**Automatyczne komponowanie muzyki**

**Filip Czajkowski**

**Nr albumu 131481**

**Katedra Architektury Systemów Komputerowych**

**Praca dyplomowa z kierunku Informatyka, studia stacjonarne II stopnia**

# Streszczenie:

Poniższy dokument opisuje powstały projekt dyplomowy magisterski, którego celem było stworzenie aplikacji komponującej muzykę w dopasowaniu do zadanej linii melodycznej. W tym celu powstał *MusicAnalyzer*, program układający zapis nutowy akordów dla fortepianu, który może służyć jako akompaniament przy wykonywaniu utworu podanego na wejściu. Dokumentacja wykonanego projektu opisuje interfejs aplikacji, logikę działania programu, wykonane testy uzyskanego rozwiązania oraz ocenę jakości wskazaną przez osoby testujące.

# Abstract:

This document describes a graduation project, which aim was to create an application capable of composing an accompaniment to a melody read from input file. Thus *MusicAnalyzer* arose, a program creating music notation of chords for a piano, which can be played as an accompaniment to a melody from input file. The documentation of a given project describes user interface of a desktop application, algorithm composing the music, tests of few final result music compositions and their feedback given by people assessing program’s efficiency.

# Spis treści

[Streszczenie: 1](#_Toc460550329)

[Abstract: 1](#_Toc460550330)

[Spis treści 2](#_Toc460550331)

[1. Wykaz stosowanych pojęć i skrótów 4](#_Toc460550332)

[2. Wstęp i cel pracy 5](#_Toc460550333)

[2.1. Wstęp 5](#_Toc460550334)

[2.2. Cel pracy 5](#_Toc460550335)

[3. Opis dziedziny 7](#_Toc460550336)

[3.1. Opis tematu projektu dyplomowego 7](#_Toc460550337)

[3.2. Podstawowe pojęcia muzyczne i funkcje harmoniczne 7](#_Toc460550338)

[3.3. Przegląd dostępnych rozwiązań 7](#_Toc460550339)

[3.3.1. Emily Howell 7](#_Toc460550340)

[3.3.2. Mezzo 9](#_Toc460550341)

[3.3.3. AthTek DigiBand 11](#_Toc460550342)

[4. Opis rozwiązania 13](#_Toc460550343)

[4.1. Opis interfejsu graficznego programu MusicAnalyzer 13](#_Toc460550344)

[4.2. Informacje bazowe – pliki konfiguracyjne 13](#_Toc460550345)

[4.3. Wczytywanie melodii wejściowej i metody określające jej cechy 14](#_Toc460550346)

[4.3.1. Zdekodowanie sygnałów dźwięków 14](#_Toc460550347)

[4.3.2. Wyszukanie zmian tonacji utworu 15](#_Toc460550348)

[4.3.3. Wyszukanie zmian metrum 15](#_Toc460550349)

[4.3.4. Ustalenie prawidłowej tonacji utworu 15](#_Toc460550350)

[4.3.5. Ustalenie prawidłowych oznaczeń dźwięków 17](#_Toc460550351)

[4.3.6. Ustalenie wartości rytmicznych wszystkich nut 17](#_Toc460550352)

[4.4. Prezentacja graficzna zapisu nutowego 19](#_Toc460550353)

[4.5. Algorytm tworzenia akompaniamentu 22](#_Toc460550354)

[4.5.1. Harmony Search 22](#_Toc460550355)

[4.5.2. Wyszukanie współbrzmień wejściowych 22](#_Toc460550356)

[4.5.3. Wybór węzłów dla akordów 22](#_Toc460550357)

[4.5.4. Adaptacja algorytmu HS 22](#_Toc460550358)

[4.5.5. Ustalenie atrybutów nut akompaniamentu 22](#_Toc460550359)

[4.6. Zapis nutowy i dźwiękowy powstałego akompaniamentu 22](#_Toc460550360)

[4.7. Stosowane rozwiązania pomocnicze 22](#_Toc460550361)

[4.7.1. Sanford.Multimedia 22](#_Toc460550362)

[4.7.2. PSAMControlLibrary 23](#_Toc460550363)

[4.7.3. PSAMWPFControlLibrary 24](#_Toc460550364)

[5. Ocena 25](#_Toc460550365)

[5.1. Założenia testów 25](#_Toc460550366)

[5.2. Wyniki testów 25](#_Toc460550367)

[5.3. Kryteria oceny 25](#_Toc460550368)

[5.4. Ocena jakości 25](#_Toc460550369)

[6. Podsumowanie 26](#_Toc460550370)

[7. Wykaz rysunków 27](#_Toc460550371)

[8. Wykaz tabel 28](#_Toc460550372)

[9. Wykaz literatury pomocniczej 29](#_Toc460550373)

[Odwołania 29](#_Toc460550374)

# Wykaz stosowanych pojęć i skrótów

1. *Test Turinga* - sposób mający dowodzić opanowania przez maszynę umiejętności myślenia w sposób podobny do ludzkiego. Zaproponowany w 1950 roku przez Alan Turinga polega na prowadzeniu rozmowy z człowiekiem i konkurującą maszyną przez osobę mającą tu rolę sędziego. Jeśli nie jest on w stanie prawidłowo wskazać, które z nich jest maszyną, to przyjmuje się, że wówczas zdaje ona test.
2. *Progresja harmoniczna* – szereg współbrzmień następujących po sobie, które tworzą dobrze zharmonizowany układ i wrażenie spójnego motywu muzycznego u odbiorcy.
3. *MIDI* (ang. Musical Instrument Digital Interface) – kompletny system służący do przekazywania informacji dźwiękowych pomiędzy elektronicznymi instrumentami muzycznymi. Pliki zapisane w tym standardzie mogą być odtwarzane na urządzeniach dzięki stałym próbkom dźwięku różnych instrumentów w nich zapisanych, sam plik zaś nie zawiera nagrania dźwięku w żadnej formie. Posiada za to kolekcję sygnałów, które odzwierciedlają różne muzyczne działania, takie jak włączenie / wyłączenie dźwięku o zadanej wysokości, naciśnięcie pedału w pianinie czy zmiana tempa. W znajdującej się na początku sekcji nagłówkowej zawiera się także zestaw atrybutów pliku potrzebnych do jego prawidłowego odczytania i odtworzenia.
4. *Tessitura* – w odniesieniu do głosów instrumentalnych lub wokalnych, przedział dźwięków najczęściej używany w danej partii.
5. *Bitmapa* – obraz w grafice rastrowej, w którym określona jest jednoznacznie zawartość każdego piksela; umożliwia to szybkie i nieskomplikowane obliczeniowo wyświetlanie obrazu, lecz nie pozwala na zwiększenie dokładności w wyniku przybliżenia.

# Wstęp i cel pracy

## Wstęp

Świat nauk ścisłych wielokrotnie w przeszłości stawiał sobie za cel zrozumienie i określenie stałymi regułami fenomenu osiągnięć nauk humanistycznych. Czy można w deterministyczny sposób opisać proces tworzenia przez poetę wiersza albo obrazu rysowanego przez malarza? Czy metody sztucznej inteligencji są w stanie naśladować, wydaję się, cudownie twórczy proces zachodzący w mózgu artysty? Na te pytanie naukowcy starają się odpowiadać twierdząco i popierają to coraz doskonalszymi osiągnięciami w tej dziedzinie, lecz na dzień dzisiejszy nie wydaje się, by los artystów został definitywnie przesądzony.

Muzyka, podobnie jak inne dziedziny sztuki, dostarczają wrażeń (w tym przypadku dźwiękowych), które nie są jednoznacznie odczytywane przez wszystkich odbiorców. Dodatkowo, ocena tego samego utworu muzycznego w zależności od gatunku muzyki, w ramach którego go odbieramy, może się znacząco różnić.

Podobne wyzwania stają przed naukowcami pracującymi nad automatycznym komponowaniem muzyki. Problem ten można podzielić na dwa rodzaje wyzwań. Pierwsze podejście zakłada zupełne zastąpienie pracy kompozytora i umożliwienie tworzenia kompletnej muzyki od podstaw przez komputer. Drugim rodzajem zmagań z tworzeniem muzyki w sposób sztuczny jest układanie akompaniamentu do istniejącej już melodii, lecz w taki sposób, by obie ze sobą dobrze współbrzmiały. Problem ten wydaje się na pierwszy rzut oka nieco prostszy, gdyż posiadamy jakiś punkt zaczepienia, wzór, do którego staramy się dopasować. Jednakże osiągnięcie sukcesu w tym zadaniu niesie ze sobą podobne problemy, co tworzenie muzyki bez wzorca. Ponadto, to podeście wymaga szeregu założeń, które należy przyjąć, aby móc później ocenić jakość stworzonego dopasowania. Czy obie melodie mają być harmonicznie dopasowane? Jakiego typu harmonia ma je wiązać? Czy bardzo podobne do siebie melodie są pożądane, czy raczej oceniane jako nudne, nieciekawe? Czy dysonanse będą odbierane jako nieudolność programu czy raczej jako twórczy dodatek? Na te i wiele innych pytań, twórca oprogramowania tworzącego akompaniament musi sobie wcześniej odpowiedzieć a także zgodnie z nimi powinien być oceniany. Problemy z oceną jakości kompozycji przekładają się także na trudności z wytworzeniem ich, w szczególności gdy odpowiada za ten proces bezduszny program komputerowy.

## Cel pracy

Celem tego projektu było zmierzenie się z problemem automatycznego komponowania muzyki z wykorzystaniem technik sztucznej inteligencji bez oparcia na posiadanej głębokiej wiedzy dziedzinowej. Jedyną muzyką znaną przez program komputerowy jest aktualnie wczytany utwór z pliku wejściowego, do którego powstać ma akompaniament. Nie posiada on zbioru kompozycji, by móc się na nich wzorować, gdyż stawiane przed nim wyzwanie, to utworzenie pasującej melodii dzięki metodom sztucznej inteligencji. Tak powstał *MusicAnalyzer,* program tworzący własną muzykę przy wykorzystaniu algorytmu *Harmony Search* w oparciu o zasady klasycznej harmonii muzycznej. Do działań w ramach pracy dyplomowej należy także dodać ocenę efektywności jego działań przy różnej parametryzacji algorytmu.

# Opis dziedziny

## Opis tematu projektu dyplomowego

Tematem projektu dyplomowego jest automatyczne komponowanie muzyki w postaci tworzenia akompaniamentu do istniejącej melodii. Jak nietrudno się domyślić, jest to problem nieposiadający deterministycznego rozwiązania, które jesteśmy także w łatwy sposób ocenić, porównać z innym. Z tego względu, głównym obiektem pracy nad tym projektem była implementacja algorytmu z dziedziny sztucznej inteligencji (*Harmony Search*), który jest względnie nowym pomysłem i nie posiada wciąż wielu zastosowań. Wybrane podejście do automatycznego generowania muzyki wiąże się z kilkoma przyjętymi założeniami. W odróżnieniu od większości tego typu programów, *MusicAnalyzer* nie posiada bazy kompozycji stworzonych ludzką ręką, by na ich podstawie tworzyć własne utwory korzystając z pewnych modyfikacji i kombinacji. Jego główną siłą jest implementacja algorytmu harmonicznego, który ma za zadanie utworzyć nową ścieżkę dźwiękową będącą w harmonii z linią melodyczną wczytaną w pliku wejściowym. Szukanie prawidłowych współbrzmień a następnie ich ocena zostały oparte na zasadach klasycznej harmonii, którymi posługują się kompozytorzy od wieków. Efektem działań algorytmu jest sekwencja zmieniających się w kluczowych momentach utworu akordów. Element improwizacji i tworzenie wyszukanej wiodącej melodii celowo nie były zgłębiane, gdyż prace skupione były na szukaniu prawidłowego tła dla melodii już istniejącej. Program przystosowany jest do wczytania pliku z zapisem wielu ścieżek dźwiękowych jednocześnie i wszystkie z nich są jednakowo brane pod uwagę w szukaniu współbrzmienia z powstającym akompaniamentem.

## Podstawowe pojęcia muzyczne i funkcje harmoniczne

## Przegląd dostępnych rozwiązań

## Emily Howell

*Emily Howell[[1]](#footnote-1)* to nazwa programu komputerowego autorstwa Davida Cope’a powstałego w latach 90-tych na uniwersyteciew Santa Cruz w Kalifornii. Projekt jest kontynuacją pracy uczonego nad programem *EMI* („Experiments in Musical Intelligence”). Zasilany był bazą utworów wybranego kompozytora a następnie tworzył nową kompozycję, która stylistyką nawiązywała do utworów z zestawu wejściowego. *Emily Howell* funkcjonuje w analogiczny sposób, natomiast jako dane wejściowe posiada jedynie utwory wytworzone przez *EMI*, dzięki czemu, przynajmniej według jego twórcy[[2]](#footnote-2), wykształciła własny styl kompozytorski.

Szczegóły algorytmu zastosowanego w programie *EMI* nie są znane, natomiast wiadomo, iż jest on oparty na kilku krokach:

|  |  |
| --- | --- |
| https://patentimages.storage.googleapis.com/US7696426B2/US07696426-20100413-D00001.png  Rysunek Schmeta działania Emily Howell, źródło https://www.google.com/patents/US7696426 | 1. Utworzenie bazy źródłowej kompozycji, których styl chcemy naśladować 2. Znalezienie wzorców (motywów dźwiękowych i rytmicznych) 3. Podział dzieła na segmenty zgodnie ze znalezionym rytmem 4. Analiza hierarchiczna – wszystkim współbrzmieniom przypisywane są ich funkcje, jakie pełnią w danej frazie; pomaga to określić np. które akordy po sobie występują; 5. Nieliniowe rekombinacje i modyfikacje powstających fragmentów z posiadanych wzorców 6. Wybór najlepszego utworu i zwrócenie go na wyjście programu |

Dodatkowo program jest wciąż ulepszany, gdyż powstające utwory są oceniane przez dr Cope’a i wyniki jego szczegółowej oceny wpływają na proces tworzenia kolejnych wyników.

Rozwiązania stosowane w obu projektach okazały się na tyle wydajne, iż David Cope posiadł patent[[3]](#footnote-3) na terenie Stanów Zjednoczonych chroniący jego autorski algorytm komponowania muzyki. Ponadto program *EMI* w 1997 roku na Uniwersytecie w Oregonie zmierzył się w teście[[4]](#footnote-4) przypominającym muzyczny odpowiednik Testu Turinga*.* Na podstawie posiadanej bazy utworów J. S. Bachastworzył własną kompozycję (miniaturę fortepianową) imitującą styl geniusza epoki baroku. Podobne zadanie otrzymał szanowany, współczesnykompozytorSteve Larson*.* Następnie pianistka wykonała przed oceniającą publicznością powyższe utwory oraz oryginalny utwór Bacha. Zebrani mieli za zadanie ocenić, która z kompozycji jest oryginałem, która stworzona przez współczesnego kompozytora, a która przez program *EMI*. Ich werdykt był bardzo zaskakujący, gdyż to komputerowo wytworzony utwór został uznany za oryginalne dzieło z XVII wieku, za to utwór Larsona wzięto za wytwór sztucznej inteligencji, natomiast pierwotna kompozycja Bacha została odebrana jako imitacja autorstwa współczesnego kompozytora.

Wyniki przedstawionego powyżej konkursu nie dają nam pewności co do pozytywnej oceny walorów artystycznych kompozycji programu *EMI*, gdyż publiczność w swoim głosowaniu źle oceniła utwór samego Bacha. Można jednak sądzić, że efekty obliczeń *Emily Howell* można przyjąć za dobrej jakości muzykę, ponieważ doczekały się one dwóch albumów na płytach: *From Darkness, Light* (2009) i *Breathless* (2012). Niemniej jednak, sztuczne tworzenie muzyki jest wciąż źle odbierane przez środowisko artystyczne, przez co wielu muzyków odmawia wykonywania utworów autorstwa *Emily*.

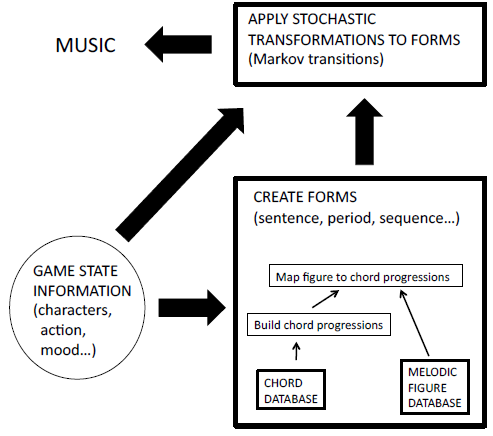
## Mezzo

W 2012 roku na tej samej uczelni, na której działał David Cope, pojawił się kolejny projekt z gatunku automatycznego komponowania muzyki. Wówczas Daniel Lankford Brown ukończył swój przewód doktorski o temacie *Expressing narrative function in adaptive, computer-composed music[[5]](#footnote-5)*. W ramach swoim badań stworzył program komputerowy *Mezzo*, który generuje na żywo tło muzyczne do dowolnej gry komputerowej prowadzonej przez użytkownika komputera. Powstająca muzyka pochodzi z gatunku neoromantyzmu, gdyż zgodnie z zamierzeniem, zasady harmonii i komponowania melodii zapisane w programie były konsultowane z takimi muzykologami jak V. Kofi Agawu, M. Grabócz, W. Caplin, R. Hatten czy B. Almén.

Projekt ten starał się sprostać przede wszystkim dwóm wyzwaniom. Jak stworzyć melodię, która ciągle ewoluuje, lecz mimo powtarzanych czynności przez gracza nie staje się nudna? Drugim wyzwaniem jest problem dopasowania muzyki do emocji przeżywanych przez gracza. Według autora, można im sprostać, gdy sposób powtarzania motywów muzycznych będzie czymś więcej niż repetycją, gdy zawarte w nich frazy i dźwięki będą miały określone funkcje, które będą się stopniowo zmieniać, ewoluować a niektóre z nich będą się wyróżniać tworząc główną linię melodyczną.

Tworzenie własnych kompozycji opiera się wpierw na wczytaniu istniejących motywów muzycznych oraz progresji harmonicznych. Następnie przy pomocy algorytmu genetycznego powstają indywidualne progresje tworzone jednak na wzór tych wejściowych, które dzięki temu posiadają wspólny styl muzyczny. Wówczas, kiedy gracz uruchamia grę i w nią gra, program w czasie rzeczywistym stara się dopasować posiadane frazy muzyczne do odpowiednich momentów w grze, zachowując przy tym ich ciągłość, łagodne przejścia i podobieństwo między kolejnymi motywami. W ramach kolejnych fraz generowane są kolejne zaburzenia, kulminacje, aby zsynchronizować je z tym, co się dzieje w grze. Dla przykładu, w odpowiedzi na osiągnięcie sukcesu przez gracza, pojawia się progresja wstępująca, która odzwierciedla emocje związane radością i tryumfem. Analogicznie, w momencie porażki pojawia się melodia zstępująca, często w tonacji molowej, która daje odczucie zawiedzenia, straconej szansy. Aby uniknąć zbytniej powtarzalności tych modyfikacji, są one dobierane stochastycznie: możliwe transformacje są zamodelowane w postaci łańcuchów Markowa. Powtórzenia motywów i przejścia między określonymi frazami są oznaczone pewnym prawdopodobieństwem (zgodnie z parametrami dla danego fragmentu gry), przez co modyfikacja regresji harmonicznych nawet w tym samym fragmencie gry jest za każdym razem nieco inna.

Poniższy schemat przedstawia sposób generowania akompaniamentu muzycznego przez program *Mezzo* podczas gry.



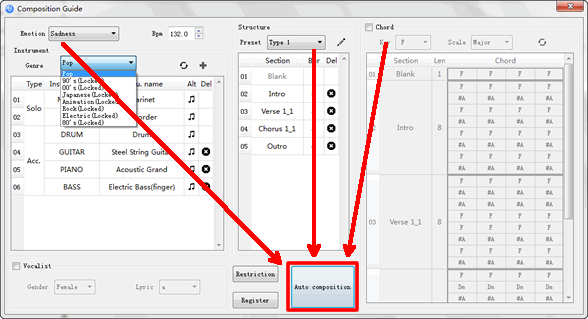
Rysunek Schemat działania programu Mezzo, źródło http://www.danielbrownmusic.com/uploads/1/3/2/3/13234393/final\_dissertation\_final\_edit.pdf

s. 77

Na początku każdej iteracji następuje pobranie informacji o stanie gry (emocje, osiągnięcia, przewidywany humor gracza itd.). Następnie przy wykorzystaniu posiadanej bazy współbrzmień i figur melodycznych, powstają progresje muzyczne na najbliższe kilka lub kilkanaście taktów, które tworzą zdania muzyczne i całe frazy. W ostatnim etapie zachodzą transformacje stochastyczne aby wprowadzić do powstałych fraz element twórczy, niepowtarzalny.

## AthTek DigiBand

*DigiBand*[[6]](#footnote-6) to rozwiązanie komercyjne, które umożliwia łatwe generowanie linii melodycznych dla wybranych instrumentów. Można wskazać rytm, metrum, instrumenty, styl muzyki oraz charakter (klimat emocjonalny) utworu, lub też nagrać próbkę dźwiękową i program sam ułoży pozostałe ścieżki dźwiękowe w dopasowaniu do wczytanej melodii. Jest to narzędzie reklamowane jako pomoc przy tworzeniu własnych kompozycji, gdyż wedle wyboru instrumentów i gatunku pożądanej muzyki, resztę pracy wykonuje zamiast człowieka. Jego zaletą jest także szybkie działanie, lecz niestety brak informacji o algorytmie z jakiego korzysta przy generowaniu muzyki.



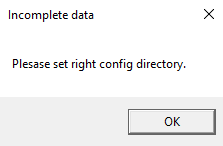
Rysunek Interfejs programu DigiBand, źródło http://www.athtek.com/image/digiband/composition.jpg, strona odwiedzona 1.09.2016

# Opis rozwiązania

## Opis interfejsu graficznego programu MusicAnalyzer

## Informacje bazowe – pliki konfiguracyjne

Do prawidłowego działania programu potrzebny jest zestaw plików konfiguracyjnych, które są dołączone także do projektu *MusicAnalyzer*. Aplikacja wymaga wskazania lokalizacji, w której mają się one znajdować. Jeśli ich tam nie znajdzie, podczas wczytywania pliku wejściowego pokaże się komunikat:



Rysunek Komunikat o braku plików konfiguracyjnych w MusicAnalyzer

Dalsze działanie programu zostanie przerwane, konieczne będzie wskazanie innej lokalizacji na komputerze zawierającej wszystkie niżej wskazane pliki:

* **basicPitches.txt** – plik zawiera 12 wierszy, w których wyszczególnione są wszystkie podstawowe dźwięki gamy (C-dur) w postaci pary nr kolejny i oznaczenie muzyczne np. „0 C”; dzięki niemu możliwe jest np. przyporządkowanie nazw dźwięków do określonej wartości zapisanej w sygnale MIDI
* **minorSequence.txt** – plik ten zawiera 7 wierszy (tyle dźwięków zawiera każda gama), a każdy z nich zawiera numer kolejny dźwięku gamy molowej z ciągu wszystkich dźwięków dostępnych na klawiaturze fortepianu w obrębie oktawy (zawartych w pliku *basicPitches.txt*); umożliwia to rozpoznanie, czy dany dźwięk należy do skali oraz z jakich dźwięków można składać akordu tonalne
* **majorSequence.txt** – ten plik zawiera analogiczne 7 wierszy, jak opisany powyżej, z tym, że jest to ciąg dźwięków dowolnej gamy durowej
* **majorChords.txt** – zawiera spis akordów (trójdźwięków) charakterystycznych dla gamy durowej:
  + tonika
  + subdominanta
  + dominanta (w skali harmonicznej)
  + trójdźwięk na 2-im stopniu gamy
  + trójdźwięk na 3-im stopniu gamy
  + trójdźwięk na 6-ym stopniu gamy

widza ta pozwala stworzyć powyższe akordy odpowiednie dla każdej tonacji

* **minorChords.txt** – zawiera spis akordów (trójdźwięków) charakterystycznych dla gamy molowej:
  + wszystkie trójdźwięki wskazane dla gamy durowej
  + dominanta (w skali eolskiej)
  + trójdźwięk na 3-im stopniu gamy (w skali eolskiej)

## Wczytywanie melodii wejściowej i metody określające jej cechy

Proces wczytywania i analizowania informacji zawartych w wejściowym pliku *MIDI* w programie *MusicAnalyzer* składa się z kilku etapów:

## Zdekodowanie sygnałów dźwięków

Obiekt pliku dźwiękowego przechowuje klasa *Sequence* (z biblioteki *Sanford*), która jest kolekcją obiektów typu *Track*, które odpowiadają ścieżkom dźwiękowym, do których przypisane są sygnały dźwiękowe. Należy zatem iterować każdą z nich, aby uzyskać wszystkie dźwięki. Jednak do ścieżki dźwiękowej przypisane są nie tylko sygnały pojawienia się i wygaśnięcia dźwięku, dlatego z kolekcji obiektów *MidiEvent* należy wybrać tylko te będące typu *ChannelMessage*. Następnie jeśli mamy do czynienia z sygnałem o komendzie *NoteOn*, to jest to sygnał pojawienia się dźwięku, który dodatkowo posiada atrybuty:

* Wysokość dźwięku (możliwe wartości z zakresu *<A0 = 21; C8 = 109>*)
* Poziom głośności dźwięku (0 – 100)
* Umiejscowienie w czasie (ilość taktowań od początku utworu)
* Nr ścieżki dźwiękowej

Niestety w plikach *MIDI* moment zakończenia dźwięku jest przez różne urządzenia oznaczany dwojako: poprzez nadanie sygnału *NoteOff* lub sygnału *NoteOn*, którego poziom natężenia dźwięku = 0. W przypadku spełnienia jednego z powyższych warunków zapisywane są dodatkowe atrybuty wskazanej nuty:

* Oznaczenie znakowe nuty (np. C4)
* Jej odpowiednik w podstawowej skali (C)
* Oktawa
* Koniec trwania
* Długość trwania

## Wyszukanie zmian tonacji utworu

Podobnie, jak odszyfrowywanie dźwięków, wyszukiwanie zmian tonacji utworu wymaga przejrzenia wszystkich ścieżek w pliku i przypisanych doń sygnałów. W tym przypadku należy jednak odnaleźć obiekty *MidiEvent* będące typu *MetaMessage*, których atrybut *MetaType = KeySignature*. Tak powstają instancje klasy *Tonation*, które następnie są uzupełniane między innymi o atrybuty:

* *Offset* – przesunięcie tonacji względem skali C-dur
* Zbiór dźwięków gamy należących do tej skali
* Dźwięki wszystkich akordów należących do tej skali

Dodatkowo, po znalezieniu wszystkich zmian tonacji, dla każdej z nich ustalany jest moment jej początku i końca (ilość taktowań od początku utworu).

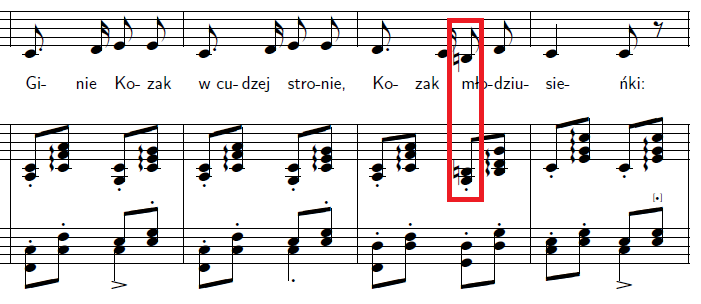
## Wyszukanie zmian metrum

Wyznaczanie metrum i jego zmian odbywa się analogicznie jak dla tonacji, z tą różnicą, że atrybut *MetaType* musi przyjmować wartość *TimeSignature*. Po znalezieniu wszystkich takich zdarzeń, zapisywane jest także momenty początku i końca okresu ich obowiązywania.

## Ustalenie prawidłowej tonacji utworu

Niestety tonacja utworu odczytana z pliku *MIDI* przy pomocy narzędzi z biblioteki *Sanford.Multimedia*[[7]](#footnote-7), nie daje nam gwarancji, że jest prawidłowa. Wynika to z faktu, iż w pliku zapisana jest wartość przesunięcia danej tonacji względem tonacji bazowej (C-dur bądź a–moll). Dlatego, aby odróżnić tonację molową od durowej, która posiada te same znaki przy kluczu, musimy wykonać analizę występowania charakterystycznych dźwięków skali.

1. Dla dowolnej pary tonacji dur – moll, posiadających te same znaki przy kluczu, większość dźwięków występując w zapisie nutowym jest taka sama. Istotna różnica może tkwić w siódmym podwyższonym dźwięków w skali molowej, który w takim przypadku nie ma swojego odpowiednika w skali durowej. Kompozytorzy stosują ten zabieg, aby uwydatnić brzmienie dominanty, która wówczas buduje napięcie w utworze dążące do rozwiązania na akordzie toniki. Przykładem może być pieśń S. Moniuszki „Kozak”, w której przed zakończeniem każdego wersu w zwrotce występuje akord dominanty z podwyższonym VII stopniem gamy i następnie rozwiązuje się on na tonikę.



Rysunek Fragment zapisu nutowego pieśni "Kozak" S. Moniuszki z akompaniamentem pianina (c-moll).

Oczywiście, ten zabieg nie jest obowiązkowy, lecz występuje w ogromnej większości kompozycji klasycznych, co w porównaniu z faktem, iż odpowiadający mu piąty dźwięk w skali durowej praktycznie nigdy nie jest podwyższany, umożliwia dość celnie określić, czy dany utwór jest w skali durowej czy molowej. Skonstruowano zatem warunek realizujący to założenie:

Jeśli *ilość wystąpień VII stopnia* <= (2 \* *ilość wystąpień podwyższonego VII st.*)

To jest to gama molowa

Inaczej jest to gama durowa

1. Drugą cechą utworu poddawaną analizie, jest statystyka akordów na początków taktów, czyli w miejscach najistotniejszych dla budowania rytmu i układu harmonicznego. Są to węzły, które osadzają rytm, występują na nich dłuższe wartości rytmiczne i to one w dużej mierze charakteryzują tonację, w jakiej napisany zostały badany utwór. Dlatego równorzędnym warunkiem dla analizy występowania podwyższonego VII stopnia gamy, jest określenie, czy częściej na początku taktu występuje akord toniki w tonacji dur czy moll:

Jeśli *ilość wystąpień toniki durowej na początku taktu* <= *ilość wystąpień toniki molowej na początku taktu*

To jest to gama molowa

Inaczej jest to gama durowa

Oba opisane powyżej warunki zostały połączone alternatywą, to znaczy aby gamę uznać za molową, wystarczy spełnić tylko jeden z nich.

## Ustalenie prawidłowych oznaczeń dźwięków

Kolejną niejednoznacznością, z jaką należy się zmierzyć korzystając z plików *MIDI*, jest fakt, iż posiadając jedynie bezwzględną wysokość każdego z dźwięków, nie jesteśmy w stanie od razu zaprezentować ich na pięciolinii w zapisie nutowym. Dla przykładu, dźwięk o wysokości 49 może być jednocześnie oznaczony jako Cis4 i Des4. Jest to uzależnione od tonacji utworu w którym te dźwięki występują.

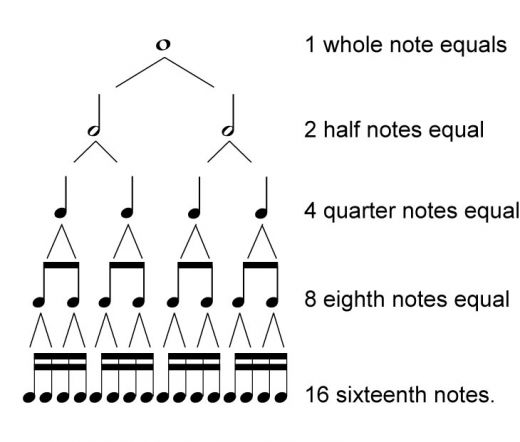


Rysunek Tonacje i znaki przy kluczu wiolinowym, źródło http://blog.gitarius.pl/kurs\_gitarowy/chro4.png

Jeśli wskazana tonacja przy kluczu posiada krzyżyki, wówczas dźwięk ten oznaczylibyśmy jako Cis4, w przeciwnym razie (posiada bemole lub nie posiada znaków przy kluczu), to będzie on oznaczony jako Des4. Ma to znaczenie przede wszystkim pod kątem prawidłowej prezentacji zapisu nutowego, lecz także pozwala uniknąć problemów z porównywaniem dźwięków w innych etapach prowadzonych analiz.

## Ustalenie wartości rytmicznych wszystkich nut

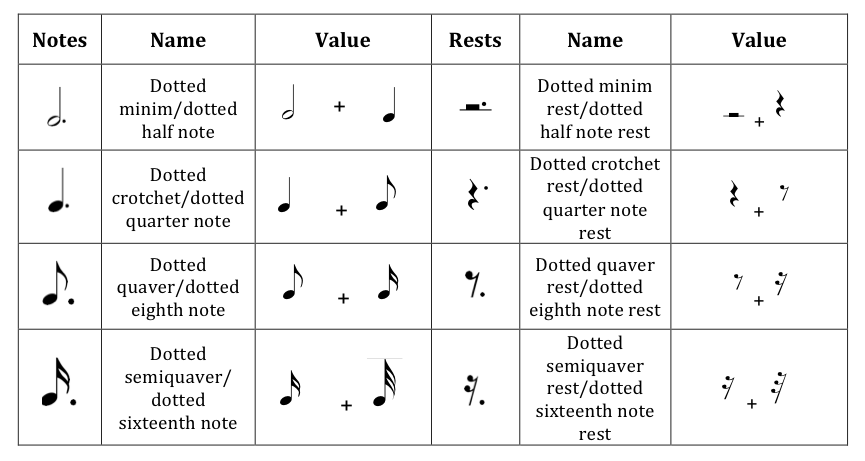
Ostatnim etapem uzupełniania atrybutów wczytanych nut jest prawidłowe dopasowanie wartości rytmicznych względem długości ich trwania. Jest to możliwe dzięki uzyskaniu we wcześniejszych etapach informacji o metrum utworu oraz zapisanej w pliku wartości *Division* – ilości taktowań zegara w ramach jednej ćwierćnuty. Mając tą wiedzę jesteśmy w stanie przypisać wartości rytmiczne dźwiękom w zależności od długości ich trwania. Ten krok jest konieczny dla prawidłowego zaprezentowania zapisu nutowego, dlatego też *MusicAnalyzer* przechowuje ten atrybut w formie typu wyliczeniowego *PSAMControlLibrary.MusicalSymbolDuration*, który jest rozpoznawany przez komponent graficzny (*IncipitViewerWPF*) prezentującego nuty na pięciolinii.



Rysunek Podział wartości rytmicznych, źródło http://whittier.mpls.k12.mn.us/uploads/rhythm\_tree\_1.jpg

Powyższy schemat przedstawia jedynie podstawowe wartości, których długość jest pewną potęgą dwójki, co oczywiście nie wystarcza, by odpowiednio zaprezentować wszystkie nuty. Dlatego zapis nutowy umożliwia przedłużanie wskazanej wartości o połowę poprzez dodanie obok kropki. Możliwe jest także dodawanie kolejnych kropek, wówczas każda z nich przedłuża o dwukrotnie mniejszą wartość (1/4, 1/8 bazowej wartości rytmicznej itd.). Ilustruje to poniższa tabela.

Tabela Wartości rytmiczne przedłużane, źródło https://musictheorysite.files.wordpress.com/2015/06/dotted-value.png



W programie *MusicAnalyzer* w procesie dopasowywania wartości rytmicznych przyjęto założenie, że nuta może mieć maksymalnie dwie kropki przedłużające jej wartość, gdyż w praktyce nie spotyka się ich w większej ilości. Występuje tu jednak dodatkowa trudność w związku z zapisem dźwięków w formacie *MIDI*. Mimo, że w zapisie nutowym dwa dźwięki występują bezpośrednio po sobie, to jednak, aby nie powodować wrażenia nachodzenia ich na siebie oraz by odtwarzacz pliku prawidłowo wyłączył pierwszy dźwięk i włączył kolejny, wszystkie dźwięki zazwyczaj są trochę krótsze (o ok. 10-15% czasu). Powoduje to problem matematyczny, gdyż wskazane wartości należy odpowiednio zaokrąglić w górę, lecz jednocześnie odróżnić od siebie różne wartości o zbliżonych do siebie długościach (np. półnutę od ćwierćnuty z dwiema kropkami).

## Prezentacja graficzna zapisu nutowego

Po ustaleniu wszystkich atrybutów nut, zmian tonacji i metrum, należy odpowiednio przygotować każdą z pięciolinii. Biblioteka *PSAMWPFControlLibrary* umożliwia prezentowanie zapisu nutowego w elemencie graficznym typu *IncipitViewerWPF*. Jego interfejs wymaga podawania po kolei wszystkich oznaczeń, które mają się pokazać na pięciolinii, a zatem nie tylko nut, ale też klucza, oznaczenia metrum czy kresek taktowych. Do tego celu służy przeciążona metoda:

*PSAMWPFControlLibrary*.*IncipitViewerWPF.AddMusicalSymbol(MusicalSymbol symbol)*

Umożliwia dodanie do zapisu dowolnych obiektów, których klasy dziedziczą po klasie *MusicalSymbol.*

*MusicAnalyzer* prezentuje nuty przypisane do osobnych ścieżek dźwiękowych na oddzielnych pięcioliniach. W tym celu, dla każdego kanału należy wskazać w jakim kluczu przedstawić zapis nutowy (wiolinowym czy basowym).



Rysunek Fragment partii tenora z chóralnej aranżacji utworu ludowego "Oberek powiślański"

Powyższy przykład dobrze ilustruje problem z wyborem klucza dla wskazanej melodii. Tessitura głosu tenorowego jest często zbyt wysoka do zapisu w kluczu basowym, natomiast bywa też za nisko, jak na klucz wiolinowy (przykład na rysunku). W obu przypadkach łączy się to z koniecznością dorysowywania dodatkowych linii powyżej bądź poniżej pięciolinii. Prowadzona jest zatem analiza wysokości wszystkich nut należących do każdej ścieżki dźwiękowej, aby określić, który z kluczy pozwala zaprezentować je bardziej przejrzyście. Program zlicza nuty, które występują powyżej i poniżej wartości granicznej, za którą przyjęto dźwięk C4 – dźwięk równie odległy od obu pięciolinii. Jeśli więcej jest dźwięków wyższych lub równych C4, to zapis będzie prowadzony w kluczu wiolinowym, w przeciwnym wypadku - w kluczu basowym.

Po ustaleniu klucza danej pięciolinii, program iteruje po wszystkich nutach należących do wskazanej ścieżki dźwiękowej. Zanim doda ją do kolekcji elementów należących do pięciolinii, każdorazowo sprawdza, czy w momencie pojawienia się kolejnej nuty nie jest także przypisany sygnał zmiany tonacji lub metrum. W takim przypadku, najpierw należy wstawić znaki przy kluczu bądź nowe metrum a dopiero później nutę. Dodatkowo następuje sprawdzanie, czy na ten moment nie przypada koniec taktu. Wówczas należy postawić znak kreski taktowej. Ostatnim elementem, który może poprzedzać nutę jest znak pauzy wynikający z faktu, iż poprzednia nuta skończyła się wcześniej. Jeśli zbiega się to dodatkowo z momentem stawiania kreski taktowej, należy sprawdzić, czy pauza należy jeszcze do poprzedniego taktu, czy już do następnego i umieścić znaki w odpowiedniej kolejności. Program każdorazowo wylicza odpowiednią długość przerwy i przyporządkowuje jej wartość rytmiczną. W programie *MusicAnalyzer* przyjęto uproszczenie, iż wstawiane pauzy będą miały swoje nominalne wartości rytmiczne i nie będą przedłużane.

Gdy na pięciolinii należy jednak umieścić nutę, program uzupełnia jej wcześniej wyliczone atrybuty jeszcze o bezpośrednie cechy widoku. Należy do nich połączenie nut zaczynających się w tym samym momencie i należących do tego samego kanału dźwiękowego w pionowy akord. Tę zależność i jej prezentację przedstawia poniższa przykładowa ilustracja:



Rysunek Trójdźwięk zapisany na pięciolinii, źródło http://www.codeproject.com/KB/miscctrl/psamcontrollibrary/examplechords.png, strona odwiedzona 1.09.2016

Aby nuta była w ten sposób zaprezentowana, musi otrzymać wartość wskaźnika *isChordElement = true*. Konstruowanie akordów odbywa się poprzez porównanie aktualnej nuty z poprzednią. Jeśli obie zaczynają się w tym samym czasie, to należy im ustawić powyższą flagę, w przeciwnym razie przyjmuje ona domyślną wartość *false*.

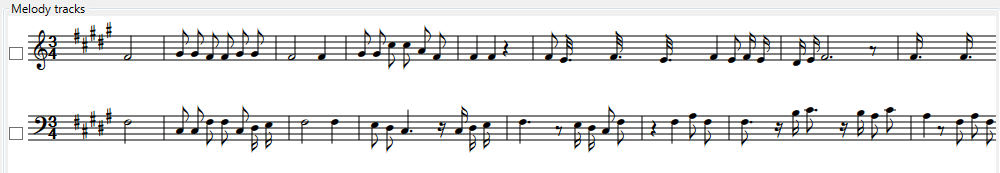
Ostatnim atrybutem nuty pozostającym do ustalenia przed wygenerowaniem jej widoku jest kierunek ogonka nuty, jeśli dana wartość rytmiczna taki posiada. Dotyczy to zatem wszystkich nut za wyjątkiem całej nuty. Możliwe różnice w prezentacji przedstawia poniższy rysunek:



Rysunek Fragment zapisu nutowego z nutami o różnym kierunku

Widać wyraźnie, że nuta o niższej wysokości dźwięku prezentowana w dolnej części pięciolinii jest prezentowana bardziej przystępnie, gdy jej ogonek jest skierowany do góry, a dla wyższych nut analogicznie w dół. *MusicAnalyzer* dla każdej nuty określa ten kierunek w zależności od wysokości nuty i klucza pięciolinii, na jakiej się znajduje. Nuty z ogonkami ‘w dół’ w kluczu wiolinowym to od H4 i wyżej, a niższe mają ogonek ‘do góry’. W kluczu basowym ogonek ‘w dół’ mają nuty od D3 i wyżej, a niższe ‘do góry’. Za przechowanie tej informacji na potrzeby wyświetlenia nuty w obiekcie *IncipitViewerWPF* odpowiada typ wyliczeniowy *PSAMControlLibrary.NoteStemDirection*. Istnieje także powiązanie nut należących do tego samego akordu, które są prezentowane w pionie. W takim przypadku, wiążąca jest pozycja najniższej nuty w akordzie, ona determinuje kierunek ogonka, a pozostałe prezentowane są z tym samym kierunkiem.

Po ustaleniu opisanych wyżej atrybutów związanych z prezentacją zapisu nutowego, obiekt klasy *IncipitViewerWPF* wyświetla wszystkie elementy zgodnie z kolejnością podania ich na wejściu. Biblioteka *PSAMControlLibrary* umożliwia też wiązanie sąsiadujących ze sobą nut poprzez określenie ich atrybutu w postaci typu wyliczeniowego *PSAMControlLibrary*.*NoteBeamType*, lecz w tym projekcie to rozwiązanie nie jest stosowane. Jeśli nuty nie są częścią akordu granego jednocześnie, każdorazowo prezentowane są osobno.



Rysunek Przykładowy zapis nutowy dwóch głosów w programie MusicAnalyzer

## Algorytm tworzenia akompaniamentu

## Harmony Search

## Wyszukanie współbrzmień wejściowych

## Wybór węzłów dla akordów

## Adaptacja algorytmu HS

## Ustalenie atrybutów nut akompaniamentu

## Zapis nutowy i dźwiękowy powstałego akompaniamentu

Po zakończeniu działania algorytmu *Harmony Search*, *MusicAnalyzer* wybiera najlepsze utworzone rozwiązanie i prezentuje jego zapis nutowy jako kolejną ścieżkę dźwiękową widoczną na ekranie głównym aplikacji. Zawarte w niej nuty posiadają analogicznie te same atrybuty, co zdekodowane dźwięki ze ścieżek z pliku wejściowego.

Możliwe jest także wydrukowanie zapisu nutowego akompaniamentu w formacie pionowym i w podziale na strony w zależności od długości zapisu. Opcja ta dostępna jest z menu głównego ekranu (*Menu -> Print accompaniment*). Należało w tym przypadku dokonać pewnych przekształceń, gdyż element graficzny w oknie aplikacji prezentuje nuty w jednym wierszu, co w przypadku prezentacji na wydruku nie byłoby funkcjonalne. W tym celu należało podzielić jedną pięciolinię na wiele systemów tak, aby w pełnej wielkości mogły się one pomieścić na kartce papieru. Aby podzielić element graficzny na wiele części należało wpierw zapisać jego zawartość jako bitmapę a następnie kolejne fragmenty skopiować do nowej bitmapy, której wymiary umożliwiają pomieszczenie wielu systemów z pięcioliniami. Dodatkowo należało zastosować podział na strony w przypadku, gdy zapis miałby zajmować więcej niż 1 stronę. Pomocny okazał się artykuł[[8]](#footnote-8) Benjamina Walkera, w którym opisuje on sposób implementacji podziału wydruku na strony przy konwersji elementów graficznych na bitmapy.

Przy użyciu narzędzi systemowych *Windows,* wydruk można zapisać do pliku *.pdf*. Przykładowy wydruk jest przedstawiony na poniższej ilustracji. Brak bocznych marginesów jest zdeterminowany tym, iż wskazane narzędzie zapisujące plik w formacie *.pdf* umożliwia wydruk na całej stronie, bez marginesów. Przy użyciu fizycznej drukarki, ilość wierszy i ich szerokość będzie dostosowana do możliwości drukarki.



Rysunek Przykładowy wydruk zapisu nutowego akompaniamentu

Program pozwala także zapisać powstałą ścieżkę dźwiękową do pliku *MIDI*. Umożliwia to także odtworzenie melodii bazowej wraz z akompaniamentem.

## Stosowane rozwiązania pomocnicze

W aplikacji *MusicAnalyzer* stosowane były rozwiązania udostępniane przez ogólnodostępne biblioteki działające zgodnie z licencją *Open Source*, dzięki czemu mogą być używane do celów niekomercyjnych. Dołączenie ich referencji jest konieczne do skompilowania projektu.

## Sanford.Multimedia

*Sanford*.*Multimedia* to biblioteka dla platformy .Net udostępniająca klasy i metody ułatwiające odczytywanie i analizę plików dźwiękowych w formacie *Musical Instrument Digital Interface (MIDI)*. Projekt autorstwa Leslie Sanford umożliwia wczytanie pliku wejściowego, odkodowanie zawartych w nim sygnałów oraz odtworzenie zapisanej w nim muzyki. Poniżej przedstawione zostały wybrane klasy, które były używane w projekcie *MusicAnalyzer* i dostarczały funkcjonalności potrzebnych do operacji na pliku *MIDI*:

* **MidiEvent** – odzwierciedla zdarzenia (sygnały) zachodzące w określonych momentach czasu, które zawierają różnorakie informacje związane z daną chwilą; przykładem takich zdarzeń są przykładowo: zmiana tonacji, zmiana tempa, początek trwania określonego dźwięku lub jego koniec;
* **Track** – odpowiada pojedynczej ścieżce dźwiękowej zapisanej w pliku; zawiera w sobie kolekcję obiektów klasy **MidiEvent**, czyli zdarzeń zaistniałych na danej ścieżce;
* **Sequence** – odpowiada całości zawartości pliku MIDI i posiada kolekcję obiektów typu **Track**, przez co umożliwia zapisywanie, odczytywanie i odtwarzanie pliku muzycznego w całości;
* **ChannelMessage** – wiadomość zawarta w zdarzeniu **MidiEvent**, która jest przypisana do konkretnego kanału dźwiękowego (stereo); są to głównie wiadomości o rozpoczęciu (*NoteOn*) lub zakończeniu (*NoteOff*) dźwięku o wskazanej wysokości i głośności;
* **MetaMessage** – wiadomości pozostałych typów, które nie są bezpośrednio związane z danym kanałem, lecz ze ścieżką dźwiękową (np. zmiana tonacji)
* **OutputDevice** – klasa opisująca urządzenie umożliwiające wysyłanie sygnałów MIDI, aby móc odsłuchać utwór muzyczny
* **Sequencer** – umożliwia odtwarzanie sygnałów muzycznych zapisanych w obiekcie **Sequence**; tworzy kolejkę sygnałów i wysyła je poprzez **OutputDevice**

## PSAMControlLibrary

Jest to biblioteka[[9]](#footnote-9) funkcji umożliwiających prezentację graficzną zapisu nutowego w technologii *Windows Forms* na platformie *.Net*. Projekt autorstwa Jacka Salamona posiada też bardziej rozbudowaną wersję komercyjną, lecz używana w tym projekcie biblioteka jest dostępna w ramach licencji *BSD*. Udostępnia ona obiekty odwzorowujące elementy zapisu melodii na pięciolinii oraz umożliwia wygenerowanie widoku zapisu nutowego w formatce w technologii *Windows Forms*. Kilka używanych obiektów oraz typów wyliczeniowych:

* **Clef** – klucz na pięciolinii (basowy lub wiolinowy)
* **Key** – zestaw znaków przy kluczu odpowiadający tonacji utworu (krzyżyki lub bemole)
* **TimeSignature** – oznaczenie metrum (np. ¾)
* **Note** – nuta na pięciolinii o odpowiedniej wysokości i wartości rytmicznej
* **MusicSymbolDuration** – wartość rytmiczna nuty (np. ósemka, ćwierćnuta)
* **BarLine** – kreska taktowa
* **Rest** – pauza odpowiedniej długości (np. pauza ósemkowa, ćwierćnutowa)

## PSAMWPFControlLibrary

Opisywana powyżej biblioteka została stworzona z myślą o interfejsie graficznym w technologii *Windows Forms*. Jako że *MusicAnalyzer* tworzy okno aplikacji dzięki *Windows Presentation Foundation*, konieczne było wykorzystanie biblioteki *PSAMWPFControlLibrary[[10]](#footnote-10)*, która jest przystosowana do budowy interfejsu w tej technologii i daje też dużo szersze możliwości graficzne. Dzięki dobrej kompatybilności *Windows Forms* i *WPF*, można było użyć elementów muzycznych z pierwotnej biblioteki, natomiast konieczne było jedynie osadzenie ich w elemencie typu *IncipitViewerWPF*, który gromadził na pięciolinii dodawane kolejno do niego elementy.

Aby wymienione powyżej znaki pokazywały się prawidłowo, należy w systemie operacyjnym zainstalować czcionkę *Polihymnia* dołączonej do projektu i dystrybuowanej zgodnie z licencją *Sil Open Font Licence*.

# Ocena

## Założenia testów

## Wyniki testów

## Kryteria oceny

## Ocena jakości

# Podsumowanie

# Wykaz rysunków

[Rysunek 1 Schmeta działania Emily Howell, źródło https://www.google.com/patents/US7696426 8](#_Toc460550292)

[Rysunek 2 Schemat działania programu Mezzo, źródło http://www.danielbrownmusic.com/uploads/1/3/2/3/13234393/final\_dissertation\_final\_edit.pdf 11](#_Toc460550293)

[Rysunek 3 Interfejs programu DigiBand, źródło http://www.athtek.com/image/digiband/composition.jpg, strona odwiedzona 1.09.2016 12](#_Toc460550294)

[Rysunek 4 Komunikat o braku plików konfiguracyjnych w MusicAnalyzer 13](#_Toc460550295)

[Rysunek 5 Fragment zapisu nutowego pieśni "Kozak" S. Moniuszki z akompaniamentem pianina (c-moll). 16](#_Toc460550296)

[Rysunek 6 Tonacje i znaki przy kluczu wiolinowym, źródło http://blog.gitarius.pl/kurs\_gitarowy/chro4.png 17](#_Toc460550297)

[Rysunek 7 Podział wartości rytmicznych, źródło http://whittier.mpls.k12.mn.us/uploads/rhythm\_tree\_1.jpg 18](#_Toc460550298)

[Rysunek 8 Fragment partii tenora z chóralnej aranżacji utworu ludowego "Oberek powiślański" 19](#_Toc460550299)

[Rysunek 9 Trójdźwięk zapisany na pięciolinii, źródło http://www.codeproject.com/KB/miscctrl/psamcontrollibrary/examplechords.png, strona odwiedzona 1.09.2016 20](#_Toc460550300)

[Rysunek 10 Fragment zapisu nutowego z nutami o różnym kierunku 21](#_Toc460550301)

[Rysunek 11 Przykładowy zapis nutowy dwóch głosów w programie MusicAnalyzer 21](#_Toc460550302)

# Wykaz tabel

[Tabela 1 Wartości rytmiczne przedłużane, źródło https://musictheorysite.files.wordpress.com/2015/06/dotted-value.png 18](#_Toc460366457)

# Wykaz literatury pomocniczej

**Brak źródeł w bieżącym dokumencie.**

# Odwołania

**Brak źródeł w bieżącym dokumencie.**

– użyć bibteX szukać przez google scholars

BibteX 2 Word

1. http://artsites.ucsc.edu/faculty/cope/Emily-howell.htm, strona odwiedzona dnia 20.08.2016 r. [↑](#footnote-ref-1)
2. „Virtual composer makes beautiful music—and stirs controversy”, Jacqui Cheng, http://arstechnica.com/science/2009/09/virtual-composer-makes-beautiful-musicand-stirs-controversy/, strona odwiedzona dnia 20.08.2016 r. [↑](#footnote-ref-2)
3. https://www.google.com/patents/US7696426, strona odwiedzona dnia 22.08.2016 r. [↑](#footnote-ref-3)
4. http://www.nytimes.com/1997/11/11/science/undiscovered-bach-no-a-computer-wrote-it.html?pagewanted=all, strona odwiedzona dnia 20.08.2016 r. [↑](#footnote-ref-4)
5. http://www.danielbrownmusic.com/uploads/1/3/2/3/13234393/final\_dissertation\_final\_edit.pdf, strona odwiedzona dnia 22.08.2016 r. [↑](#footnote-ref-5)
6. http://www.athtek.com/digiband/music-composition-software.html, strona odwiedzona 1.09.2016 [↑](#footnote-ref-6)
7. http://www.codeproject.com/Articles/6228/C-MIDI-Toolkit, strona odwiedzona dnia 15.05.2016 r. [↑](#footnote-ref-7)
8. http://www.codeproject.com/Articles/339416/Printing-large-WPF-UserControls, strona odwiedzona w dniu 08.09.2016 [↑](#footnote-ref-8)
9. http://www.codeproject.com/Articles/87329/PSAM-Control-Library, strona odwiedzona dnia 22.03.2016 r. [↑](#footnote-ref-9)
10. http://www.codeproject.com/Articles/89582/PSAM-WPF-Control-Library, strona odwiedzona dnia 22.03.2016 [↑](#footnote-ref-10)