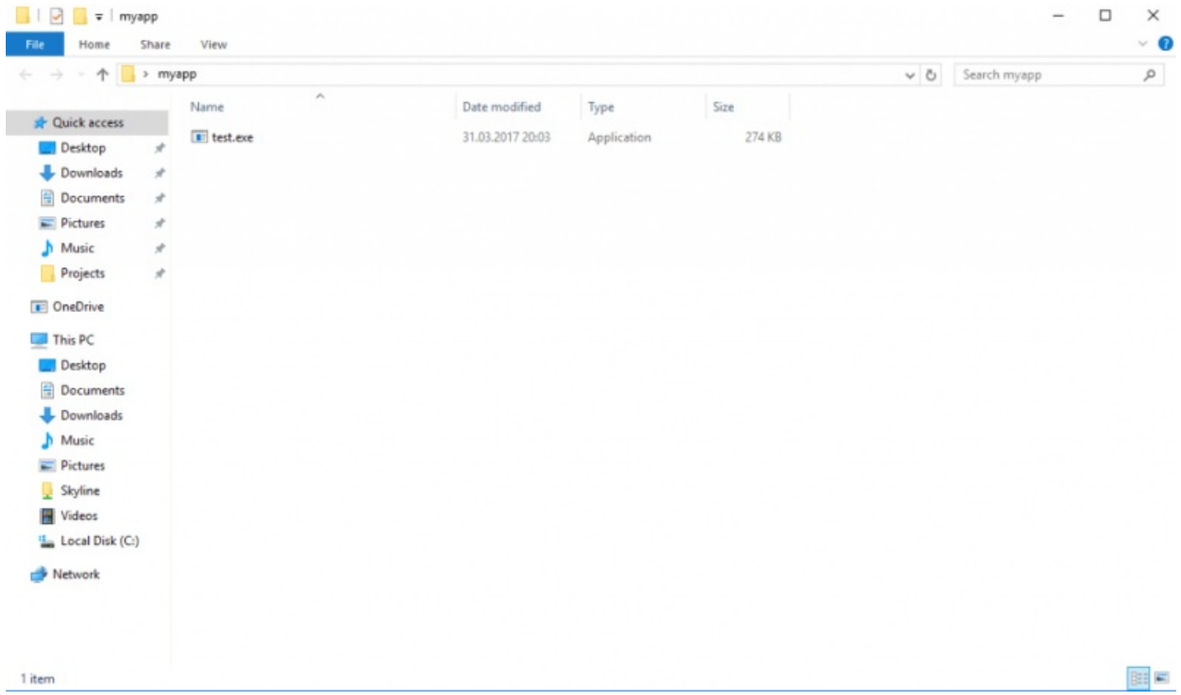
Następnie należy przygotować folder zawierający potrzebne pliki do uruchomienia aplikacji, a więc:

1. plik wykonywalny – po skompilowaniu znajduje się pod ścieżką: *<project_root>\<arch>\Release*, np: *test_project\x64\Release\test.exe*
2. pliki *dll*, dla SFML istnieją w wersji przekompilowanej pod ścieżką: <SFML_root>\bin

Przykład gotowego folderu z wszystkimi potrzebnymi plikami znajduje się powyżej na screenie.

Linkowanie statyczne

Tutaj mamy odwrotną sytuację do poprzedniej, a więc nie potrzebujemy żadnych dodatkowych plików poza plikiem wykonywalnym, ponieważ biblioteki są „wtopione” w plik wykonywalny.



(https://i1.wp.com/www.shead.ayz.pl/wp-content/uploads/2017/03/build-3.png)Cechy tego rozwiązania:

- \+ nieco szybsze uruchomienie,
- \+ łatwiejsza dystrybucja,
- – większy rozmiar aplikacji,
- – nieco większe zużycie zasobów (z tych samych bibliotek może korzystać tylko jeden program wykonywalny).

Aby zbudować aplikację statycznie należy:

1. dodać makro „SFML_STATIC” w *Properties -> Configuration properties -> C/C++ -> Preprocessor*
2. w *Properties -> Configuration properties -> Input -> Additional Dependencies* należy dodać biblioteki zbudowane pod linkowanie statyczne, a więc np.: sfml-graphics-s.lib, sfml-system-s.lib, sfml-window-s.lib (z post-fixem „-s”); oprócz tego należy dodać wszystkie biblioteki [zależne (https://www.sfml-dev.org/faq.php#build-link-static)], np: opengl32.lib, freetype.lib, jpeg.lib, winmm.lib

Dystrybucja programu odbywa się przez dostarczenie finalnego pliku „.exe” do klienta (nie trzeba dostarczać żadnych dodatkowych plików).

Kiedy linkować statycznie, kiedy dynamicznie?

Tutaj wszystko zależy od waszych preferencji i samego projektu. W moim odczuciu, aby linkować statycznie trzeba mieć dobry powód, na pewno darowałbym je sobie dla trybu *Debug* oraz sytuacji gdy często jesteśmy zmuszani do aktualizowania kodu programu, dzięki linkowaniu dynamicznemu musimy podmienić tylko zmieniane pliki, a nie wszystko jak to by było w przypadku linkowania statycznego.

Dociekliwych zachęcam do samodzielnej eksploracji tego tematu, bo tutaj jedynie napomknąłem o pewnych rzeczach ;)

Podsumowanie

Dzisiaj zgłębiliśmy „tajniki” budowania aplikacji SFML w wersji „dla klienta”. Mam nadzieję, że wpis jest przydatny, a samo budowanie programów stało się znacznie prostsze.

Tradycyjnie zachęcam do podzieleniem się tym wpisem ze swoimi znajomymi, a także zachęcam do systemu komentarzy oraz strony na facebook’u.

Code ON!