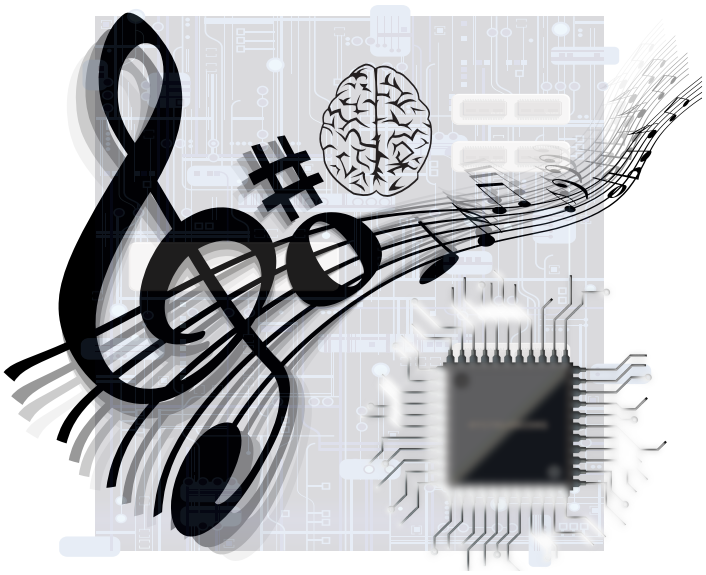
Wydział Informatyki
Zachodniopomorskiego Uniwersytetu
Technologicznego w Szczecinie

Praca doktorska

Łukasz Mazurowski

Model systemu algorytmicznego komponowania muzyki na podstawie wzorców



Promotor rozprawy:
prof. dr hab. inż. Antoni Wiliński

Szczecin, 2017

Spis treści

Charakterystyka pracy . . . 3

Rozdział 1. Komponowanie algorytmiczne muzyki - wprowadzenie i analiza rozwiązań . . . 5

1.1. Wprowadzenie . . . 5
1.2. Metody komputerowego generowania muzyki . . . 15
1.3. Komputerowe systemy generowania muzyki . . . 41
1.4. Sformułowanie problemu badawczego . . . 48

Rozdział 2. Koncepcja systemu algorytmicznego komponowania muzyki . 49

2.1. Proces komponowania . . . 49
2.2. Komponowanie bez definiowania formy generowanego utworu . . . 52
2.2.1. Generowanie wzorców . . . 57
2.2.2. Generowanie kompozycji wyjściowej . . . 63
2.3. Komponowanie z uwzględnieniem opisu formy generowanego utworu . . . 66
2.3.1. Generowanie wzorców . . . 67
2.3.2. Generowanie kompozycji wyjściowej . . . 71
2.4. Instrumentacja i synteza kompozycji . . . 73

Rozdział 3. Implementacja systemu algorytmicznego komponowania muzyki . . . 75

3.1. Komponowanie bez definiowania formy kompozycji . . . 75
3.1.1. Generowanie wzorców muzycznych . . . 78
3.1.2. Generowanie kompozycji muzycznej . . . 82
3.2. Komponowanie z uwzględnieniem opisu formy kompozycji . . . 84
3.2.1. Parametry kontrolujące proces komponowania . . . 85
3.2.2. Generowanie progresji akordów . . . 86
3.2.3. Generowanie melodii i linii basowej . . . 87
3.2.4. Generowanie linii instrumentów perkusyjnych . . . 89
3.2.5. Finalizacja procesu komponowania . . . 92
3.3. Synteza wygenerowanej kompozycji . . . 94

Rozdział 4. Eksperymenty i wyniki badań . . . 97

4.1. Generowanie kompozycji bez opisu formy . . . 97
4.1.1. Modyfikacja parametru długości dźwięków . . . 102
4.1.2. Modyfikacja parametru długości wzorca . . . 103
4.1.3. Modyfikacja parametrów zbioru wygenerowanych i wykorzystanych wzorców muzycznych . . . 105
4.1.4. Modyfikacja struktury tablicy możliwych powtórzeń wzorca . . . 107
4.1.5. Wnioski z eksperymentów . . . 108
4.2. Generowanie kompozycji bazujące na opisie formy . . . 113
4.2.1. Modyfikacja wektora opisującego strukturę kompozycji . . . 116
4.2.2. Modyfikacja wektora skali muzycznej . . . 120

4.2.3. Modyfikacja wektorów lokalizacji dźwięków 124
4.2.4. Modyfikacja wektorów przesunięć dźwięków 126
4.2.5. Modyfikacja wektorów długości dźwięków 129
4.2.6. Modyfikacja liczby i czasu trwania akordów 132
4.2.7. Wnioski z eksperymentów 134

Rozdział 5. Konkluzje oraz perspektywy rozwoju i wdrożeń 136

Publikowanie wyników pracy 138
Perspektywy rozwoju 140
Wnioski końcowe 143

Bibliografia 145

Glosariusz 150

Spis rysunków 153

Spis tablic 156

Dodatek A. Listingi kodów 158

Dodatek B. Zawartość dołączonej płyty CD 191

Charakterystyka pracy

Praca poświęcona jest problematyce generowania za pomocą programów komputerowych utworów muzycznych o charakterze użytkowym. Utwory takie muszą spełniać jednocześnie szereg warunków decydujących o ich wartości użytkowej (pośrednio - cenie), atrakcyjności artystycznej, zmiennej w czasie sile oddziaływania na odbiorców muzyki, adekwatności do kontekstu emisji (radio, sklepy, inne miejsca publiczne itp.) oraz poprawności w aspekcie prawnym.

W tym krótkim opisie zamiaru autora kryje się złożony wielowątkowy problem wymagający z pewnością stosowania metod naukowych i wyrafinowanego warsztatu informatycznego. Od samego początku także oczywiste było stosowanie metod heurystycznych do generowania muzyki. Metody te nie dają oczywistego jednoznacznego "najlepszego" rozwiązania - tu zresztą trudno o ewidentne kryteria jakości muzyki sztucznej. W tej sytuacji problem badawczy jawił się raczej jako poszukujący przybliżonego rozwiązania jednak przy całym szeregu formalnych restrykcji.

Muzyka światowa, prawie niezależnie od kultury a nawet cywilizacji, w której wiekami wyrastała, posiada twarde matematyczne reguły związane z akceptacją przez człowieka np. następstwa dźwięków lub różnicy sąsiadujących częstotliwości. Podejmując się opracowania heurystycznych algorytmów do kreowania muzyki z użyciem komputera należy przewidywać konieczność uwzględniania tych ograniczeń formalnych, ale także przewidywać dozę heurystycznej swobody w komponowaniu muzyki. Winna być ona bowiem taką, która będzie podobała się odbiorcy (nie wszystkim, gdyż takich utworów nie ma), ale przede wszystkim nie drażniła, nie naruszała wzorców estetycznych muzyki.

Takie oto wyzwanie ulokować można z naukowego punktu widzenia przede wszystkim w informatyce technicznej. Powstające tu programy komputerowe wymagały będą znajomości warsztatu programistycznego ale również metod matematycznych z obszaru sztucznej inteligencji. Być może konieczną okaże się taka modyfikacja metod stricte informatycznych do realizacji postawionego zadania, że powstaną nowe, zmienione metody uniwersalne, zaadaptowane na potrzeby tej pracy. Z metodologicznego punktu widzenia niniejsza praca ma charakter metodyczny. Dominującym jej wyzwaniem będzie poszukiwanie informatycznych metod twórczego oddziaływania - w szerokiej perspektywie oddziaływania na świat zewnętrzny za pomocą sztucznej muzyki.

Dzieląc świat nauki na dziedziny twórcze i dziedziny poznawcze, informatyka techniczna z pewnością należy do tych pierwszych. Te drugie chętniej posługują się modelem jako przybliżonym obrazem świata, doskonalonym po to, by go lepiej poznać, przewidzieć i wykorzystywać. To drugie podejście z pewnością pojawi się w pracy jako narzędzie doskonalenia metod informatyki.

Mylący może być także przymiotnik “techniczna” w kontekście pracy dotyczącej muzyki. Jednak informatyka techniczna, w odróżnieniu od matematycznej, to właściwie oddzielna dyscyplina naukowa o charakterze twórczym, czynnie (za pomocą algorytmów) oddziaływająca na świat.

Ta praca, w swym założeniu, będzie przykładem niemal wzorowego oddziaływania rezultatów algorytmicznego przetwarzania muzyki na ludzi. Owe rezultaty to kompozycje muzyczne utworzone za pomocą autorskich metod tworzenia muzyki generatywnej.

Rozdział 1

Komponowanie algorytmiczne muzyki - wprowadzenie i analiza rozwiązań

W niniejszym rozdziale dokonano teoretycznego wprowadzenia do zagadnienia algorytmicznego komponowania muzyki oraz zanalizowano historycznie istotne rozwiązania z tej dziedziny, ujęte w literaturze światowej. Dla potrzeby prawidłowego zrozumienia przez Czytelnika problematyki zawartej w pracy, autor dokonał przeglądu głównych zagadnień związanych z teorią muzyki, w wymiarze wystarczającym do przyswojenia treści zawartych w pracy. Definicje przytoczonych pojęć znajdują się w rozdziale "Glosariusz".

W literaturze z dziedziny opisywanej przez autora wykorzystywane jest pojęcie muzyki generatywnej, będące podzbiorem szerszego pojęcia sztuki generatywnej. Muzyka generatywna odnosi się do muzyki utworzonej przez system autonomiczny. Pojęcie autonomiczności systemu w kontekście algorytmicznego komponowania muzyki wiąże się ze sztucznym systemem potrafiącym niezależnie tworzyć sztukę (np. muzykę) z wykorzystaniem algorytmu determinującego sposób generowania.

Boden i Edmonds zauważyli, że "*pojęcia sztuka generatywna i sztuka komputerowa są stosowane zamiennie od bardzo dawna*" [25]. Pojęcie muzyki generatywnej zostało zdefiniowane przez Briana Eno, promującego i wykorzystującego metody sztuki generatywnej w swojej pracy [21]. W roku 2003 Galanter rozszerzył pojęcie "sztuka generatywna" podając następującą jego definicję:

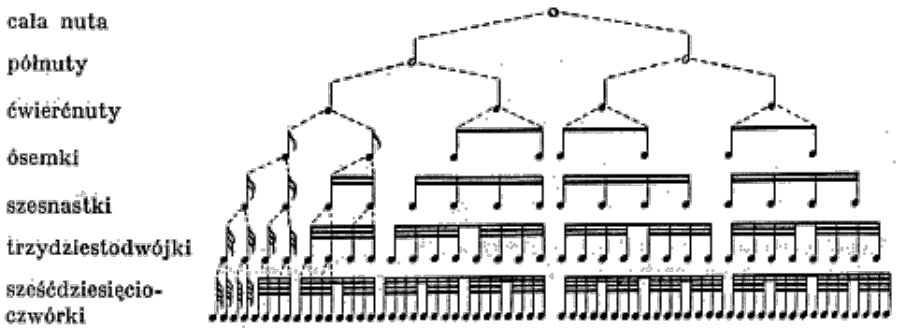
Sztuka generatywna odnosi się do każdej czynności, w której artysta tworzy proces, taki jak zbiór reguł języka naturalnego, program komputerowy, maszynę lub inną inwencję określoną proceduralnie, która jest następnie wprowadzona w ruch z pewną dozą autonomii przyczyniającą się lub będąca wynikiem kompletnej pracy twórczej. [78]

Podsumowaniem dokonanego przeglądu rozwiązań i odniesienia do problemu badań jest sformułowanie celu niniejszej pracy oraz hipotezy badawczej.

1.1. Wprowadzenie

Przedstawiona w pracy problematyka, choć umieszczona w dziedzinie informatyki, osadzona jest również w obrębie domeny muzyki, z której czerpie pojęcia, definicje i reguły. Autor pracy zakłada, że Czytelnik wywodzi się ze środowiska informatycznego i nie musi posiadać wiedzy i umiejętności związanych z muzyczną częścią dysertacji. Biorąc pod uwagę powyższe stwierdzenie, w dalszej części tego podrozdziału zaprezentowane zostały podstawy teorii muzyki, niezbędne do prawidłowej analizy badań i rezultatów uzyskanych przez autora.

Zanim zostaną omówione podstawowe definicje występujące w dziedzinie muzyki, należy w tym miejscu przytoczyć pięć składowych dźwięku, mające wpływ na jego charakter i odbiór przez słuchacza, tj.: częstotliwość, amplituda, tembr, czas trwania oraz obwiednia.



Rys. 1.1. Regularny podział wartości rytmicznych. Źródło [55].

Częstotliwość dźwięku wyznacza jego wysokość. Przedstawia się ją jako liczbę wibracji przypadającą na sekundę, która jest odbierana przez narząd słuchu człowieka jako dźwięk o określonej (przez liczbę tych drgań) wysokości. W muzyce elektronicznej częstotliwość dźwięku może być utożsamiana z liczbą wibracji na sekundę membrany źródła dźwięku (np. głośnika), z którego wydobywa się muzyka. [91]

Amplituda określa głośność dźwięku i jest przekazywana przez membranę głośnika jako różnica pomiędzy neutralnym położeniem membrany a jej szczytowym położeniem podczas ruchu do przodu, do tyłu w czasie emitowania dźwięku. W instrumentach akustycznych amplituda jest kontrolowana przez muzyka poprzez łagodne bądź mocne - uderzenie w klawisz, dmuchnięcie w róg, prowadzenie smyczka, itd. [91]

Tembr (barwa) jest naturą i jakością dźwięku muzycznego, czasami nazywany bywa kolorem dźwięku. Ucho ludzkie dźwięki możliwości odbioru barwy dźwięku, umożliwia rozróżnienie dźwięku o tej samej wysokości, odgrywanego przez różne instrumenty. Fale dźwięków muzycznych są złożone i zawierają więcej niż jedną częstotliwość czy ton podstawowy. Te dodatkowe fale występujące w strukturze dźwięku nazywane są harmonicznymi. Jeżeli jedna z fal w strukturze dźwięku jest falą dominującą, to człowiek jest w stanie rozróżnić wysokość tego dźwięku. Bywa jednak czasem tak, że dźwięki emitowane przez instrument nie posiadają określonej wysokości (są to niektóre instrumenty należące do grupy instrumentów perkusyjnych). [91]

Czas trwania jest to długość dźwięku wyrażona w jednostce czasu. Instrumenty akustyczne posiadają ograniczone możliwości podtrzymywania dźwięku. Długość dźwięku jest blisko związana z jego obwiednią. [91]

Podstawowe elementy notacji

Graficzną reprezentacją dźwięku w zapisie muzycznym jest nuta, czyli podstawowy element notacji muzycznej, której wygląd w zasadzie został ukształtowany w XVII – XVIII wieku. Nuta określa wysokość i czas trwania dźwięku muzycznego. Jako wysokość dźwięku należy rozumieć częstotliwość z jaką występuje. Wysokość dźwięku zależy od położenia nuty na strukturze zwanej pięciolinią. Im wyżej położona jest nuta, tym wyższy dźwięk symbolizuje. Drugą cechą dźwięku muzycznego jest wartość rytmiczna, określająca czas trwania dźwięku. Regularny podział wartości muzycznych przedstawiono na rys. 1.1.

Nazwa interwału	Liczba stopni	Liczba półtonów
pryma	1	0
sekunda	2	1 lub 2
tercja	3	3 lub 4
kwarta	4	5
tryton	4 lub 5	6
kwinta	5	7
seksta	6	8 lub 9
septyma	7	10 lub 11
oktawa	8	12
nona	9	13 lub 14
decyma	10	15 lub 16
undecyma	11	17
oktawa i tryton	11 lub 12	18
duodecyma	12	19
tercdecyma	13	20 lub 21
kwartdecyma	14	22 lub 23
kwintdecyma	15	24

Tab. 1.1. Podstawowe nazwy interwałów wraz z przypadającymi dla nich: liczbą stopni i półtonów. Opracowanie własne.

Według przedstawionego rys. 1.1, cała nuta dzieli się na dwie pół nuty, następnie półnuta dzieli się na dwie ćwierćnuty, itd. Jest to tzw. regularny podział rytmiczny (podział dwójkowy) wykorzystywany we współczesnej pisowni muzycznej. Obok tego podziału istnieje również podział trójkowy (nieregularny), który polega na dzieleniu większej wartości rytmicznej na trzy mniejsze.

Przez akcent należy rozumieć podkreślenie dźwięku o określonym położeniu w stosunku do dźwięków sąsiednich. Dźwięki akcentowane i nieakcentowane grupuje się w struktury podziału utworu muzycznego zwane *taktami*. Tekst muzyczny należy rozumieć jako graficzny zapis utworu muzycznego na pięciolinii. Wartość sumy nut i pauz określa tzw. oznaczenie taktowe, zapisywane na pięciolinii w postaci ułamka zwykłego. Licznik oznaczenia określa liczbę jednostek miarowych w takcie, oznaczonych przez liczbę w mianowniku. Dla przykładu oznaczenie taktowe $\frac{3}{4}$ wskazuje, że w jednym takcie mają być umieszczone trzy ćwierćnuty bądź liczba dźwięków, których suma wartości rytmicznych da wartość trzech ćwierćnut. Pogrupowanie dźwięków w takty wprowadza uporządkowanie informacji zawartej w kompozycji oraz upraszcza analizę jej zawartości.

W ujęciu muzycznym podstawowym zestawem dźwięków połączonych w grupę jest skala muzyczna, która tworzy szereg dźwięków ułożonych według stałego schematu [55]. Najwyższy dźwięk (inaczej stopień) danej skali muzycznej jest powtórzeniem najniższego dźwięku w *interwale* oktawy. Nazwy interwałów określają liczbę stopni zawartych między dźwiękami tworzącymi dany interwał, wraz z tymi dźwiękami. W tab. 1.1 przedstawione zostały nazwy interwałów z odpowiadającymi im liczbami stopni i półtonów, określających odległość pomiędzy dźwiękami.

W systemie modalnym, który jest jednym z najstarszych systemów klasyfikacji dźwięków, zakres dźwięków muzycznych został podzielony na siedem skal (rys.

Rys. 1.2. Skale systemu modalnego. Opracowanie własne.

1.2): jońską, dorycką, frygijską, lidyjską, miksolidyjską, eolską oraz lokrycką. Kolejne stopnie skali oznaczone zostały za pomocą cyfr rzymskich I - VII, natomiast cyframi arabskimi przedstawiono odległości pomiędzy kolejnymi stopniami w tonach.

Każda z wymienionych na rys. 1.2 skal charakteryzuje się swoim specyficznym i rozpoznawalnym brzmieniem. Skala durowa (majorowa) jest jedną z podstawowych skal muzycznych wykorzystywanych w muzyce popularnej. Charakterystyczny rozkład interwałów skali durowej sprawia, że ma ona bardzo mocny charakter brzmieniowy. Obok skali durowej popularną skalą systemu modalnego jest skala molowa (minorowa). Charakter brzmieniowy skali molowej jest zdecydowanie miękki, co znacząco odróżnia ją od skali majorowej. Wariacje skali durowej (jońskiej) i molowej (eolskiej) przedstawione na rys. 1.2 wprowadzają dodatkowe urozmaicenia brzmieniowe poprzez zmiany interwałów kolejnych stopni skali.

Zaprezentowane w tej części pracy elementarne informacje na temat graficznej notacji dźwięku muzycznego, właściwości tego dźwięku oraz relacji łączących go z innymi dźwiękami w ramach skali, pozwalają na dalsze omówienie podstawowych elementów składających się na kompozycję muzyczną.

Elementy dzieła muzycznego

Na każde dzieło muzyczne, bez względu na jego złożoność budowy, składa się szereg elementów muzycznych: *rytm*, *metrum*, *tempo*, *melodia*, *harmonia*, *dynamika*, *agogika*, *artykulacja*, *frazowanie*, *kolorystyka dźwiękowa*, *struktura formalna* [55].

Rytm jest w muzyce czynnikiem nieodzownym, przejawiającym się w zorganizowanym ruchu dźwięków, z który każdy posiada określony czas trwania. O charakterze utworu muzycznego decyduje schemat rytmiczny określający formę rytmu. Nie wszystkie dźwięki muzyczne uporządkowane w schemacie posiadają jednakową wagę, jednakowy ciężar gatunkowy. Czynnikiem regulującym ten aspekt jest metrum muzyczne, porządkujące przebiegi za pomocą *akcentów metrycznych* przypadających na niektóre dźwięki w podobny sposób, jak to ma miejsce w poezji wierszowanej [55].

Znaczący wpływ na ukształtowanie przebiegów rytmiczno-metrycznych posiada tempo. Występujące w utworze muzycznym wartości rytmiczne wskazuję jedynie wzajemne stosunki czasu trwania poszczególnych wartości metrycznych. Dla przykładu półnuta trwa dwukrotnie krócej od całej nuty, ćwierćnuta - dwukrotnie krócej od półnuty itd. Tempo nie zmienia tych stosunków, określa jednak dokładniej czas trwania poszczególnych dźwięków. Tempo wpływa również na rozmieszczenie i rolę poszczególnych akcentów metrycznych. Kiedy utwór muzyczny jest wykonywany w tempie wolnym znacznie więcej dźwięków muzycznych otrzymują akcenty metryczne aniżeli przy wykonywaniu tego samego utworu w tempie szybkim [55].

Z racji niezmiernej wagi rytmu w muzyce, w utworach spotkać można nawet dość długie odcinki dzieła przeznaczone do wykonania wyłącznie przez instrumenty perkusyjne. Należy również dodać, że w muzyce ludów pierwotnych rytm jest czynnikiem dominującym. Trudno jednak sobie wyobrazić kompozycję muzyczną bez elementów melodii, tworzącej pewną spójną logiczną całość następstw dźwięków różnej wysokości o zorganizowanej strukturze rytmicznej. Zasadniczo można wyróżnić dwie grupy melodii, pod względem wykorzystywanych w nich dźwięków muzycznych: *melodie diatoniczne* oraz *melodie chromatycznie rozszerzone* [55].

Innym podziałem melodii jest rozróżnienie na melodie wokalne oraz melodie instrumentalne. Melodia wokalna jest przeznaczona do wykonania przez głos ludzki, a przez ograniczone możliwości głosu ludzkiego, nie może być zbyt ruchliwa i posiada ograniczoną skalę. Melodia instrumentalna jest natomiast dostosowana do skali i możliwości instrumentu muzycznego, i może być dużo bardziej ruchliwsza od melodii wokalnej, może zawierać gwałtowne skoki, itp. [55]

Melodie można również rozróżnić pod względem ich złożoności. W tym podziale istnieją melodie proste, nieornamentalne oraz takie, w których na tle właściwej melodii występuje szereg dźwięków ozdabiających (melodia ornamentalna, z łac. ornamentum = ozdoba) [55]. Sekwencja dźwięków w melodii może rozwijać się pochodami małych interwałów (prym i sekund), może rozwijać się na dźwiękach akordowych, postępować skokami itp., a także tworzyć w swym przebiegu linię poziomą, linię wznoszącą się lub opadającą i wreszcie linię falistą.

Melodia i rytm są głównymi czynnikami występującymi w dziele muzycznym. Obok nich na ogół spotkać można czynnik harmoniczny, wyrażający się współbrzmieniami dźwięków, czyli harmonią. Z chwilą równoczesnego prowadzenia dwóch głosów mamy do czynienia z harmonią. W kompozycji czynnik harmoniczny może pojawić się pod dwiema różnymi postaciami: *polifonii* i *homofonii* [55].

W polifonii wszystkie prowadzone głosy dają w wyniku swego równoczesnego przebiegu współbrzmienia układające się w możliwe do określenia akordy. Polifonia występuje w wielu formach, jednak najczęściej wykorzystywane w utworach muzycznych formy polifoniczne to imitacja, kanon, fuga, inwencja, itp. [55]

W homofonii wszystkie głosy są całkowicie podporządkowane jednej melodii. Melodia homofoniczna najczęściej znajduje się w głosie górnym, a pozostałe głosy

tworzą tzw. podparcie harmoniczne zwane akompaniamentem. Głosy akompaniujące nie posiadają samodzielnych linii melodycznych, jak to ma miejsce w polifonii. Kompozytor podczas tworzenia dzieła muzycznego na ogół wykorzystuje za równo homofonię jak i polifonię [55].

Forma kompozycji muzycznej

Poznanie wprowadzonych wcześniej pojęć daje możliwość spojrzenia na strukturę, która przy wykorzystaniu odpowiednich reguł matematycznych szereguje dźwięki w czasie tworząc kompozycje. Jak podaje Shoenberg w [16] "*najmniejszą jednostką strukturalną jest fraza, rodzaj molekuły muzycznej składającej się z określonej liczby zintegrowanych zdarzeń muzycznych, posiadających pewną kompletność, dobrze przystosowanych do kombinacji z innymi podobnymi jednostkami*". Istotnym elementem frazy są jej elementy wzbogacające tzw. motywy. Melodia, która w kompozycji jest strukturą wiodącą, posiada motyw główny, wokół którego tworzone są wariacje (urozmaicenia) oraz poboczne figury muzyczne.

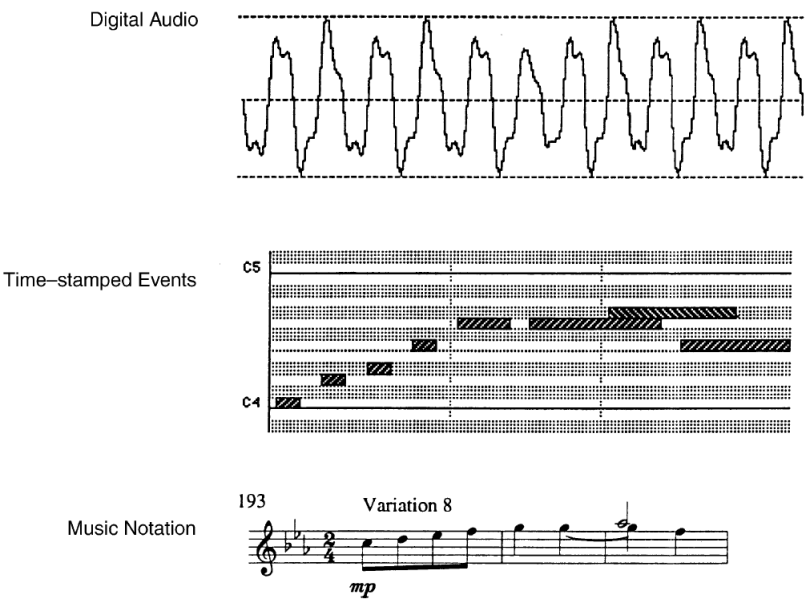
Istotną charakterystyką utworu jest powtarzalność jego elementów (w tym melodii, której wariacje motywów powtarzane są w różnych częściach kompozycji). Ta cecha kompozycji związana z jego formą, opisującą ogólny przebieg kompozycji w czasie. Shoenberg podaje w [16], że "*przytłaczającą większością proporcji form muzycznych jest struktura skomponowana z trzech części. Trzecia część jest czasem czystym powtórzeniem pierwszej, ale częściej jest bardziej lub mniej zmodyfikowanym powtórzeniem. Druga część jest zorganizowana jako kontrast*". Części formy (a więc i części kompozycji) oznacza się poprzez litery alfabetu. Dla przykładu przytoczony przez Shoenberga opis formy można zapisać jako ABA, gdzie A - jest to pierwsza i trzecia część kompozycji (początek i zakończenie), natomiast B stanowi kontrastowy środek.

Forma stanowi skrócony opis kompozycji - nie przedstawia szczegółów związanych np. z szeregiem czasowym dźwięków tworzących poszczególne części kompozycji. W kolejnej części tego rozdziału przedstawiono popularne formaty reprezentacji kompozycji.

Formaty reprezentacji utworu muzycznego

Formy reprezentacji utworu muzycznego związane są ze sposobem (formatem) zapisu informacji muzycznych. Jak podano w [26], w generalnej klasyfikacji form zapisu muzyki możemy wyróżnić trzy podstawowe reprezentacje: postać cyfrowego sygnału audio (ang. *digital audio*), notację muzyczną (ang. *music notation*) oraz zapis zdarzeń datowanych (ang. *time-stamped events*). Przedstawioną klasyfikację ilustruje rys. 1.3.

Strukturę najniższej warstwy informacji stanowi cyfrowy zapis muzyki (*digital audio*). W procesie pozyskiwania informacji muzycznej (ang. *music IR, music information retrieval*) ta forma reprezentacji jest najtrudniejsza pod względem ekstrakcji wiedzy (co zostanie opisane w dalszej części pracy). Notacja muzyczna (*music notation*) to graficzna reprezentacja kompozycji, zawierająca szeregi nut i definiująca sposób odtworzenia utworu (w hierarchii opisu informacji muzycznej stanowi najwyższą warstwę opisu). Kolejną formą reprezentacji jest opis zdarzeń datowanych (*time-stamped events*), który stanowi warstwę pośrednią pomiędzy cyfrowym zapisem muzyki a jej reprezentacją nutową. W opisie zdarzeń datowanych stosuje



Rys. 1.3. Podstawowe reprezentacje muzyki. Źródło: [26].

się m.in. tzw. wykresy pianolowe, które autor wykorzystał w dalszej części pracy poświęconej opisowi metod automatycznego komponowania utworu muzycznego.

Charakterystykę podanych form reprezentacji muzyki przedstawiono w tab. 1.2. Jak opisano w [26] określenia "Konwersja do lewej" i "Konwersja do prawej" oznaczają trudność automatycznej konwersji z danej reprezentacji w inną. Przy dużej wyrazistości formy muzycznej (w zapisie nutowym) dysponujemy kompletną wiedzą o utworze (np. określone jest tempo jego odtwarzania czy szereg czasowy wszystkich dźwięków występujących w kompozycji). Notacja muzyczna jest wyłącznie opisem sposobu odtworzenia danej kompozycji - nie zawiera informacji o instrumentach, które biorą udział w odgrywaniu jej poszczególnych elementów.

W procesie przetwarzania muzycznej informacji często pomocnym w ukazaniu stopnia trudności pozyskania informacji okazuje się porównanie reprezentacji muzyki do form reprezentacji tekstu. Podobieństwa w tym zakresie pokazano w tab. 1.3.

Dobór metody pozyskiwania informacji muzycznej jest ściśle powiązany z jej formą. W literaturze traktującej o metodach MIR (*music information retrieval*) wyraźnie rozgranicza się problem pozyskiwania od percepcji pozyskanych informacji muzycznych przez człowieka. Odbiór kompozycji muzycznej ma charakter mniej lub bardziej indywidualny i zależy od predyspozycji słuchacza. W aspekcie przetwarzania i odbioru muzyki ważnym elementem jest rozgraniczenie pomiędzy monofonią a polifonią.

Monofonia jest określeniem determinującym występowanie w jednej chwili czasowej tylko jednego dźwięku. Przykładem monofoniczności może być melodia główna kompozycji śpiewana przez jednego artystę. Nakładanie się wielu dźwięków w czasie jest domeną polifonii i to właśnie ona występuje w niemalże każdym współczesnym utworze muzyki rozrywkowej. Występowanie polifonii niesie za sobą stosowanie bardzo złożonych metod przetwarzania informacji w sytuacji, gdy struktura opisu muzyki jest minimalna (cyfrowe audio zawierające utwór muzyczny) oraz względnie

Reprezentacja	Cyfrowe audio	Zdarzenia datowane	Notacja symboliczna
Przykłady	Audio CD, plik MP3, plik WAV	plik MIDI	zapis nutowy
Jednostka opisu	próbka sygnału	zdarzenie, komunikat	nuta, klucz wiolinowy, tekst piosenki
Wyrazistość struktury	znikoma	mała (częściowa informacja muzyczna)	duża (całkowita wiedza o utworze)
Konwersja do lewej	-	łatwa	łatwa
Konwersja do prawej	dla jednego dźwięku jednocześnie: bardzo łatwa; dla dwóch dźwięków jednocześnie: trudna; dla większej liczby dźwięków: bardzo trudna	dosyć trudna	-
Idealna do	muzyki, odgłosów zwierząt, efektów dźwiękowych, mowy	muzyki	muzyki

Tab. 1.2. Charakterystyka podstawowowych reprezentacji muzyki (na podstawie [26]). Opracowanie własne.

Złożoność struktury	minimalna	średnia	maksymalna
Reprezentacja muzyczna	cyfrowe audio (CD, MP3)	zdarzenia datowane (plik MIDI)	notacja muzyczna (nuty)
Reprezentacja tekstu	audio (mowa)	tekst nieformatowany	tekst formatowany (HTML)

Tab. 1.3. Reprezentacje tekstu w porównaniu z reprezentacją muzyki (na podstawie [26]). Opracowanie własne.

proste mechanizmy pozyskiwania informacji ze struktur bardzo złożonych (notacja muzyczna lub zdarzenia datowane w postaci plików *MIDI*).

Na potrzeby autorskich badań opisanych w tej pracy jako formę reprezentacji utworu muzycznego zastosowano strukturę zaprezentowaną przez Eerole i Toiviainena w [48]. Przykładową macierz o rozmiarze [8x7], opisującą oktawę od dźwięku $C4$ (60) do $C5$ (72), przedstawiono w tab. 1.4.

Onset (bpm)	Dur. (bpm)	MIDI ch.	Pitch	Vol.	Onset (sec.)	Dur. (sec.)
0	0.4167	1.0000	60.0000	100.0000	0	0.2500
0.4167	0.4167	1.0000	62.0000	100.0000	0.2500	0.2500
0.8333	0.4167	1.0000	64.0000	100.0000	0.5000	0.2500
1.2500	0.4167	1.0000	65.0000	100.0000	0.7500	0.2500
1.6667	0.4167	1.0000	67.0000	100.0000	1.0000	0.2500
2.0833	0.4167	1.0000	69.0000	100.0000	1.2500	0.2500
2.5000	0.4167	1.0000	71.0000	100.0000	1.5000	0.2500
2.9167	0.4167	1.0000	72.0000	100.0000	1.7500	0.2500

Tab. 1.4. Przykładowa macierz nut w środowisku Matlab. Opracowanie własne.

bajt																
000	4D	54	68	64	00	00	00	06	00	01	00	01	00	78	4D	54
010	72	6B	00	00	00	53	00	FF	58	04	04	02	20	08	00	FF
020	51	03	09	27	C0	00	90	3C	64	31	80	3C	64	00	90	3E
030	64	31	80	3E	64	01	90	40	64	31	80	40	64	01	90	41
040	64	31	80	41	64	01	90	43	64	31	80	43	64	01	90	45
050	64	31	80	45	64	01	90	47	64	31	80	47	64	01	90	48
060	64	31	80	48	64	00	FF	2F	00							

Tab. 1.5. Heksadecymalna postać pliku *MIDI* wygenerowanego na podstawie macierzy nut z rys. 1.4. Opracowanie własne.

Każdy wiersz macierzy reprezentuje sparametryzowaną nutę (dźwięk muzyczny). Kolejne kolumny zawierają wartości parametrów danego dźwięku. W pierwszej kolumnie znajdują się wartości rozpoczęcia dźwięku w jednostkach bpm, związanych z tempem odgrywania utworu. Druga kolumna zawiera wartości długości dźwięków w jednostkach bpm. W kolumnie trzeciej występują numery wykorzystanego przez dany dźwięk kanału *MIDI* (standard *MIDI* zakłada jednoczesne wykorzystanie 16 kanałów, przy czym w przytoczonym przykładzie z tab. 1.4 wykorzystywany jest jedynie kanał nr 1). Kolumna czwarta zawiera wartości wysokości dźwięków zapisane w postaci liczb *MIDI*, które odpowiadają poszczególnym częstotliwościom dźwięków wykorzystywanych w muzyce (ich zakres jest ograniczony od 0 do 127). W kolumnie piątej znajdują się wartości głośności dźwięków zawarte w przedziale od 0 (cisza) do 127 (najgłośniej). Kolumny szósta i siódma zawierają wartości przedstawione w kolumnach pierwszej i drugiej wyrażone w sekundach.

Macierz nut przedstawia reprezentację pewnej sekwencji komend *MIDI*, która w przyjętym podejściu analizuje jedynie ograniczony zbiór komunikatów *MIDI* przechowywanych w postaci pliku *MIDI*. Do przedstawienia sposobu interpretacji macierzy nut, w tab. 1.5 przedstawiony został bajtowy zapis pliku *MIDI* wygenerowanego na podstawie macierzy z tab. 1.4.

Na podstawie dokumentacji technicznej standardu *MIDI* File [3] oraz [2] została wykonana poniższa analiza pliku *MIDI* przedstawionego w tabeli 1.5. Cztery pierwsze bajty: 4D 54 68 64, oznaczają początek pliku *MIDI* ("MThd"). Następnie cztery kolejne bajty: 00 00 00 06 wyznaczają rozmiar nagłówka pliki *MIDI* wyrażonego w bajtach (dla plików *MIDI* standardowa wartość wynosi 6). Następne dwa bajty: 00

Delta Time	Event Type Value	MIDI Channel	Param. 1	Param. 2
zmienna długość	4 bity	4 bity	1 bajt	1 bajt

Tab. 1.6. Format kanałowego komunikatu $MIDI$ (na podstawie [2]). Opracowanie własne.

01 wyznaczają format pliku *MIDI*, który w opisywanym przypadku oznacza typ 1 - plik może zawierać wiele sekwencji zapisanych w utworze pod postacią wielu ścieżek [3]. Kolejne dwa bajty: 00 01 oznaczają liczbę ścieżek zdefiniowanych w pliku *MIDI*. Następne dwa bajty: 00 78 wyznaczają liczbę uderzeń przypadającą na ćwierć nutę, która w opisywanym przypadku wynosi 120 ($7 * 16 + 8 = 120$). Są to ostanie dwa bajty nagłówka pliku *MIDI*.

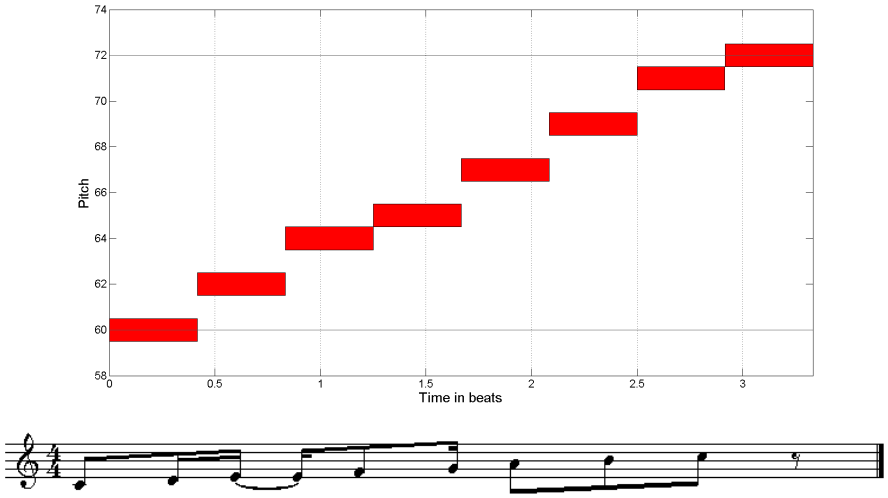
Sekwencja bajtów 4D 54 72 6B wyznaczają początek ścieżki *MIDI*. Następne cztery bajty: 00 00 00 53 wyznaczają liczbę bajtów przypadającą na zdefiniowaną ścieżkę. Komunikaty *MIDI* umieszczone są w pliku *MIDI* razem z tzw. wartością *delta time*, która wyraża ilość czasu, która musi minąć aby wywołany był kolejny komunikat. Tak więc następny bajt 00 oznacza, że komunikat występujący po tej wartości delta time będzie wywołany natychmiast. Komunikatem występującym w opisywanym pliku jest meta komunikat *Time Signature* z sekwencją bajtów FF 58 04 04 02 20 08, przy czym FF 58 04 wyznacza typ komunikaty, a kolejne bajty 04 02 20 08 oznaczają: 04 - cztery uderzenia na takt, 02 - 2^2=4 jednostką wyznaczającą uderzenie jest ćwierćnuta, 20 - 32 - liczba uderzeń zegara przypadająca na ćwierćnutę, 08 - 8 wartości nut o długości $\frac{1}{32}$ przypada na jedną ćwierćnutę (uderzenie). Kolejnym meta komunikatem występującym w analizowanym pliku jest *Set Tempo* (FF 51 03), po którym występują 3 bajty: 09 27 C0, wyznaczające wartość tempa w mikrosekundach przypadających na ćwierćnutę ($9 * 16^4 + 2 * 16^3 + 7 * 16^2 + 12 * 16 = 600000$ mikrosekund).

W dalszej części analizowanego pliku *MIDI* znajduje się sekcja komunikatów *MIDI* należących do grupy komunikatów kanałowych (*MIDI Channel Events*, [2]). Komunikaty kanałowe posiadają strukturę przedstawioną w tab. 1.6.

Prezentowana w tab. 1.5 struktura zwiera jedynie komunikaty *Note On* (4 bity wyznaczające wartość 9) oraz *Note Off* (4 bity wyznaczające wartość 8). Komunikaty są wywoływane dla kanału 0 *MIDI*, przez co w analizowanej strukturze często pojawiają się bajty 90 (*Note On*, kanał 0) i 80 (*Note Off*, kanał 0). Pierwszym komunikatem kanałowym występującym w strukturze 1.5 jest komunikat *Note On* o sekwencji bajtów 00 90 3C 64, z wartością *delta time* równą 00 oraz parametrami 3C (wysokość nuty, $3*16+12 = 60$) i 64 (dynamika nuty, $6*16+4 = 100$). Następujący po nim komunikat *Note Off* posiada strukturę 31 80 3C 64, w której *delta time* wynosi $3 * 16 + 1 = 49$ oraz parametry wysokości i dynamiki nuty posiadają wartości takie same, jak w przypadku poprzednio opisywanego komunikatu *Note On* (3C, 64). Analizując przykład z tab. 1.5 można zauważyć naprzemienne rozmieszczenie komunikatów *Note On* (90) i *Note Off* (80) w kanale 0, z różnymi wartościami *delta time* (00, 01 i 31) oraz kolejnymi wartościami wysokości dźwięków: $3C = 60(C4)$, $3E = 62(D4)$, $40 = 64(E4)$, $41 = 65(F4)$, $43 = 67(G4)$, $45 = 69(A4)$, $47 = 71(B4)$ oraz $48-72(C5)$. Opisywaną strukturę kończy meta komunikat *End of Track Marker* (sekwencja bajtów 00 FF 2F 00).

Wizualizacją macierzy z tab. 1.4 jest pianolowy wykres przebiegu występujących w niej dźwięków muzycznych przedstawiony na rys. 1.4. Na osi pionowej

umieszczono nazwy dźwięków odpowiadające poszczególnym wartościom wysokości dźwięku wg standardu *MIDI*. Na osi poziomej znajdują się przedziały czasowe w jednostkach bpm. Prostokąty występujące na wykresie symbolizują dźwięki muzyczne, a długość tych prostokątów wyznacza czas trwania (wybrzmiewania) dźwięku.



Rys. 1.4. Pianolowy wykres (góra) oraz zapis nutowy (dół) macierzy z tab. 1.4. Opracowanie własne.

W zaprezentowanym w tej części pracy krótkim wprowadzeniu, autor poruszył teoretyczne aspekty muzycznego i informatycznego charakteru dziedziny analizowanego problemu badawczego. Przedstawione zostały podstawowe definicje wykorzystywanych w pracy pojęć muzycznych, a także opis formy reprezentacji danych przetwarzanych i uzyskiwanych w ramach modelowanego systemu algorytmicznego komponowania muzyki. W kolejnej części tego rozdziału zostaną przedstawione wybrane, najczęściej wykorzystywane metody komputerowego generowania muzyki w kontekście algorytmicznego komponowania utworu muzycznego.

1.2. Metody komputerowego generowania muzyki

W niniejszym rozdziale zaprezentowano metody algorytmicznego komponowania muzyki powszechnie występujące w literaturze światowej. Autor przedstawił tu własne implementacje wybranych metod w środowisku Matlab, a do wizualizacji wyników ich działania wykorzystał wykresy pianolowe. W zasadzie każda z implementowanych tu metod może być przedmiotem odrębnych badań i udoskonaleń, stąd przykłady tu prezentowane mają wyłącznie charakter informacyjny.

Geneza algorytmicznego podejścia do komponowania muzyki

Koncepcja komponowania utworu muzycznego przez maszynę jako akt sztucznej twórczości muzycznej przez wielu krytyków może być uważana za niechcianą ingerencję w sferę zarezerwowaną i zdominowaną przez człowieka. Należy jednak zaznaczyć, że pierwsze próby zautomatyzowania komponowania utworu, mające na celu przyspieszenie, ułatwienie i analizę tego procesu, odbyły się na długo przed

pojawieniem się pierwszej elektronicznej maszyny liczącej. Marsenne, Kircher, Kirnberger, Himmel - to nazwiska matematyków i kompozytorów XVII i XVIII wieku, którzy poprzez swoją pasję do muzyki, umiejętności matematyczne oraz błyskotliwy umysł dokonali pierwszych prób modelowania procesu kreatywnego komponowania muzyki za pomocą ówcześnie znanych metod matematycznych.

Marsenne i Kircher [19] reprezentowali spojrzenie na proces komponowania jako umiejętność zastosowania kombinatoryki w definiowaniu i syntezie elementów tworzących ówczesne kompozycje muzyczne. Tworzenie muzyki zostało sprowadzone do analizy możliwych przejść pomiędzy występującymi w kanonie muzyki figurami muzycznymi, będącymi zbiorem odpowiednio zdefiniowanych i następujących po sobie dźwięków.

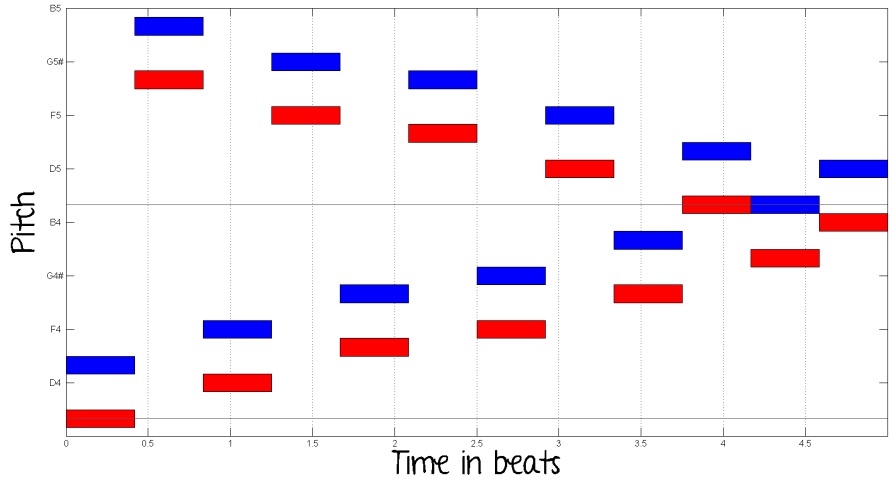
Kirnberger i Himmel [69] zaproponowali zastosowanie tzw. "Muzycznej gry w kości", polegającej na zastosowaniu macierzy gotowych fragmentów kompozycji, których ułożenie uzależnione było od rzutu dwiema sześciennymi kostkami. Suma wyrzuconych oczek wskazywała na identyfikator kolejnego fragmentu występującego w kompozycji wynikowej. Tak zdefiniowane podejście do komponowania muzyki stosowali ówcześnie wielcy mistrzowie klasyki, jak Mozart czy Bach.

Metoda Shoenberga - idea serializmu (Druga Szkoła Wiedeńska)

Arnold Shoenberg zaproponował ideę, której celem było obalenie tezy o hierarchicznej organizacji tonalnej skali muzycznej, w której poszczególne jej elementy mają różną ważność. Wprowadził pojęcie serializmu, w którym założył jednakowe znaczenie relacji dla wszystkich elementów dwunastostopniowej skali muzycznej. W praktyce oznaczało to przyjęcie serii dwunastu nut jako funkcji będącej podstawą kompozycji. Podejście zaproponowane przez Shoenberga definiowało jedno fundamentalne założenie: każda nuta występująca w wejściowej serii powinna być użyta w tej samej kolejności w serii wykorzystanej w kompozycji, co więcej, nie dopuszcza się występowania powtórzeń nut w kompozycji, zanim wszystkie nuty z podstawowej serii nut (podstawowego zbioru nut) nie zostaną użyte [85].

Kompozycja zbudowana z identycznych pojawiających się po sobie serii nut szybko może okazać się dla odbiorcy monotonna. Aby uniknąć efektu znużenia słuchacza, Shoenberg wprowadził pewne dopuszczalne transformacje bazowej serii nut, które nazwał odpowiednio: forwards, backwards, upside-down i backwards upside-down. Dodatkowo metoda Shoenberga wprowadza transpozycje: serie mogą zaczynać się od różnych nut, jednakże przy zachowaniu takich samych odległości (interwałów) pomiędzy pozostałymi składowymi serii, jak to ma miejsce w bazowej serii nut. Podejście Shoenberga traktuje serie nut jedynie w ujęciu ich wysokości (częstotliwości dźwięku). Późniejsi spadkobiercy idei serializmu tacy jak Anton Webern, Olivier Messiaen oraz Pierre Boulez, zdefiniowali pojęcie serii jako zbioru muzycznych parametrów określających nutę: wysokość, długość, czas narastania i dynamika.

Rozszerzenie idei serializmu zaproponowane przez Bouleza stwarzało możliwość niemal automatycznego generowania fragmentów utworu muzycznego. W założeniu generowany fragment poddawany był ciągłej transformacji w ten sposób, że każda wysokość dźwięku muzycznego występowała w parze zawsze z innym czasem jego wybrzmiewania, dynamiki oraz czasem narastania (tzw. atak). Boulez oprócz serii wysokości dźwięków zdefiniował serie długości dźwięków, serie wartości dynamiki dla poszczególnych dźwięków oraz serie trybów czasów narastania dźwięku (dźwięki



Rys. 1.5. Wykres pianolowy przykładowej serii nut (kolor czerwony) oraz serii przetransponowanej o 3 półtony (kolor niebieski). Opracowanie własne.

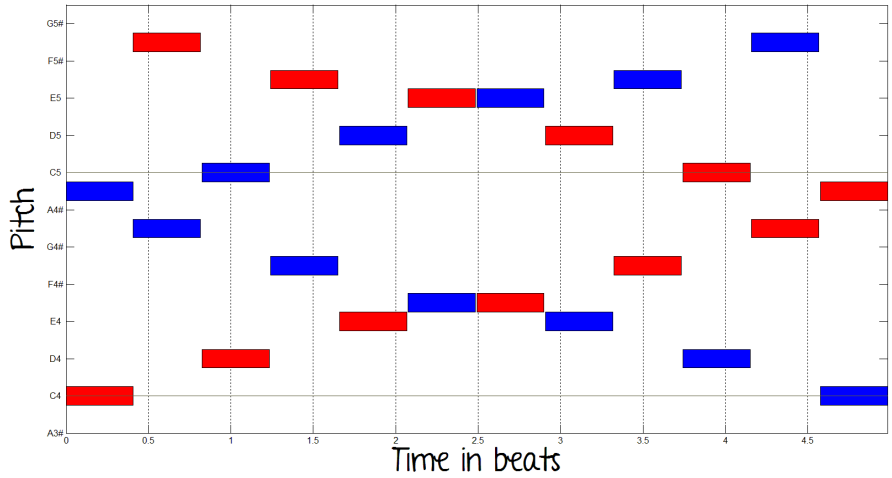
nagłe, szarpane, przeciągające się, itd.). Jak wielcy klasycy minionych epok (Mozart i Beethoven) tak i Schoenberg, Webern czy Alban Berg (kompozytor austriacki, student Antona Shoenberga) byli mieszkańcami Wiednia. Z tego powodu ideę sztuki komponowania zaproponowane przez Shoenberga oraz jej późniejsze rozszerzenia przyjęto nazywać Drugą Szkołą Wiedeńską [85].

Transformacje zaproponowane przez Shoenberga i jego następców są wykorzystywane przez współczesnych kompozytorów muzyki klasycznej w celu urozmaicenia np. głównego motywu występującego w utworze. Warto w tym miejscu przedstawić możliwości zastosowania tych transformacji w postaci zalgorytmizowanej, która dalej może zostać wykorzystana do budowania systemu generowania kompozycji muzycznych. W dalszej części przedstawiono implementacje wspomnianych metod Shoenberga w postaci kodu Matlab z wykorzystaniem wprowadzonej i opisanej wcześniej macierzy nut *MIDI*.

Algorytm transpozycji Shoenberga został zaprezentowany na listingu 5.1. Z założenia wg idei serializmu zmienna *inNoteMatrix* powinna być macierzą nut zawierającą dwanaście nut o różnych niepowtarzalnych wysokościach. Zmienna *amount* określa liczbę półtonów o jaką należy zwiększyć każdy z elementów wejściowej serii nut. Wizualizację transpozycji przykładowej serii przedstawiono na rys. 1.5. Kolorem czerwonym zaznaczono wejściową serię nut *inNoteMatrix*, kolorem niebieskim - serię *outNoteMatrix*. W przykładzie przyjęto wartość zmiennej *amount* równą 3 półtony.

Kolejną transformacją serii zaproponowaną przez Shoenberga jest inwersja. Dokładna inwersja zakłada odwrócenie serii dźwięków zarówno w aspekcie ich wysokości jak i wartości rytmicznych. Niektórzy kompozytorzy używają jednak niepełnej inwersji, odwracając jedynie wysokości dźwięków lub wartości rytmiczne. Implementacja podstawowej wersji tej metody została przedstawiona na listingu 5.2.

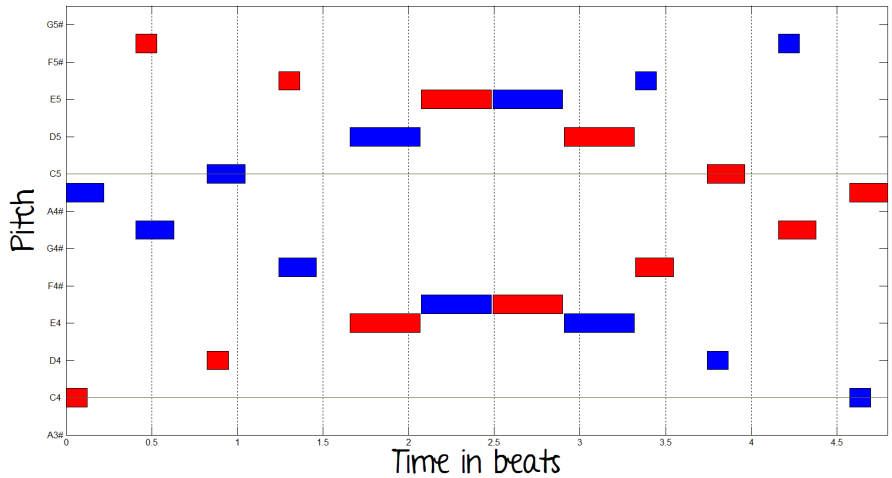
Mechanizm metody polega na odwróceniu wartości wysokości dźwięków znajdujących się w macierzy nut. Wysokości te są umiejscowione w macierzy w 4 kolumnie. Na rys. 1.6 został zaprezentowany przykładowy wykres pianolowy dla wyniko-



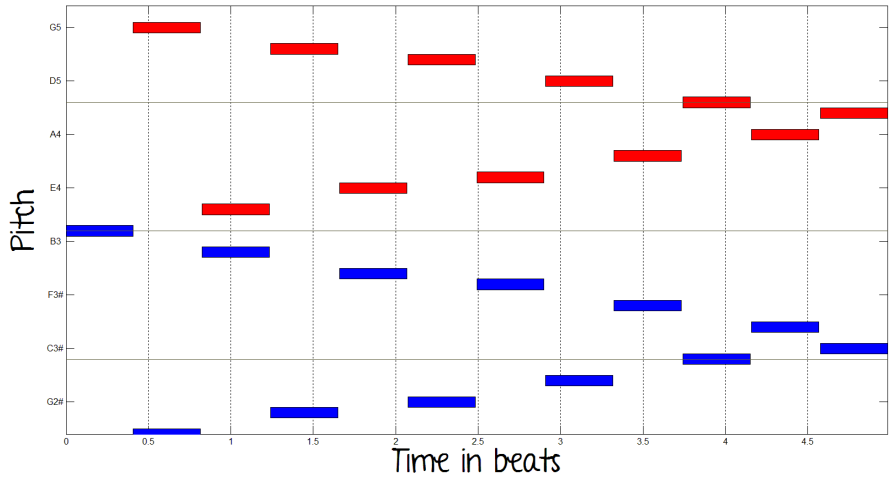
Rys. 1.6. Wykres pianolowy przykładowej serii nut (kolor czerwony) oraz serii z odwróconą wysokością dźwięków (kolor niebieski). Opracowanie własne.

wej macierzy nut oraz wejściowej macierzy nut metody inwersji wysokości dźwięków, zaprezentowanej na listingu 5.2.

Metodę inwersji wysokości dźwięków można poszerzyć o inwersję wartości rytmicznych. Przykład takiej modyfikacji przedstawiono na listingu 5.3. Rozszerzona metoda inwersji jest podobna do inwersji wysokości nut (lisiting 5.2). Jedyną różnicą jest ostatnia linijka kodu, w której następuje odwrócenie kolejności wartości rytmicznych. W macierzy nut zdefiniowanej w *MIDI* Toolbox wartości rytmiczne (długości) nut przechowywane są w postaci wyrażonej przez liczbę uderzeń na minutę (bpm) oraz w postaci liczby sekund. Z tego powodu istnieje konieczność odwrócenia zarówno kolumny wartości bpm (kolumna 2) jak i wartości długości dźwięku, wyrażonej w sekundach (kolumna 7). Na rys. 1.7 został przedstawiony wykres pianolowy przykładowych macierzy nut *inNoteMatrix* i *outNoteMatrix*.



Rys. 1.7. Wykres pianolowy przykładowej serii nut (kolor czerwony) oraz serii odwrotnej (kolor niebieski). Opracowanie własne.



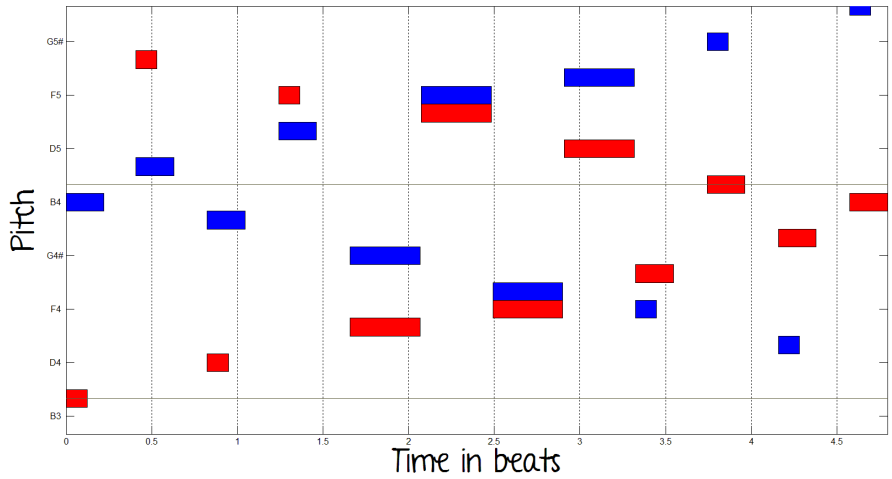
Rys. 1.8. Wykres pianolowy przykładowej serii nut (kolor czerwony) oraz serii przetransformowanej z wykorzystaniem metody backwards upside-down (kolor niebieski). Opracowanie własne.

Kolejną transformacją zaproponowaną przez Schoenberga jest metoda upside-down, której implementację w środowisku Matlab przedstawiono na listingu 5.4. W metodzie obliczane są różnice interwałów kolejnych występujących w macierzy *inNoteMatrix* serii wysokości dźwięków. Różnice przechowywane są w zmiennej *diffNote*. Pierwszy element serii wysokości dźwięków jest wspólny zarówno dla macierzy wejściowej jaki i budowanej. Następnie w pętli dokonuje się wyliczenia nowej i-tej wartości wysokości dźwięku dla macierzy *outNoteMatrix*, z wykorzystaniem wcześniej obliczonych wartości różnic interwałów oraz poprzednio obliczonej wartości wysokości dźwięku macierzy *outNoteMatrix*. Na rys. 1.8 przestawiono wykres pianolowy dla przykładowej serii nut oraz serii przetransformowanej z wykorzystaniem metody upside-down.

Ostatnią z wymienionych transformacji jest metoda backwards upside-down, będąca połączeniem dwóch wcześniej zdefiniowanych transformacji: backwards i upside-down. Przykład implementacji w środowisku Matlab przedstawiono na listingu 5.5.

Na macierzy wejściowej *inNoteMatrix* dokonuje się pełnej inwersji, której wynikiem jest pośrednia macierz nut wykorzystana do dalszej transformacji. Następnie pośrednia macierz *buffNoteMatrix* jest przekształcana metodą upside-down do postaci wynikowej macierzy *outNoteMatrix*. Na rys. 1.9 został przedstawiony wykres pianolowy dla przykładowej serii nut oraz serii nut przetworzonej za pomocą metody backwards upside-down.

Idea serializmu Schoenberga ma zarówno zwolenników jak i przeciwników, jednak transformacje zaproponowane przez Schoenberga są wykorzystywane przez współczesnych kompozytorów muzyki do urozmaicania i przetwarzania utworów muzycznych, w celu poszukiwania nowej formy, koncepcji i myśli. Zapisanie wymienionych w tym rozdziale metod do postaci programowej ma na celu ukazanie idei serializmu jako zestawu gotowych transformacji, które mogą być wykorzystane do budowy systemu algorytmicznego komponowania muzyki.



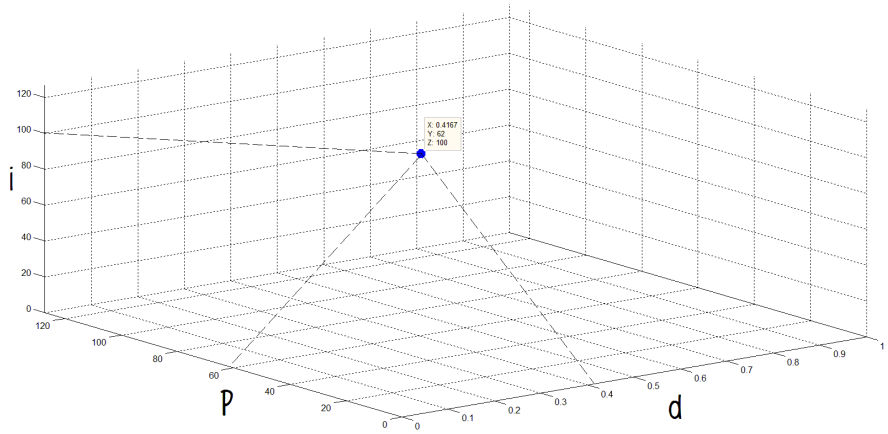
Rys. 1.9. Wykres pianolowy przykładowej serii nut (kolor czerwony) oraz serii przetransformowanej z wykorzystaniem metody backwards upside-down (kolor niebieski). Opracowanie własne.

Metoda formalizmu muzycznego Xenakisa

Serializm Schoenberga nie zapełnił całkowicie przestrzeni nowatorskich sposobów kreowania sztuki muzycznej, a w niektórych aspektach był krytykowany za zbytnie oddalanie się w stronę kompozycji atonalnej. Przeciwnikiem serializmu był znany kompozytor amerykański John Cage, który w swoich metodach komponowania wykorzystywał m.in rzut monetą jako warunek decyzyjny przy generowaniu kolejnych fragmentów utworu. Obie tendencje doprowadziły do nowego spojrzenia na komponowanie muzyki, w szczególności do generatywnego procesu leżącego u podstaw stworzenia fragmentu muzycznego, powstałego zarówno w sposób losowy (Cage) jak i bardziej kontrolowany (Shoenberg) [85].

Ciekawe podejście zostało zaprezentowane przez kompozytora Iannisa Xenakisa, który nie trzymał się ani ściśle figlarnego sposobu zaprezentowane przez Cage, ani równie uszeregowanych struktur wprowadzonych przez Schoenberga. Xenakis był zainspirowany wieloma współczesnymi muzykami komputerowymi, a jego podejście ucieleśnia statystyczne zasady rządzące procesem tworzenia muzyki. Naukowcy używają statystyk aby uogólnić prognozy dotyczące masowych i losowych fluktuacji, takich jak np. ogólna prędkość i trajektoria molekuł w chmurze gazu. Xenakis domniemywał, że podobne przesłanki mogą być zastosowane do dużej liczby zdarzeń muzycznych, takich jak dźwięk generowany z wielu skrzypiec, w wielu zakresach . W tym rozdziale przedstawiono aspekt prac Xenakisa, będący abstrakcyjnym formalizmem podkreślającym jego myśl muzyczną. Przedstawione tu podejście ucieleśnia muzyczny formalizm bazujący na teorii mnogości, a zainspirowany jest problematyką wprowadzoną przez Xenakisa w książce zatytułowanej Formalized Music (wydanej po raz pierwszy w roku 1971, z późniejszym wydaniem [106]).

Na wstępie zdefiniowane zostaną trzy zbiory zawierające dodatnie liczby rzeczywiste: P, D i I, odpowiadające interwałom wysokości dźwięku, czasie jego wybrzmiewania oraz intensywności (zdefiniowanej w dalszej części). Dodatkowo zbiory te są uporządkowane.



Rys. 1.10. Przykładowy wektor v_m. Opracowanie własne.

Ogólne prawo kompozycji dla każdego z wprowadzonych zbiorów, może zostać zdefiniowane następująco: Niech v_m będzie wektorem składającym się z komponentów p_n, d_n oraz i_n, takie, że $p_n \in P$, $d_n \in D$ oraz $i_n \in I$, odpowiednio rozmieszczone w wektorze $v_m = \{p_n,\ d_n,\ i_n\}$. W tym kontekście, wektor jest punktem w przestrzeni trójwymiarowej, którego współrzędne są określone przez p_n, $d_n$oraz i_n. Przykładowy wektor $v_m = \{62,\ 0.4167,\ 100\}$ został przedstawiony na rys. 1.10. Szczególny przypadek wektora, w którym wszystkie elementy są zerem, jest określony jako wektor zerowy v_0, definiujący początek układu współrzędnych.

Bazując na właściwościach algebraicznych zbiorów wprowadzonych wcześniej, można zdefiniować dwa prawa kompozycji dla wektora v_m: prawo dodawania oraz prawo multiplikacji.

Prawo dodawania wektorów zdefiniowano następująco:

$(v_i = \{p_m, d_n, i_o\}) + (v_j = \{p_x, d_y, i_z\}) \Rightarrow (v_i + v_j) = \{(p_m + p_x), (d_n + d_y), (i_o + i_z)\}$

Prawo multiplikacji (c jest liczbą naturalną):

$c \times (v_i = \{p_m, d_n, i_o\}) \Rightarrow c \times v_i = \{c \times p_m, c \times d_n, c \times i_o\}$

Ważnym ograniczeniem narzuconym na wartości p_n, d_n oraz i_n jest wykorzystywanie wartości z zakresu słyszalnego dla człowieka (dowolność w wyborze wartości wysokości dźwięku, jego długości i intensywności tylko w zakresie, w którym człowiek jest w stanie zidentyfikować wektor). Dla przykładu zdefiniowano wektor zerowy $v_0 = \{60,\ 1,\ 64\}$. Wartości składowych odpowiadają wybranym właściwościom dźwięku opisanego w ramach macierzy nut narzędzia *MIDI* Toolbox. Elementy zbioru P są reprezentowane jako półtony w standardzie nuty *MIDI* i są rozmieszczone w macierzy nut, w czwartej kolumnie (rys. 1.4). Elementy zbioru D są przedstawione jako metryczne uderzenia (jednostki bpm) i znajdują się w macierzy nut w kolumnie drugiej. Zbiór I (intensywności) określony jest na wartościach definiujących głośność dźwięku (od *pianissimo* do *fortissimo*) wyrażoną w postaci wartości z przedziału 0 - 127. Tak więc, w prezentowanym przykładzie początkiem układu będzie wektor wartości: wysokość dźwięku C3 (liczba 60 w standardzie *MIDI*), długość jedno uderzenie (1 beat) oraz intensywność (*mezzo piano*, wg standardu *MIDI* - 64).

W tym punkcie zdefiniowano kilka przykładowych reguł generatywnych wykorzystywanych do tworzenia fragmentów muzycznych. Poniższe wyrażenia definiują

trzy generatory wektora (*np* oznacza *n* jednostek powyżej początku *p*, natomiast -*np* oznacza *n* jednostek poniżej początku *p*):

$v_1 = \{14p, 4d, i\}$
$v_2 = \{(X + 12)p, Yd, Zi\}$
$v_3 = \{(X - 4)p, Yd, Zi\}$

Poniższe wyrażenia F_n definują powstawanie wektorów:

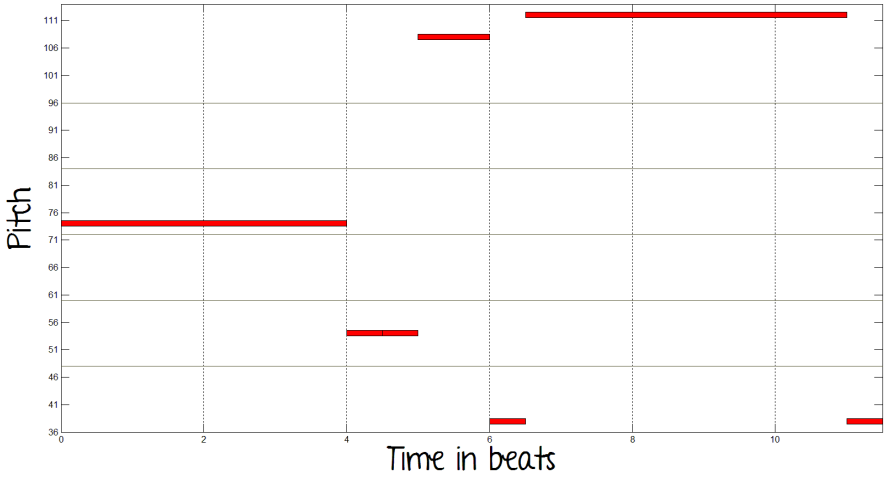
$F_1 = [v_1]$
$F_2 = [v_2, v_2, 2 \times v_2]$
$F_3 = [v_3, v_1 + v_3, v_3]$

Przedstawione wyrażenia mogą być wdrożone w postaci funkcji. Przykład implementacyjny w środowisku Matlab z wykorzystaniem narzędzia *MIDI* Toolbox przedstawiono na listingu 5.6. Funkcja *musicVectors* pobiera argument w postaci wektora v_0 (zdefiniowanego wcześniej w postaci $v_0 = \{p_0, d_0, i_0\}$). Następnie określone zostały współrzędne *x*, *y* i *z*, biorące udział w wyliczeniu wektorów w funkcjach *F1*, *F2* i *F3*. Funkcja *F1* wg wcześniejszego założenia zwraca jedynie wektor v_1. Funkcje *F2* i *F3* zwracają odpowiednie złożenia wektorów v_1,v_2 i v_3w postaci macierzy. W zmiennej *melodyMatrix* przechowywana jest konkatenacja wyjściowych wartości funkcji *F1*, *F2* i *F3* w postaci macierzy

$$\begin{bmatrix} p_1 & d_1 & i_1 \\ p_2 & d_2 & i_2 \\ \vdots & \vdots & \vdots \\ p_n & d_n & i_n \end{bmatrix}.$$

Za pomocą wbudowanej w *MIDI* Toolbox metody *createnmat* tworzona jest macierz nut, zawierająca dźwięki muzyczne o wyliczonych przez funkcje wysokościach (wartości $p_{1..n}$). Metoda *createnmat* zakłada generowanie dźwięków muzycznych o jednakowej intensywności (głośności) oraz długości, dlatego należy dokonać pewnych modyfikacji wygenerowanej macierzy. W pętli dla kolejnych dźwięków muzycznych dokonuje się ustawianie wartości czasu trwania ('dur') i intensywności ('vel') adekwatnie do wartości $d_{1..n}$ oraz $i_{1..n}$. Aby ustrzec się przed występowaniem równolegle brzmiących dźwięków, które mogłyby powodować dysonans, zastosowano przesunięcie w czasie kolejnego modyfikowanego dźwięku o wartość sumy czasu trwania i czasu pojawienia się wcześniejszego dźwięku. Ostatecznie zmienna melody zawiera macierz nut wygenerowaną na podstawie funkcji *F1*, *F2* i *F3* oraz wektorów v_1,v_2 i v_3.

Na rys. 1.11 przedstawiono wykres pianolowy melodii wygenerowanej za pomocą opisanej przykładowej implementacji generatorów wykresów i funkcji *F1*, *F2* i *F3* w środowisku Matlab. Jako wektor zerowy przyjęto $v_0 = \{60, 1, 64\}$. W implementacji pominięto wiele ważnych aspektów muzycznych, takich jak rytmika czy polifonia generowanych sentencji. Prezentowany przykład może być dowolnie rozbudowany o nowe generatory i funkcje, a także dodatkowe współrzędne wektorów (np. uwzględniające późniejsze łączenie dźwięków muzycznych w czasie). W przedstawionym przykładzie można zdefiniować wektor pauzy (odstępu między wybrzmiewaniem kolejnych dźwięków) jako $v_4 = \{0, Yd, 0\}$, czyli wektor o zerowej wartości intensywności (głośności) i wysokości dźwięku. W ten sposób istnieje możliwość kontrolowania rozmieszczenia dźwięków muzycznych w ramach macierzy nut.



Rys. 1.11. Wykres pianolowy wygenerowanej melodii dla $v_0 = \{60, 1, 64\}$ z wykorzystaniem przykładowej implementacji w środowisku Matlab. Opracowanie własne.

Komponowanie muzyki z wykorzystaniem idei łańcuchów Markowa

Systemy komponowania muzyki w sposób losowy zawsze były obiektem fascynacji kompozytorów (Mozart, Cage). Zainteresowanie tym sposobem podejścia do generowania muzyki wzrosło po pojawieniu się komputera, jako narzędzia rozszerzającego możliwości kompozytora. Należy pamiętać, że "surowe" wyniki procesu "losowania" muzyki rzadko kiedy spełniają oczekiwania kompozytorów, którzy często preferują ocenę przydatności tego procesu w kontekście swoich preferencji estetycznych i celów muzycznych. Możliwości komputera jako narzędzia programistycznego rozszerzają pojęcie muzyki generowanej w sposób losowy, o nowe aspekty, które umożliwiają m.in umieszczenie zdefiniowanych warunków komponowania w procesie losowym.

W muzyce, prawdopodobieństwo wykorzystywane jest zwykle do generowania materiału muzycznego poprzez wybór elementów ze zbioru. Generując melodię, komputer może zostać zaprogramowany tak, aby w jednym czasie losowo wybierał jedną nutę ze zbioru dopuszczalnych nut, i następnie odtwarzał ją bezpośrednio poprzez syntezator. W przypadku, gdy zbiór nie zawiera powtórzeń problem jest trywialny. Jednakże, gdy w zbiorze pojawią się duplikacje (replikacje) danej nuty, rosną jej szanse na wylosowanie do kompozycji wyjściowej. Dodatkowo przy wykorzystaniu prawdopodobieństwa warunkowego, szansa losowania może być modyfikowana bazując na wystąpieniu poprzedniej nuty.

Matematycy zaproponowali jednak inne narzędzie dokonywania wyboru na podstawie prawdopodobieństwa, które nie opiera się wyłącznie na liczbie elementów: funkcje dystrybucji. W tym kontekście, pojęcie generatora stochastycznego oznacza system, który generuje sekwencje parametrów muzycznych, wybierając je z danego zbioru, bazując na określonych funkcjach dystrybucji [85].

W zależności od zastosowania w aplikacji, funkcja dystrybucji może być zaimplementowana jako macierz prawdopodobieństwa, która przechowuje wartości odpowiadające prawdopodobieństwu zajścia określonego zdarzenia lub serii zdarzeń. W systemie komponowania, macierze prawdopodobieństw mogą być częścią procedury decyzyjnej podczas generowania nowego utworu. Przy zastosowaniu rozkładu

Procedura	Wartość prawdopodobieństwa
$transp(inNoteMatrix, 8)$	p_1
$backwards(inNoteMatrix)$	p_2
$fullBackwards(inNoteMatrix)$	p_3
$upsideDown(inNoteMatrix)$	p_4
$backwardsUpsideDown(inNoteMatrix)$	p_5

Tab. 1.7. Przykładowa tabela prawdopodobieństwa zajścia zdarzenia w systemie komponowania muzyki (na podstawie [85]). Opracowanie własne.

równomiernego, proces wyboru jednego z n równomiernie dystrybuowanych zdarzeń może być zaimplementowany jako procedura losowego wyboru wartości pomiędzy 0 a *n-1*. Zdarzeniem może być w tym kontekście wykonanie określonej transformacji na serii nut lub zwrócenie serii nut przetworzonej w poprzednim kroku losowania.

Inną metodą implementacji macierzy prawdopodobieństwa jest wykorzystanie mechanizmu kumulacji. Najczęściej używaną wizualizacją tego podejścia jest linia prosta o długości 1, podzielona na segmenty o szerokości związanej z prawdopodobieństwem wystąpienia określonego zdarzenia. Aby wygenerować przykładowy wynik, należy losowo wybrać liczbę z przedziału pomiędzy 0 i 1, a następnie określić, do którego segmentu linii należy. Suma wszystkich wartości prawdopodobieństwa jest równa 0.

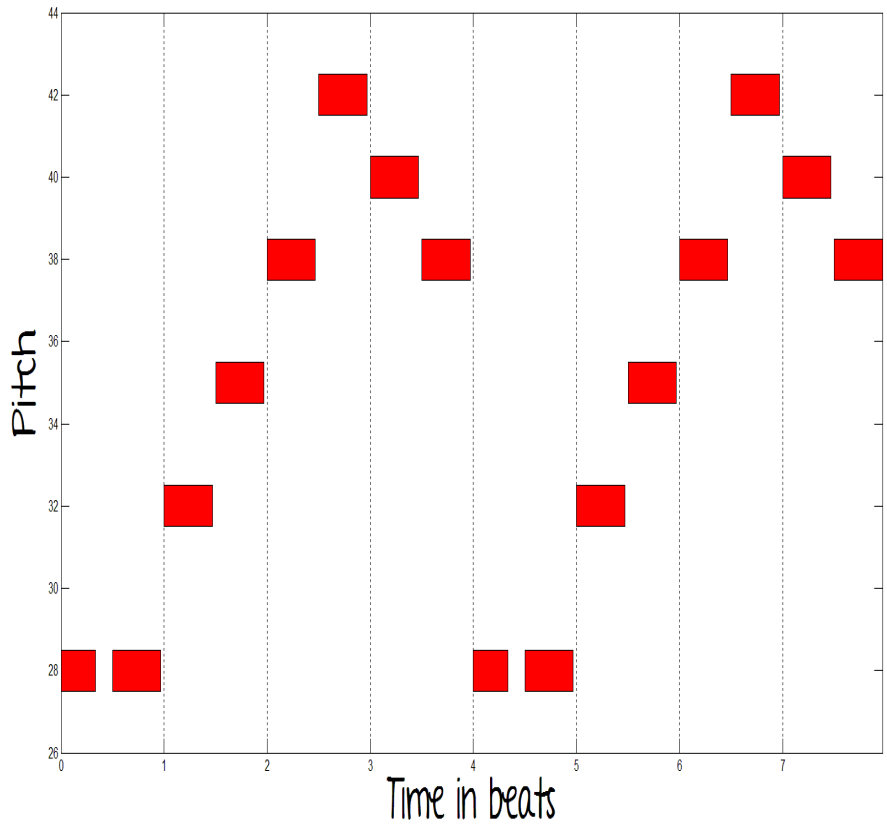
W kontekście systemu komponowania muzyki, kompozytor domyślnie określa n możliwych decyzji podjętych podczas generowania utworu, a każda z nich nawiązuje do:

1. procedury generowania zdarzeń muzycznych,
2. liczby z przedziału pomiędzy 0 i 1, która określa prawdopodobieństwo wyboru procedury

Przykładową sytuację przedstawia tab. 1.7, w której jako procedury zastosowano sformułowane wcześniej transformacje Shoenberga. Lista procedur może zostać poszerzona o kolejne, z zachowaniem warunku sumowania wszystkich wartości prawdopodobieństwa do wartości 1. W odniesieniu do tab. 1.7 przykładową implementację mechanizmu wyboru kumulatywnego przedstawiono na listingu 5.7.

Zaprezentowana procedura pobiera dwa argumenty wejściowe: macierz nut (*inNoteMatrix*) oraz wektor p zawierający wartości prawdopodobieństw wystąpienia (wyboru) określonej procedury. Wynikowa przetransformowana macierz nut zwracana jest w zmiennej *outNoteMatrix*. Każdy wybór kumulatywny związany jest z dokładnie jedną procedurą (zgodnie z tab. 1.7).

Tabele (macierze) prawdopodobieństw wystąpienia określonego zdarzenia związane są z systemami bazującymi na prawdopodobieństwie warunkowym, w których prawdopodobieństwo wystąpienia danego zdarzenia zależy od poprzedniego lub kilku poprzednich zdarzeń. Systemy takie nazywane są łańcuchami Markowa. Liczba poprzednich zdarzeń branych pod uwagę do obliczenia pojawienia się bieżącego zdarzenia nosi nazwę rzędu łańcucha. W takim ujęciu łańcuch Markowa, w którym do obliczenia prawdopodobieństwa wystąpienia bierze się pod uwagę jedynie poprzednie zdarzenie, nazywa się łańcuchem pierwszego rzędu. Gdy do obliczeń prawdopodobieństwa brane są pod uwagę dwa poprzednie zdarzenia, łańcuch Markowa jest drugiego rzędu. Łańcuch Markowa zerowego rzędu zdefiniowany jest w sytuacji, gdy wystąpienie bieżącego zdarzenia nie zależy od żadnego wcześniejszego zdarzenia.



Rys. 1.12. Wykres pianolowy fragmentu utworu "Pretty woman" Roya Orbisona. Opracowanie własne.

W ogólności n-ty łańcuch Markowa jest reprezentowany przez macierz prawdopodobieństwa tranzycji (przejść pomiędzy zdarzeniami) o wymiarach $n+1$. Macierz tranzycji zawiera informacje o prawdopodobieństwie przejścia do określonego stanu (zajścia zdarzenia) łańcucha, mając na uwadze wystąpienia n poprzednich stanów. Stanem (zdarzeniem) w ujęciu systemów komponowania muzyki może być zarówno zastosowanie odpowiedniej procedury, jak i element struktury muzycznej (utwór, jego fragment bądź pojedynczy dźwięk). W zastosowaniu idei macierzy nut stanem może być zarówno postać całej macierzy jak i jej fragment (np. pojedynczy wiersz, będący sparametryzowanym dźwiękiem muzycznym).

W celu przedstawienia przykładowej metody tworzenia macierzy transpozycji, będącej potencjalnie częścią systemu komponowania muzyki, posłużono się przykładowym fragmentem utworu Roya Orbisona "Pretty woman", którego wykres pianolowy przedstawiono na rys. 1.12. Stanem należy w tym kontekście nazywać pojawienie się dźwięku muzycznego o określonej wysokości. Z przykładu z rys. 1.12 można wyróżnić zbiór stanów $S = \{28, 32, 35, 38, 40, 42\}$, w jakich może znajdować się łańcuch. Aby wyznaczyć macierz prawdopodobieństwa tranzycji dla analizowanego fragmentu, można wykorzystać przykładową implementację algorytmu generowania macierzy tranzycji, przedstawioną na listingu 5.8.

Funkcja generująca macierz tranzycji (*markov1Order*) dla łańcucha pierwszego rzędu, pobiera jako argument wejściowy macierz nut analizowanego fragmentu

-	28	32	35	38	40	42
28	0.67	0	0	0.33	0	0
32	0	0	1	0	0	0
35	0	0	0	1	0	0
38	0.25	0	0	0	0	0.75
40	0	0	0	1	0	0
42	0	0	0	0	1	0

Tab. 1.8. Macierz prawdopodobieństwa tranzycji łańcucha Markowa pierwszego rzędu dla przykładowego fragmentu utworu z rys 1.12. Opracowanie własne.

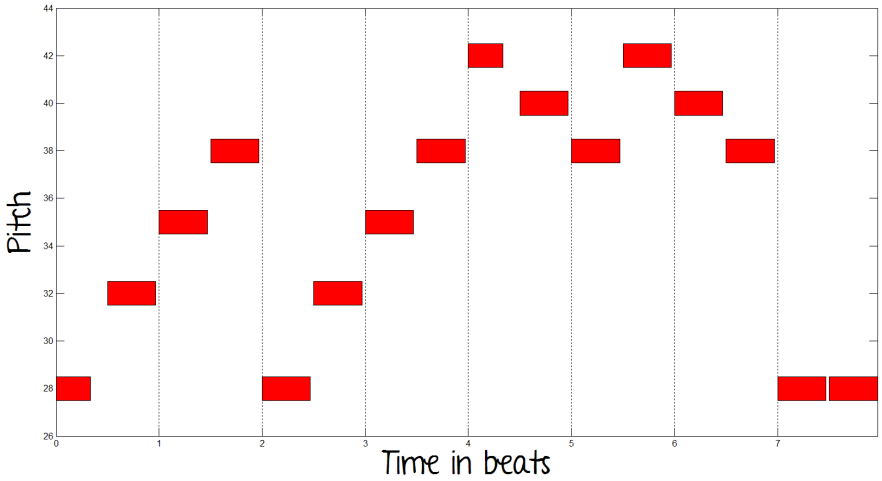
utworu (*inputNoteMatrix*), zwraca natomiast zmienną *transMatrix*, będącą wygenerowaną dynamicznie macierzą tranzycji oraz zmienną *uniqueStates*, zawierającą wektor możliwych stanów łańcucha. Początkowo wyszukiwane są unikalne wartości wysokości dźwięków muzycznych (4 kolumna w macierzy nut), będących podstawą do oszacowania liczby stanów, w jakich może znajdować się łańcuch. Po zainicjowaniu macierzy zerowymi wartościami komórek, następuje analiza wejściowej macierzy nut pod względem lokalizacji znalezionych stanów oraz stanów ich poprzedzających. W odpowiednich komórkach macierzy umieszczane są wartości liczby wystąpień danego stanu poprzedzającego analizowany. Ostatnim elementem algorytmu jest normalizacja wierszy - suma prawdopodobieństw wystąpienia danego stanu (kolumna) po analizowanym (wiersz) wynosi 1.

W tab. 1.8 przedstawiono macierz prawdopodobieństwa tranzycji łańcucha pierwszego rzędu dla analizowanego przykładu. Poziomo przedstawione są bieżące stany łańcucha, podczas gdy w układzie pionowym znajdują się stany poprzedzające. Przykładowo aby określić prawdopodobieństwo pojawienia się dźwięku o wysokości 32 po dźwięku o wysokości 28, należy znaleźć stan 28 w pierwszej kolumnie macierzy, poziomo wyszukać kolumnę dla stanu 32, a następnie w miejscu przecięcia wiersza stanu 28 i kolumny 32 odczytać wartość prawdopodobieństwa przejścia pomiędzy tymi stanami. W przytoczonym wypadku wartość ta wynosi 0,5.

Łańcuchy Markowa posiadają kilka istotnych, z punktu widzenia twórcy systemu komponowania muzyki, właściwości matematycznych. Stan X jest osiągalny ze stanu Y, jeżeli jest możliwe osiągnięcie stanu X ze stanu Y w skończonej liczbie kroków. Jeżeli stan Y jest osiągalny ze stanu X, a stan X jest osiągalny ze stanu Y, to dwa stany są ze sobą skomunikowane. Relacja komunikacji w łańcuchu Markowa oznacza równoważność ($X \leftrightarrow Y$), ponieważ spełnia następujące wymagania:

— jest zwrotna, co oznacza, że stan zawsze jest skomunikowany z samym sobą,
— jest symetryczna: jeżeli stan X komunikuje się ze stanem Y, to stan Y również komunikuje się ze stanem X,
— jest przechodnia: jeżeli stan X komunikuje się ze stanem Y, a stan Y komunikuje się ze stanem Z, to stan X komunikuje się również ze stanem Z.

Stany łańcucha Markowa mogą być pogrupowane w równoważne klasy skomunikowanych stanów. Te stany, które są pewnie osiągalne (ich prawdopodobieństwo wystąpienia wynosi 1), po tym jak zostały osiągnięte w łańcuchu, nazywa się stanami rekurencyjnymi, a klasę do której należą - klasą rekurencyjną. Stany, które mogą być już nie osiągalne (w tym, te które nie są rekurencyjne, a w ogólności - ich prawdopodobieństwo wystąpienia w łańcuchu jest mniejsze od 1) nazywa się stanami chwilowymi (przejściowymi), a klasę do której należą - klasą chwilową (przejściową).



Rys. 1.13. Wykres pianolowy przykładowej melodii wygenerowanej na podstawie macierzy z tab. 1.8. Opracowanie własne.

Każdy łańcuch Markowa powinien posiadać przynajmniej jedną klasę rekurencyjną oraz klasy chwilowe (łańcuch nie może zawierać jedynie klas chwilowych). Łańcuch Markowa zawierający dokładnie jedną klasę rekurencyjną i możliwe klasy chwilowe określa się jako ergodyczny. Są to łańcuchy preferowane w systemach komponowania muzyki, ponieważ umożliwiają kompozytorom predykcję zachowania łańcucha oraz wyników, jakie są generowane na podstawie jego wykonania [85].

Przykład wygenerowanego fragmentu melodii na podstawie macierzy przedstawionej w tab. 1.8. pokazano na rys. 1.13. Początek łańcucha zainicjowano stanem 28, z którego nastąpiło przejście do stanu 32 (z prawdopodobieństwem 0,5). Następnie wystąpiły kolejne przejścia z prawdopodobieństwem wystąpienia przedstawionym w macierzy tranzycji. Pełny łańcuch dla wygenerowanej melodii został przedstawiony poniżej. Strzałki pokazują kierunek przejścia pomiędzy stanami, natomiast wartość prawdopodobieństwa danej tranzycji umieszczona została nad strzałką.

$28 \xrightarrow{0,5} 32 \xrightarrow{1,0} 35 \xrightarrow{1,0} 38 \xrightarrow{0,25} 28 \xrightarrow{0,5} 32 \xrightarrow{1,0} 35 \xrightarrow{1,0} 38 \xrightarrow{0,75} 42 \xrightarrow{1,0} 40 \xrightarrow{1,0} 38 \xrightarrow{0,75} 42 \xrightarrow{1,0} 40 \xrightarrow{1,0} 38 \xrightarrow{0,25} 28 \xrightarrow{0,5} 28$

Łańcuch Markowa wyższego rzędu, w których wystąpienie zdarzenia zależy od większej liczby zdarzeń go poprzedzających, działa na podobnej zasadzie, co macierz tranzycji z tab. 1.8. Różnicą są dodatkowe wymiary, przypadające kolejnym generacjom poprzednich stanów. Dla przykładu, macierz prawdopodobieństwa tranzycji łańcucha drugiego rzędu musi posiadać trzy wymiary: jeden dla generacji bieżącego stanu, jeden dla poprzednio wygenerowanego stanu oraz jeden dla stanu poprzedzającego wcześniej wygenerowany.

Gramatyki komponowania

Poprzednio przedstawiono utwór muzyczny w ujęciu funkcji dystrybucji tworzących go elementów. Wykorzystując łańcuchy Markowa zamodelowano proces komponowania utworu, w którym stanem modelu może być zarówno zdefiniowane przekształcenie, fragment utworu (wzorzec) bądź dźwięk muzyczny. Analizując budowę jakiegokolwiek utworu muzycznego zauważyć można pewną połączoną hierarchiczną strukturę: na najniższym poziomie tej hierarchii znajdują się dźwięki

muzyczne, które budują *motywy*, te z kolei tworzą *frazy* pogrupowane w *zdania muzyczne*, które następnie tworzą *okresy*. Niektóre formy muzyczne dają się przedstawić w postaci pojedynczej hierarchicznej zagnieżdżonej struktury, często jednak złożoność formy jest na tyle duża, że jej przedstawienie w postaci jednej struktury wydaje się być bezcelowe. Niemniej jednak kompozycje wydają się posiadać, przynajmniej częściowo, organizację typu warstwowego [85].

Hierarchiczne podejście do formy muzycznej posiada silne odniesienie do lingwistyki. W takim porównaniu dźwięki muzyczne (nuty) mogą bezpośrednio odpowiadać fonemom, które przechodzą w słowa, te w zdania, a następnie akapity, i tak dalej. To podobieństwo spowodowało wzajemne przenikanie zainteresowań pomiędzy muzykologami a lingwistami.

Ogólnie rzecz biorąc lingwistyka skoncentrowana jest na budowie słów i zdań języka. Język można formalnie zdefiniować jako podzbiór nieskończonego zbioru uformowanego przez wszystkie możliwe kombinacje zbioru jednostek podstawowych. Dla przykładu, podstawową jednostką języka muzycznego może być dźwięk muzyczny. W takim wypadku wszystkie możliwe kombinacje dźwięków tworzą uniwersum wszystkich możliwych kompozycji muzycznych. Język muzyczny może być zatem traktowany jako podzbiór kombinacji utworzonych z wykorzystaniem pewnych ograniczeń i reguł. Ten formalizm określający ograniczenia i reguły, które mogą być również wykorzystywane w generowaniu muzyki, nosi nazwę gramatyki formalnej.

Pojęcie gramatyki formalnej pojawiło się w latach 50-tych kiedy to znany lingwista Noam Chomsky opublikował rewolucyjną książkę "Syntatic Structures" (1957). Chomsky zasugerował, że ludzie posiadają zdolność mówienia danym językiem naturalnym i rozumienia języka głównie dlatego, że opanowaliśmy jego gramatykę [85]. Zdaniem Chomsky'ego, specyfika gramatyki musi bazować na formalizmie matematycznym w celu dokładnego opisania funkcjonalności gramatyki (np. formalne reguły opisu, generowania i transformacji zdań). Gramatyka powinna zatem zarządzać jednoznacznym i obiektywnym charakteryzowaniem zdań, bez możliwości wprowadzania domysłów. W ogólności Chomsky uważał, że powinna istnieć możliwość zdefiniowania uniwersalnej gramatyki, która mogłaby być zaaplikowana we wszystkich językach. Język może być badany z dwóch różnych punktów widzenia oznaczonych jako tzw. głęboką strukturę oraz strukturę powierzchniową. Chomsky postawił tezę, że umysł człowieka od urodzenia jest ściśle zakorzeniony w zdolności przetwarzania głębokiej struktury. Struktura powierzchniowa reprezentuje natomiast specyficzne cechy konkretnych języków, które są nabywane przez dzieci podczas dorastania i rozwoju w danej kulturze.

Wielu kompozytorów-badaczy pozostało i pozostaje pod wpływem prac Chomsky'ego. W początkowych pracach Lerdhala i Jackendoffa ([67]) oraz Cope'a ([36]) widać próby bezpośredniego zastosowania założeń zdefiniowanych przez Chomsky'ego w aspekcie muzyki. Powstało wiele prac inspirowanych ogólnymi zasadami strukturalnego opisu zdań, w tym wiele przydatnych rozwiązań związanych z analizowaniem muzyki ([77]).

W celu zdefiniowania gramatyki formalnej należy określić *zbiór symboli terminalnych*, *zbiór symboli nieterminalnych* , *symbol startowy* oraz zbiór reguł produkcji słowa . Poniżej został przedstawiony przykład definicji gramatyki dla prostego systemu generowania muzyki pracującego w środowisku Matlab z wykorzystaniem narzędzia *MIDI* Toolbox.

Dla potrzeb przykładu zdefiniowano kilka fundamentalnych pojęć wykorzystywanych w regułach komponowania (w tym sensie: produkcji) muzyki:
— pojęcie D_n związane z dźwięku muzycznym do którego odnoszą się elementy zbioru interwałów (np. $D_1 = C4$);
— pojęcie I_n związane z interwałem pomiędzy dwoma dźwiękami muzycznymi wyrażonym w półtonach (np. $I_1 = 1$);
— pojęcie kierunku K_n interwału (np. K_1= w górę);
— pojęcie sekwencji tworzącej podzbiór elementów I i K, S_n.
— pojęcie zbioru połączonych sekwencji S, ZS_n.
Dla zdefiniowanych pojęć należy określić leksykon gramatyki, na który składają się przykładowe dwa zbiory interwałów I oraz kierunków K:
— $I = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13\}$;
— $K = \{'/','\backslash','-'\}$, przy czym: '/' oznacza "w górę", '\'oznacza "w dół", '-' oznacza "bez zmian".
Przykład reguły produkcji dla opisywanej gramatyki został zdefiniowany poniżej:
— $ZS_1 \rightarrow S_1 + S_2$
— $S_1 \rightarrow (I_5, K_1) + (I_8, K_1) + (I_{11}, K_1)$
— $S_2 \rightarrow (I_5, K_2) + (I_8, K_2)$
Pary postaci (I, K) definiują interwał oraz kierunek jego zmiany, np. (I_5, K_1) oznacza zmianę interwału o 5 półtonów w górę od bieżącego dźwięku. Wyznaczając jako punkt odniesienia dźwięk muzyczny $D_1 = C4$ oraz pamiętając, że dźwięk C4 wyraża się w wartości 60 półtonów w standardzie *MIDI*, wg reguły produkcji przedstawionej powyżej otrzymać można zbiór sekwencji ZS_1:

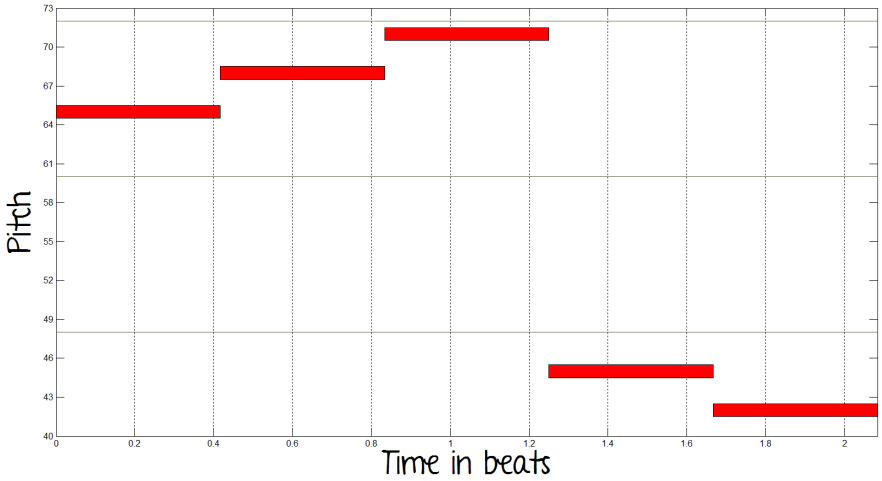
$S_1 \rightarrow$(5,/) + (8,/) + (11,/)
$S_2 \rightarrow$(5,\) + (8,\)
$ZS_1 \rightarrow$(5,/) + (8,/) + (11,/) + (5,\) + (8,\)

Na rys. 1.14 przedstawiono pianolowy wykres kompozycji wyprodukowanej wg reguły ZS_1. Aby przykład pozostał prosty i czytelny, pominięto aspekt struktury rytmicznej dźwięku, przyjmując jednakową długość wszystkich dźwięków (0,25 sekundy) oraz bezpośrednie ich następstwo (bez wprowadzania pauz pomiędzy dźwiękami).

Oprócz reguł produkcji w gramatyce formalnej istnieją tzw. reguły transformacji, które bazując na logicznym sprawdzeniu reguł produkcji, dokonują transformacji sentencji. Poniżej zaprezentowana została reguła transformacji, która tworzy nową sekwencję S_3 poprzez połączenie zbioru sentencji ZS_2 i sekwencji S_1.

Jeżeli:
— $ZS_1 \rightarrow S_1 + S_2$
— $S_1 \rightarrow (I_5, K_1) + (I_8, K_1) + (I_{11}, K_1)$
— $S_2 \rightarrow (I_5, K_2) + (I_8, K_2)$
to
— $S_3 \rightarrow ZS_2 + S_1$
— $ZS_2 \rightarrow S_1 + S_2$

Tak zdefiniowana reguła transformacji generuje kompozycję przedstawioną w postaci wykresy pianolowego na rys. 1.15. Przykładowa implementacja systemu generującego kompozycję na podstawie zdefiniowanej gramatyki formalnej została przedstawiona na listingu 5.9. Zdefiniowano zmienną I jako zbiór możliwych interwałów pomiędzy dźwiękami muzycznymi. Zmienna K przechowuje zbiór zdefiniowanych wcześniej kierunków interwałów. W macierzy komórkowej S znajdują się



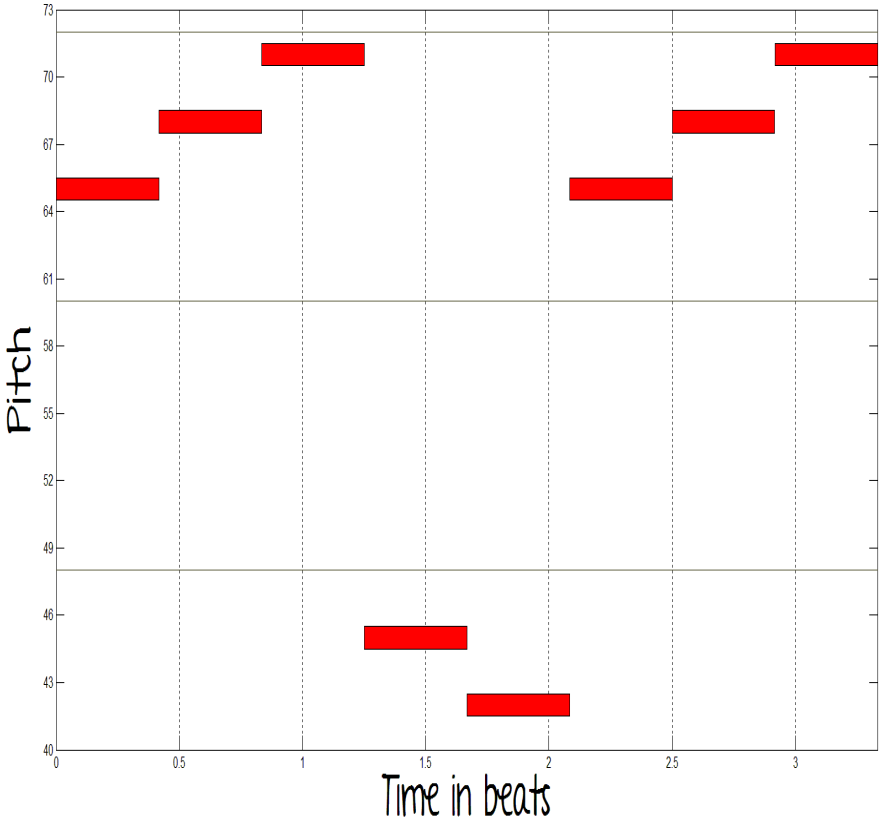
Rys. 1.14. Pianolowy wykres kompozycji wyprodukowanej wg reguły ZS_1. Opracowanie własne.

reguły definiujące sentencje (S_1 jako *S{1}* oraz S_2 jako *S{2}*). Macierz komórkowa *ZS{1}* zawiera regułę dla zbioru sentencji ZS_1 określoną w postaci *S{1}+S{2}*. Pod zmienną D przechowywana jest wartość dźwięku muzycznego, do którego odnoszą się zdefiniowane reguły.

W pętli przeszukującej zbiory sentencji budowane są wektory zawierające wartości dźwięków muzycznych, które będą następnie umieszczone w sentencjach muzycznych. W instrukcjach warunkowych wewnątrz pętli zbioru sentencji interpretowany jest kierunek interwału, co w konsekwencji powoduje odpowiednie dodanie lub odjęcie od wartości odniesienia D interwału znajdującego się w bieżącej parze (I_n, K_n). Na końcu pętli bieżącego zbioru sentencji następuje konkatenacja bieżącego wektora, zawierającego wyliczone wysokości dźwięków, z wektorem zawierającym wszystkie obliczone wcześniej wysokości dźwięku dla budowanej kompozycji. Wywołanie funkcji *createnmat* powoduje utworzenie macierzy nut zbudowanej na podstawie wektora obliczonych wysokości dźwięków.

Automaty skończone w zastosowaniach generowania muzyki

W poprzedniej części pracy opisane zostało pojęcie gramatyki formalnej wykorzystanej w systemie algorytmicznego komponowania muzyki. W wielu aspektach automaty skończone podobne są do gramatyki formalnej. Podczas gdy lingwiści używają gramatyk formalnych przede wszystkim do badania podstawowych praw generowania słów i zdań języka, naukowcy z dziedziny informatyki tworzą automaty określające prawa formowania łańcuchów tekstowych w celu budowania parserów dla kompilatorów i interpreterów określonego języka programowania. A więc w ogólności, zasady leżące u podstaw obu podejść są podobne, różnią się natomiast wykorzystywane w nich formalizmy. Jeżeli chodzi o muzykę, automaty skończone wydają się być podejściem bardziej skutecznym od gramatyk formalnych dla konstrukcji symbolicznych niskiego poziomu oraz krótkich sentencji muzycznych. Natomiast gramatyki formalne wydają się być bardziej odpowiednie od automatów w ujęciu zarządzania formami muzycznymi wyższych poziomów (np. całych utworów czy sentencji odtwarzanych symultanicznie) [85].

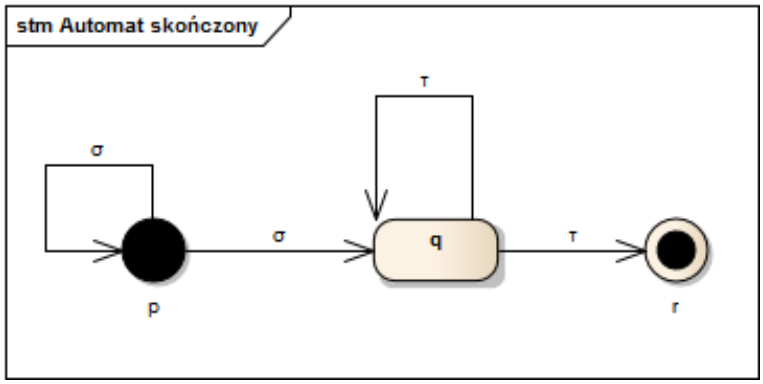


Rys. 1.15. Wykres pianolowy kompozycji wygenerowanej wg przykładowej reguły transformacji. Opracowanie własne.

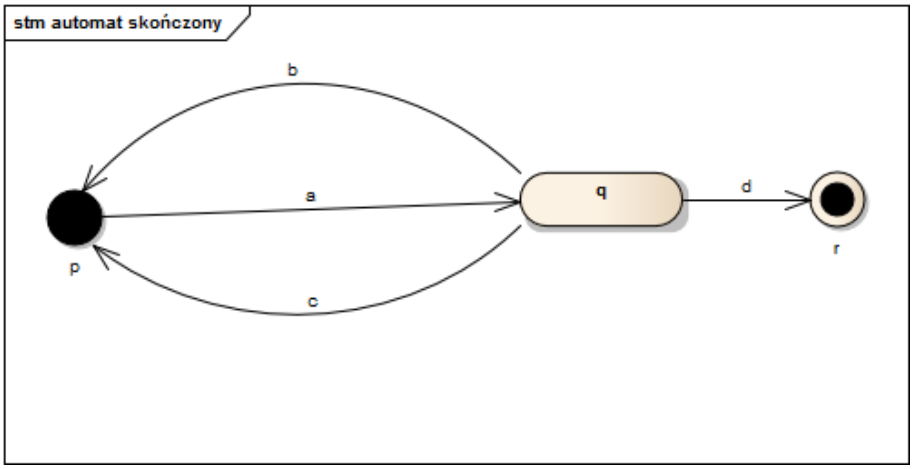
Automat skończony może być zdefiniowany jako $A = \{S,\ I,\ T,\ K\}$, gdzie:
— S - zbiór elementów zwanych stanami,
— I - podzbiór zbioru S, którego elementy nazywane są stanami początkowymi,
— T - podzbiór zbioru S, którego elementy nazywane są stanami terminalnymi,
— K - zbiór elementów złożonych zwanych krawędziami.
Krawędzie to złożone elementy wiążące dwa stany (elementy zbioru S) za pomocą połączenia oznaczonego pewnym operatorem. Przykładowo element zbioru K reprezentowany przez (p, σ, q) jest krawędzią od p do q połączoną za pomocą operatora σ. Podstawową ideą automatu skończonego jest traktowanie krawędzi (p, σ, q) jako transformacji stanu p do stanu q pod wpływem zadziałania akcji σ. W podanym przypadku stan p jest stanem początkowym, natomiast q jest terminalnym stanem krawędzi (p, σ, q).

Automaty definiują ścieżki transformacji z jednego stanu do innego. W pewnych automatach bardziej złożone ścieżki prowadzące od stanu początkowego p do stanu r mogą łączyć (obejmować) sekwencję krawędzi $e_1, e_2, ..., e_n$, tak, że stan p jest początkowym stanem krawędzi e_1, a stan r jest stanem terminalnym dla krawędzi e_n a stan terminalny pośrednich krawędzi jest stanem początkowym dla kolejnych.

Na rys. 1.16 przedstawiono przykład wizualizacji maszyny stanów za pomocą diagramu UML maszyny stanowej. Język UML zawiera wiele notacji graficznych służących do obrazowania, specyfikowania, tworzenia i dokumentowania modelo-



Rys. 1.16. Prosty automat skończony przedstawiony za pomocą diagramu UML maszyny stanów w programie Enterprise Architect. Opracowanie własne.



Rys. 1.17. Przykład automatu skończonego generującego sekwencje rytmiczne. Opracowanie własne.

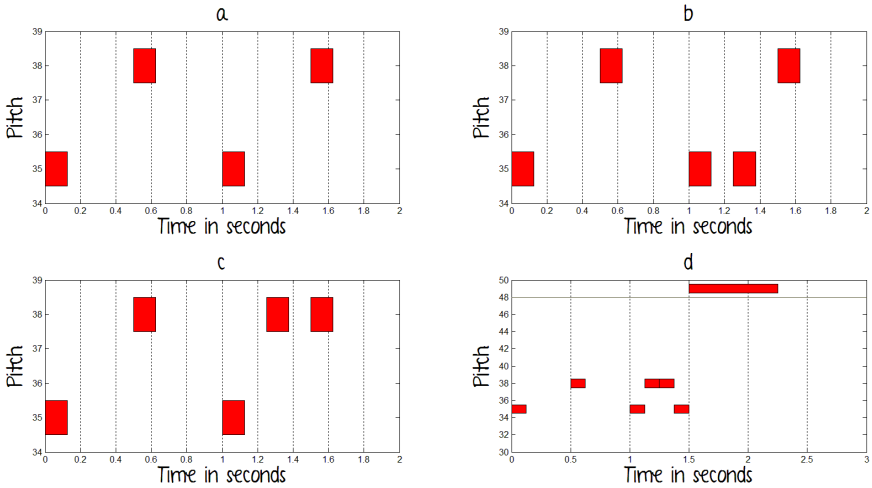
wanych systemów. W odniesieniu do opisu automatów skończonych wykorzystano diagram maszyny stanowej, ponieważ pozwala on jednoznacznie i kompletnie specyfikować zachowanie się automatu. W celu zapoznania się ze szczegółami dotyczącymi tej notacji języka UML, autor odsyła Czytelnika do rozdziału 5 w [105].

W zaprezentowanym na rys. 1.16 przykładzie automatu skończonego występują dwa operatory krawędzi: σ i τ. Stanem początkowym jest stan p, a stanem terminalnym stan r . Przykładowy automat skończony jest więc zdefiniowany następująco:

$$A = (\{p, q, r\}, \{p\}, \{r\}, \{(p, \sigma, p), (p, \sigma, q), (q, \tau, q), (q, \tau, r)\})$$

Automaty skończone są elastycznym narzędziem dla muzyków-naukowców wykorzystujących w swojej pracy komputer, ponieważ graficzna reprezentacja automatu pozwala łatwiej zdefiniować podstawowe koncepcje budowane w systemie algorytmicznego komponowania muzyki, bez wykorzystywania skomplikowanego zapisu matematycznego. Dzięki notacji graficznej można w czytelny sposób zdefiniować reguły funkcjonowania automatu, które następnie mogą zostać przeniesione do dowolnego języka programowania.

Jednym ze sposobów implementacji automatu skończonego w ujęciu komponowania muzyki jest etykietowanie krawędzi automatu za pomocą nut bądź krótkich



Rys. 1.18. Wykresy pianolowe sekwencji rytmicznych definiujących operatory krawędzi a, b, c i d. Opracowanie własne.

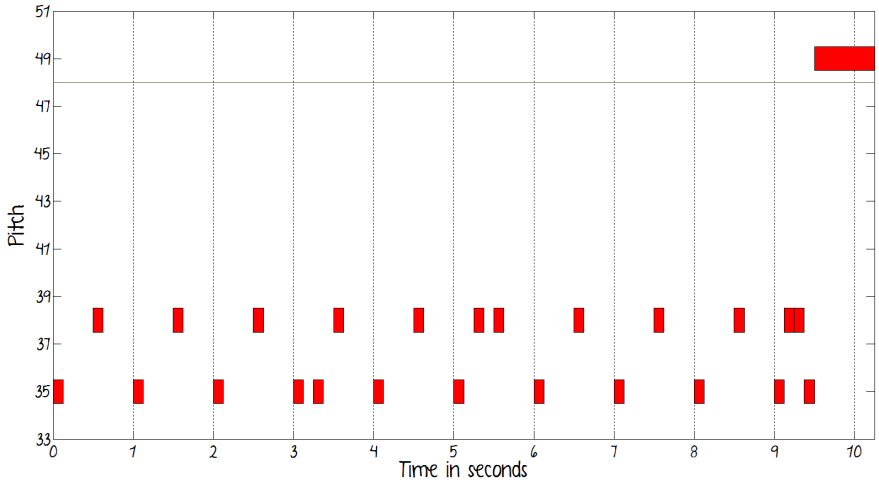
fragmentów muzycznych [85]. Sposób ten jest wykorzystywany w przypadku posługiwania się grafem skierowanym, który opisuje zasadę działania danego automatu. Autor w tym miejscu zaproponował diagram UML maszyny stanowej, w którym krawędzie zostały opisane literami, zdefiniowanymi w dalszej części jako fragmenty (sekwencje) rytmiczne.

W celu przedstawienia wykorzystania automatu skończonego w systemach algorytmicznego komponowania muzyki, posłużono się automatem, którego budowa przedstawiona została na rys. 1.17. Operatory krawędzi a, b, c i d zdefiniowano jako muzyczne sekwencje rytmiczne, przedstawione na rys. 1.18 w postaci wykresów pianolowych. Prezentowany automat skończony jest więc zdefiniowany następująco:

$$A = (\{p, q, r\}, \{p\}, \{r\}, \{(p, a, q), (q, b, p), (q, c, p), (q, d, r)\})$$

Sekwencje rytmiczne definiujące wymienione operatory krawędzi, przedstawione w postaci wykresów pianolowych, zostały zaimplementowane jako macierze nut w środowisku Matlab. Przykładowa implementacja została pokazana na listingu 5.10. Początkowo za pomocą wbudowanej w narzędzie *MIDI* Toolbox funkcji *createnmat*, utworzono macierz nut (*basepattern*) zawierającą 16 dźwięków o długości 0.125 sekundy i dynamice 127, odgrywanych w kanale *MIDI* o numerze 10 (zarezerwowanym dla instrumentów perkusyjnych). Następnie na podstawie macierzy *basepattern* wygenerowano cztery macierze definiujące operatory krawędzi automatu skończonego: *pattern1* dla operatora a, *pattern2* dla b, *pattern3* dla c oraz *pattern4* dla d.

W każdej ze zdefiniowanej macierzy nut początkowo wszystkie dźwięki posiadały wartości wysokości równe 0 (funkcja *zeros*, wywołana jako argument funkcji *createnmat*, listing 5.10). Dopiero za pomocą przypisania do odpowiednich wierszy kolumny 4 (wysokości dźwięków) zdefiniowanych wektorów wartości wysokości, otrzymano sekwencje rytmiczne, odgrywane za pomocą instrumentów perkusyjnych o numerach: 35 (Bass Drum 2, kocioł centralny), 38 (Snare Drum 1, werbel) i 49 (Crash Cymbal 1, talerz Crash). Dodatkowo dla *pattern4* wykorzystano funkcję shift (wbudowaną w *MIDI* Toolbox), za pomocą której przesunięto (zwiększono) długość 13-stego dźwięku macierzy (parametr 'dur') o wartość równą trzykrotności podsta-

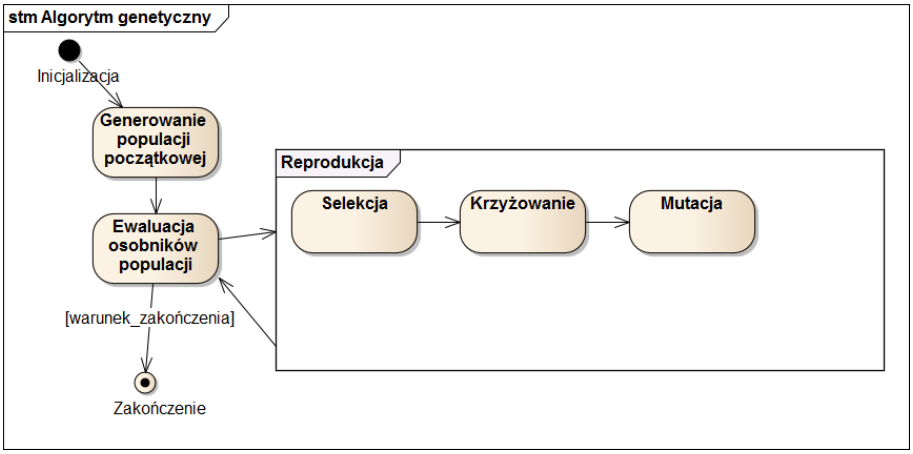


Rys. 1.19. Wykres pianolowy przykładowej sekwencji rytmicznej wygenerowanej przez zaimplementowany automat skończony. Opracowanie własne.

wowej zdefiniowanej długości dźwięku, równej 0.125 sekundy (parametr 'sec'). Pełna gama instrumentów perkusyjnych została zdefiniowana w [1].

Na listingu 5.11 przedstawiono przykładową implementację (w środowisku Matlab) automatu skończonego generującego sekwencje rytmiczne na podstawie zdefiniowanych i zaimplementowanych wcześniej operatorów w postaci macierzy nut *pattern1*, *pattern2*, *pattern3* i *pattern4*. Zmienna *state* symbolizuje stan, w którym znajduje się aktualnie automat. Zmienna *sequence* przechowuje generowaną przez automat sekwencje rytmiczną w postaci macierzy nut. Zachowanie automatu opisane zostało wewnątrz pętli *while*. Instrukcje sterujące, w których sprawdzana jest wartość zmiennej *state*, zaimplementowane zostały adekwatnie do automatu, którego zachowanie przedstawiono na rys. 1.17, przy czym wartość 1 zmiennej *state* odpowiada stanowi początkowemu p, wartość 2 stanowi q, a wartość 3 stanowi terminalnemu r.

Automat generuje sekwencję rytmiczną (listing 5.11) poczynając od stanu p (*state*=1), w którym następuje złączenie macierzy *sequence* z macierzą *pattern1*. Jeżeli przejście do stanu p nastąpiło ze stanu q, wówczas przed złączeniem macierzy nut, dokonuje się dopasowania sekwencji w dziedzinie czasu, tzn. przesunięcia wartości rozpoczynania się dźwięków macierzy *pattern1* o wartość czasu zakończenia wybrzmiewania ostatniego dźwięku macierzy *sequence*. W stanie q (*state*=2) dokonuje się wyboru (poprzez losowanie) operatora krawędzi, z pośród dostępnego zbioru operatorów b, c i d. Jak wspomniano wcześniej, każdy z wymienionych operatorów krawędzi został zdefiniowany poprzez odpowiednią macierz nut. W losowaniu wykorzystano mechanizm wyboru kumulatywnego, przy czym prawdopodobieństwo wylosowania każdego z dostępnych operatorów jest równe. Przejście automatu do stanu terminalnego r powoduje zakończenie generowania sekwencji rytmicznej. Wykres pianolowy przykładowej wygenerowanej przez automat sekwencji rytmicznej przedstawiono na rys. 1.19.



Rys. 1.20. Typowy przebieg algorytmu genetycznego przedstawiony za pomocą diagramu maszyny stanów UML. Opracowanie własne.

Algorytmy genetyczne w systemach algorytmicznego komponowania muzyki

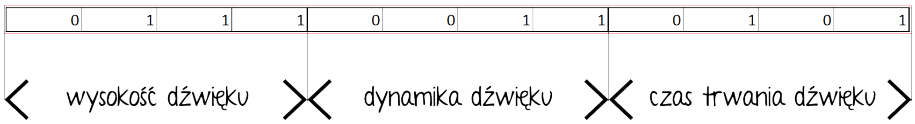
Algorytmy genetyczne są mechanizmem stosowanym w rozwiązywaniu problemów i wg. Goldberga stanowią "*odporną metodę poszukiwania w skomplikowanych przestrzeniach... i... nieskomplikowane, a przy tym potężne narzędzie poszukiwania lepszych rozwiązań. Co więcej, są wolne od zasadniczych ograniczeń nakładanych przez mocne założenia o przestrzeni poszukiwań (takie jak ciągłość, istnienie pochodnych, jednomodalność funkcji celu (funkcja posiada tylko jedno maksimum) i temu podobne).*" [44].

Mechanizm ewolucji i dziedziczności zaimplementowany w algorytmach generycznych jest stosowany zwykle do znajdowania optymalnych rozwiązań w inżynierii. Algorytmy genetyczne opisują problemy, w których może istnieć potencjalnie wiele alternatywnych rozwiązań, lecz trudno jest ocenić ich przydatność dopóki nie zostaną zaimplementowane i przetestowane.

Podstawowy przebieg algorytmu genetycznego przedstawiono za pomocą diagramu maszyny stanów UML na rys. 1.20. W stanie inicjalizacji następuje konfigurowanie parametrów mechanizmu algorytmu genetycznego określających m.in wielkość populacji (liczbę osobników), liczba generacji osobników oraz współczynniki operatorów genetycznych omówionych w dalszej części.

Po inicjalizacji następuje stan generowania populacji początkowej (zwykle w sposób losowy), zawierającej proponowane rozwiązania w postaci osobników populacji. W ujęciu algorytmów komponowania muzyki osobnikiem populacji może być dźwięk muzyczny, fragment utworu bądź cały utwór muzyczny (w zależności od kontekstu problemu).

Niezależnie od podejścia osobniki populacji początkowej są oceniane (stan "Ewaluacja osobników populacji, rys. 1.20) pod względem ich przydatności (przystosowania) do warunków zadania (środowiska). Mechanizm oceny nazywa się funkcją przystosowania i jest najważniejszym, a także najtrudniejszym do realizacji (z punktu widzenia finalnego rozwiązania) elementem algorytmu genetycznego. Wartość funkcji przystosowania jest bowiem wskaźnikiem dla mechanizmu odpowiedzialnego za wybór najlepszych osobników do stanu reprodukcji.



Rys. 1.21. Przykład 12-sto bitowego łańcucha kodującego trzy parametry dźwięku muzycznego: wysokość, dynamikę i czas trwania. Opracowanie własne.

Stan "Reprodukcja" przedstawiony na rys. 1.20 jest stanem złożonym z trzech stanów prostych. W stanie "Selekcja" algorytm dokonuje wyboru najlepszych osobników, które zostaną nagrodzone w postaci dopuszczenia ich do etapu rozmnażania (stan "Krzyżowanie"). Najlepsze osobniki są łączone w pracy, z których za pomocą operatora krzyżowania tworzone są osobniki potomne. Osobniki potomne zawierają częściowe informacje od obu rodziców. Najczęściej z dwojga rodziców tworzone są dwa osobniki potomne. Informacje zawarte w osobnikach potomnych mogą być częściowo zmienione w sposób losowy poprzez zastosowanie operatora mutacji (stan "Mutacja", rys. 1.20). Dzięki mutacji algorytm genetyczny może do istniejącej puli rozwiązań dodawać nowe, które mogą pogorszyć lub polepszyć wartość przystosowania osobnika populacji.

Nowe osobniki są lokowane w populacji. Stare, które nie zostały wylosowane w stanie "Selekcja", najczęściej giną. Osobniki potomne są następnie oceniane pod względem ich przystosowania do środowiska (użyteczności) w stanie "Ewaluacja osobników populacji". Algorytm genetyczny może działać w sposób ciągły, najczęściej jednak podaje się pewien "warunek zakończenia" (rys. 1.20) algorytmu. Warunkiem tym może być np. liczba generacji, odpowiedni poziom wartości funkcji przystosowania czy interwał czasu, w którym wartość funkcji przystosowania populacji nie zmienia się.

Istotną rolę w efektywności algorytmu genetycznego odgrywa metoda kodowania informacji genetycznej zawartej w osobnikach. Najprostszą metodą kodowania informacji genetycznej jest kodowanie binarne przedstawione m.in w pracy [44], a w ujęciu tematu algorytmicznego komponowania muzyki w pracy [85]. W kodowaniu binarnym każdy osobnik populacji jest prezentowany jako ciąg bitowy o skończonej długości. Pojedynczy bit lub grupa bitów reprezentuje różne atrybuty osobnika. Na rys. 1.21 przedstawiono przykład kodowania binarnego dla dźwięku muzycznego określonego przez parametry wysokości dźwięku, dynamiki oraz czas trwania (długość). Grupy bitów przedstawione w przykładowym dwunastoelementowym łańcuchu binarnym, są związane z rzeczywistymi wartościami odpowiednio wysokości, dynamiki i długości dźwięku muzycznego. Podczas fazy reprodukcji algorytmu genetycznego osobniki krzyżujące się wymieniają grupy bitów pomiędzy sobą, a podczas mutowania najczęściej w sposób losowy zmieniają wartości bitu na konkretnej pozycji.

Selekcja podgrupy osobników z populacji w algorytmach genetycznych może odbywać się na kilka sposobów. W ogólnym przypadku selekcja bazuje na kombinacji wartości funkcji przystosowania osobników oraz schemacie probabilistycznym. Jednym z podstawowych mechanizmów selekcji jest selekcja poprzez próbkowanie stochastyczne [47], w którym każdy z osobników jest związany z różnym odcinkiem odłożonym na linii, której długość wyznacza maksymalną wartość funkcji przystosowania. Długość segmentu odpowiada wyliczonej wartości funkcji przystosowania danego osobnika.

Innym mechanizmem jest selekcja lokalna, w której każdy osobnik osadzony jest w wymuszonym środowisku zwanym lokalnym sąsiedztwem. Osobniki umieszczone w takim środowisku mogą oddziałują jedynie na inne osobniki z danego regionu. Sąsiedztwo jest natomiast zdefiniowane przez strukturę w jakiej populacja jest dystrybuowana. Osobniki sąsiadujące ze sobą traktowane są jako grupa potencjalnych partnerów w procesie reprodukcji. Struktura grupy może przybierać postać przestrzeni jedno-, dwu- lub wielowymiarowej. Wewnątrz grupy następuje dobór osobników do reprodukcji np. poprzez wybór najlepiej dostosowanego osobnika czy też w sposób losowy z wartością prawdopodobieństwa wylosowania proporcjonalną do wartości funkcji przystosowania. Symulacje z wykorzystaniem struktury sąsiedztwa przedstawione zostały w pracy [100].

Automaty komórkowe - muzyczny kontekst użycia

W poprzedniej części pracy omówiono zastosowanie algorytmów genetycznych w procesie generowania muzyki przez maszynę. Innym podejściem bazującym na mechanizmach ewolucji są automaty komórkowe. Mechanizm automatu komórkowego jest elastycznym narzędziem do modelowania systemów dynamicznych z przestrzenią elementów dyskretnych oraz dyskretnymi interwałami czasowymi, w których następuje aktualizacja modelu systemu.

Automat komórkowy, zaproponowany przez von Neumanna i Ulana w latach 60-tych dwudziestego wieku [33], zbudowany jest z macierzy elementów nazywanych komórkami, dla których określone są odpowiednie reguły ewolucji. Macierz komórek przybiera postać pewnej d-wymiarowej siatki natomiast reguły ewolucji determinują zachowanie się automatu w czasie, poprzez aktualizację komórek macierzy automatu w sposób symultaniczny, w równych interwałach czasowych.

Formalnie automat komórkowy definiowany jest jako czwórka:

$$A = (\alpha, S, N, f)$$

Przez oznacza się regularną i uporządkowaną siatkę złożoną z komórek c, których budowa zależy od rozmiaru przestrzeni oraz kształtu komórki. S oznacza natomiast skończony zbiór stanów, jaki może przyjąć komórka c. Oznaczenie N określa skończony zbiór sąsiadów, spełniających warunek:

$$\forall c \in N, \forall r \in \alpha : r + c \in \alpha$$

Funkcję przejścia f, definiującą reguły ewolucji automatu w kolejnych iteracjach (łącznie z dynamiką tych przejść) definiuję się jako:

$$f : S^m \rightarrow S$$

W celu pokazania możliwości jakie dają automaty komórkowe w aspekcie generowania muzyki, autor przedstawi przykład implementacji modelu składającego się z dwóch automatów generujących kompozycję w postaci plików *MIDI* z wykorzystaniem *MIDI* Toolbox. Na potrzeby przykładu przyjęto, że siatki $_1$ i $_2$ obu automatów są siatkami dwuwymiarowymi o rozmiarach n wierszy na n kolumn. Najbliższe otoczenie komórki c zdefiniowano jako tzw. sąsiedztwo Moore'a, w którym komórkę centralną otaczają komórki sąsiednie graniczące z daną krawędzią lub wierzchołkiem komórki centralnej. Reguły odnoszące się do obliczania wartości (stanów) komórek

w prezentowanym przykładzie pochodzą z jednego automatu komórowego, zaprezentowanego przez brytyjskiego matematyka Johna Conwaya, o nazwie "Gra w życie". Zdefiniowano w nim następujące reguły:

— jeżeli w najbliższym otoczeniu komórki znajduje się dokładnie 3 żywych sąsiadów, to komórka ta staje się żywa w następnej jednostce czasu (rodzi się w kolejnej generacji komórek),
— jeżeli żywa komórka posiada 2 lub 3 żywych sąsiadów pozostaje nadal żywa w kolejnej generacji komórek; w przeciwnym wypadku ginie z samotności bądź zatłoczenia.

Na listingu 5.12 zaprezentowana została implementacja głównego skryptu modelu wykonanego w środowisku Matlab. W części początkowej ustalane są parametry definiujące rozmiar siatek automatu (*siatkaWymiar*), liczbę generacji komórek (*generacje*) oraz specjalne tablice (*lut* i *lut2*), wykorzystywane przy analizie i interpretacji rozmieszczenia komórek na siatkach automatu. Tablice *lut* i *lut2* zawierają wartości dopuszczalnych dźwięków, które mogą być wykorzystane przy tworzeniu kompozycji.

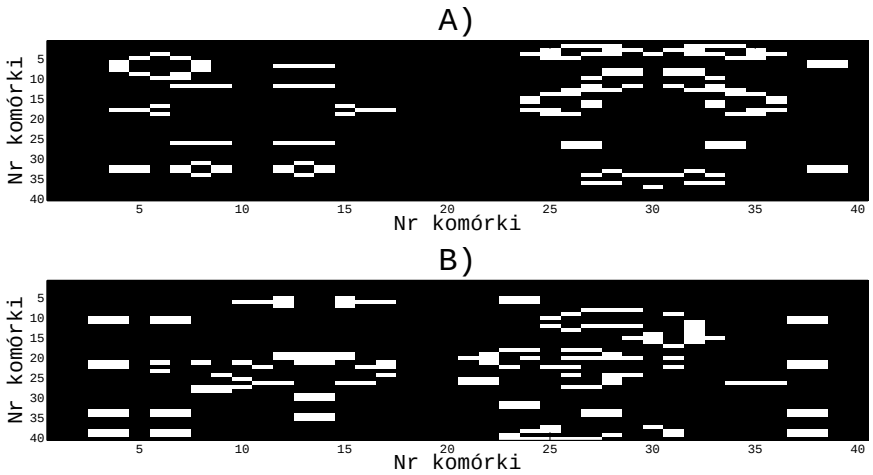
W dalszej części tworzone są wyzerowane siatki automatów (*siatka* i *siatka2*), na które w sposób losowy nanoszone są żywe komórki. Następnie tworzone są tymczasowe siatki, które przechowują wartości komórek dostępne w kolejnej generacji automatu. W ramach skryptu zdefiniowane zostały 3 pętle: pętla zewnętrzna, w której generuje się kolejne generacje siatek komórek oraz dwie wewnętrzne pętle, w których analizowane jest sąsiedztwo komórki $c_{i,j}$ oraz obliczany jest stan komórki w nowej generacji, bazując na regułach "Gry w życie".

Analiza sąsiedztwa Moore'a została zaimplementowana w postaci funkcji *skanowanie*, przedstawionej na listingu 5.13. Warunki sprawdzające otoczenie komórki uwzględniają jej lokalizację w granicznych wierszach i kolumnach siatki komórek automatu. Parametrami wejściowymi funkcji są indeksy położenia komórki na siatce oraz siatka zawierająca komórki bieżącej generacji. Funkcja *skanowanie* zwraca liczbę żywych komórek w obrębie analizowanego otoczenia.

Po uzyskaniu informacji na temat liczby żywych komórek w sąsiedztwie komórki $c_{i,j}$ wywoływana jest funkcja *reguly*, która oblicza nowy stan komórki $c_{i,j}$ w kolejnej generacji automatu. Na listingu 5.14 przedstawiono implementację funkcji *reguly* w środowisku Matlab, która oblicza nową wartość komórki w kolejnej generacji automatu. Funkcja pobiera argumenty wejściowe w postaci zmiennej przechowującej liczbę żywych komórek w sąsiedztwie komórki $c_{i,j}$ oraz wartość (stan) tej komórki w poprzedniej generacji automatu.

W zewnętrznej pętli głównego skryptu (listing 5.12) opisującego przykładowy model automatu komórkowego generującego kompozycje muzyczne występuje wywołanie funkcji *generowanieKompozycji*, której implementacja została przedstawiona na listingu 5.15. Funkcja pobiera parametry wejściowe w postaci:

— macierzy nut (*kompozycja*), do której należy dodać wygenerowany na podstawie siatki fragment kompozycji,
— macierzy reprezentującej siatkę komórek automatu (*siatka*),
— wartości przesunięcia w czasie (*offset*) wzorca muzycznego wchodzącego w skład kompozycji,
— tablicy typu look-up (*lut*), w której znajdują się wcześniej zdefiniowane wysokości dźwięków możliwe do wykorzystania podczas generowania kompozycji,
— numeru kanału *MIDI* (*kanalMidi*), w którym ma być odtwarzany wzorzec muzyczny umieszczony w kompozycji,



Rys. 1.22. Przykładowe siatki automatów komórkowych generujących składowe partie instrumentalne na podstawie tablicy *lut2* (A) i *lut* (B). Opracowanie własne.

— długości (*dlugosc*) dźwięków muzycznych występujących we wzorcu,
— przyrostu (*delta*) wartości przesunięcia w czasie kolejnych fragmentów muzycznych występujących we wzorcach.

Funkcja *generowanieKompozycji* analizuje siatkę komórek automatu kolumna po kolumnie. Za pomocą wbudowanej w *MIDI* Toolbox funkcji *createnmat* tworzony jest wzorzec muzyczny, który zawiera dźwięki wybrane za pomocą przemnożenia tablicy *lut* z wartościami stanu komórek i-tej kolumny. Ponieważ komórka może przybierać jedynie wartości 0 lub 1, to wymnożenie kolumny komórek z tablicą *lut* spowoduje utworzenie wektora zawierającego tylko te dźwięki, dla których istnieją żywe komórki w kolumnie siatki automatu. Dla dźwięków utworzonego w ten sposób wzorca (*buffor*) ustawiana jest długość dźwięków (*dlugosc*), głośność (w tym wypadku stała wartość 100) oraz kanał *MIDI* (*kanalMidi*), w którym wzorzec będzie odtwarzany. Utworzona w ten sposób macierz nut reprezentująca wzorzec jest następnie filtrowana, aby usunąć "luki" (wiersze) w macierzy, dla których wysokość dźwięku jest równa 0 (wynik przemnożenia pozycji w tablicy *lut* przez wartość komórki automatu). Następnie dokonywane jest przesunięcie wzorca w czasie o zadaną wartość (*offset*) wyrażoną w sekundach, w celu umieszczenia go w kompozycji w miejscu odpowiednim pod względem rytmicznym. W prezentowanym przykładzie umieszczenie kolejnych wzorców reprezentujących kolumnę siatki odbywa się z przesunięciem (*delta*) 1.2 sekundy, a długość nut zdefiniowano na 0.2 sekundy. Ostatnim krokiem jest zwiększenie wartości przesunięcia o zadaną wartość *delta*.

Proces generowania kompozycji w prezentowanym przykładzie został ograniczony do 8 generacji komórek automatów. Na rys. 1.21 zostały przedstawione przykładowe siatki komórek automatów generujących składowe kompozycji muzycznej na podstawie tablic *lut* i *lut2* w ostatniej (ósmej) iteracji algorytmu. Wynikowa kompozycja składa się z dwóch partii instrumentalnych przy czym automat komórkowy działający w ramach siatki z rys. 1.21A generuje partię instrumentu dla przebiegu melodii, natomiast siatka z rys. 1.21B opisuje przebieg partii instrumentów perkusyjnych.

Przedstawione podejście może zostać poszerzone o dodatkowe automaty komórkowe odpowiedzialne za generowanie partii instrumentalnych. Kolejnym rozwinięciem może być zastosowanie w kontekście komórek automatu bardziej złożonych reguł umierania i przeżywania, które nawiązywałyby do teorii muzyki.

Sztuczne sieci neuronowe generujące muzykę

Jak podano w [85] jedną z definicji muzyki jest traktowanie jej jako aktywności intelektualnej, której główne zadania obejmują zdolności do rozpoznawania wzorców słuchowych oraz wyobrażania sobie ich modyfikacji przez różne podejmowane akcje. Wymienione zdolności wymagają wyrafinowanych mechanizmów pamięci, związanych za równo ze świadomymi manipulacjami konceptu muzycznego jak i podświadomego dostępu do milionów połączonych w sieć neuronów. Zgodnie z tą definicją, emocjonalne reakcje słuchacza muzyki wynikają z pewnej formy aktywności intelektualnej.

Istnieje wiele przesłanek potwierdzających fakt, że aktywność mózgu mierzona za pomocą *EEG*, dostarcza bogate źródło informacji o procesach przetwarzania muzyki przez człowieka. Dla przykładu Saiwaki i inni w [88] dokonali statystycznej analizy przepływu informacji pomiędzy kanałami EEG podczas słuchania muzyki. Janata natomiast badał w [79] aktywność fal mózgowych w kontekście oczekiwanego naruszenia harmonii muzycznej. Z kolei Tillmann i inni w [94] badali aktywność kory czołowej podczas stymulacji sekwencjami akordów muzycznych zakończonymi toniką lub akordem przetransponowanym o pół tonu w dół lub górę.

Naukowcy na przestrzeni dekad opracowali wiele uproszczonych modeli ludzkiego mózgu tworząc sztuczne sieci neuronowe. Jednym z pierwszych modeli sztucznej sieci neuronowej był perceptron Rosenblatta [54], zaproponowany w latach 50-tych dwudziestego wieku. W dalszej kolejności pojawiło się szereg wariacji perceptronu, jednakże jego podstawowa budowa w sposób matematyczny może być opisana w postaci funkcji aktywacji unipolarnej:

$$y = \begin{cases} 1, & \varphi > 0 \\ 0, & \varphi \leq 0 \end{cases}$$

lub bipolarnej

$$sgn(\varphi) = \begin{cases} +1, & \varphi > 0 \\ -1, & \varphi \leq 0 \end{cases}$$

gdzie

$$\varphi = \sum_{n=0}^{j} w_n u_n - \theta.$$

W podanym modelu: w_n oznacza wagę n-tego połączenia wstępującego do elementu, u_n określa wartość n-tego wejścia neuronu oraz θ wyznacza wartość progową funkcji aktywacji neuronu. Często w zastosowaniach praktycznych ustala się dla każdego elementu przetwarzającego wartość progu $\theta = 0$, uzyskując tym samym jednolite wzory określające zmiany wag i progów aktywacji w procesie dostrajania parametrów sieci (uczenia sieci).

W wersji podstawowej perceptron składa się jedynie z warstwy wejściowej i wyjściowej. W obrębie warstwy wyjściowej nie istnieją wzajemne połączenia

pomiędzy elementami, dlatego też każdy z elementów można traktować jako niezależną sieć o $m+1$ wejściach i jednym wyjściu. W sieciach wielowarstowowych wyjście neuronu jest połączone z wejściami innych neuronów. Projektowanie takich sieci, a w szczególności sposób rozmieszczenia połączeń pomiedzy neuronami, definiuje zachowanie się sieci. Architektury sztucznych sieci neuronowych zakładają zarówno jednokierunkowość przepływu informacji jak i sprzężenia zwrotne do wcześniejszych warstw sieci. Sieci wielowarstwowe znajdują wiele zastosowań w obszarze m.in szeroko pojętej klasyfikacji, również w muzyce, gdzie szeroko stosowane są także sieci Kohonena (mapy samoorganizujące się [81]).

Pojęcie uczenia sztucznej sieci neuronowej dotycz w ogólności prezentowania tej sieci serii przykładowych prób definiujących problem do rozwiązania oraz przykład rozwiązania dla każdej próby problemu. Odpowiednia liczba danych treningowych powinna spowodować wykrywanie przez sieć podstawowych aspektów rozwiązań. Zdolność ta związana jest z wykrywaniem przez sieć podobieństwa przypadków problemów pojawiających się do rozwiązania z problemami, którymi sieć była trenowana.

1.3. Komputerowe systemy generowania muzyki

W niniejszym rozdziale zaprezentowane zostały znane z literatury światowej rozwiązania z dziedziny algorytmicznego komponowania muzyki. Autor dokonał próby pozycjonowania rozwiązań znanych z literatury oraz własnej koncepcji systemu w modelu porównywania algorytmów komponowania muzyki zaproponowanym przez Wollera w [104]. Znane z literatury rozwiązania zostały również sklasyfikowane pod względem rodzaju reguł komponowania wykorzystywanych w wybranym podejściu.

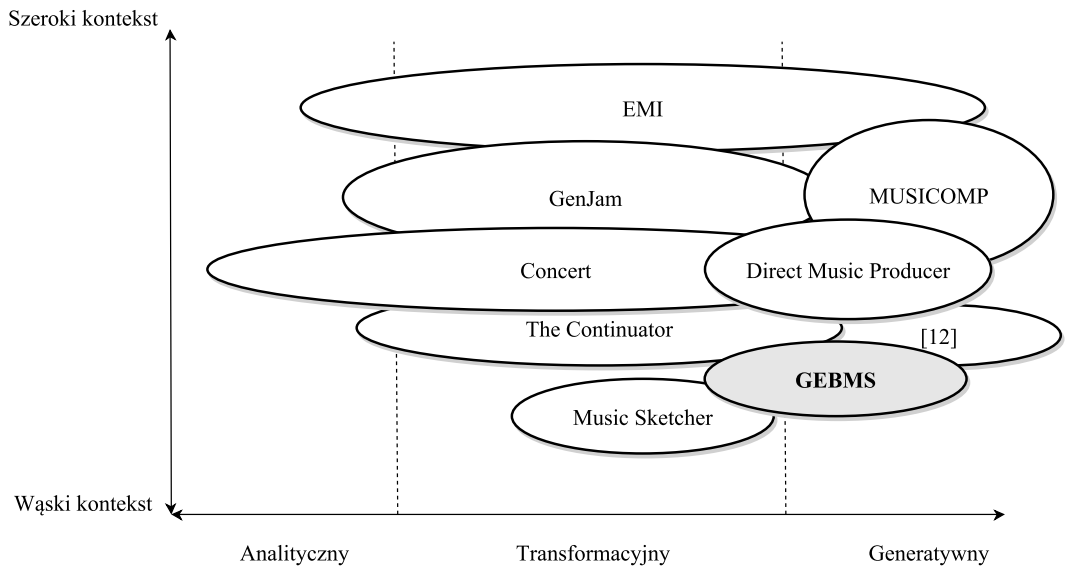
Pozycjonowanie systemów algorytmicznego komponowania muzyki w modelu Woollera

Od dekad w literaturze światowej pojawiają się nowe lub ulepszone rozwiązania w dziedzinie algorytmicznego komponowania muzyki przez maszynę. Rozwiązania te ze względu na charakter dziedziny rozważań oraz subiektywny charakter oceny jakości muzyki generowanej przez algorytmy, są nierzadko trudne do porównania. W tym aspekcie ciekawym narzędziem jest model zaproponowany przez Woollera i innych w [104]. Przytoczona praca pozwala autorom różnych systemów algorytmicznego komponowania muzyki na lokalizację swoich rozwiązań na tle innych.

Wg Woollera proces komponowania muzyki jest kontrolowany przez algorytm, który w ogólności może posiadać znamiona mechanizmu analitycznego, transformacyjnego i generatywnego. Oprócz właściwości mechanizmu proponowany model pozycjonowania zakłada również zmienną szerokość kontekstu muzycznego wykorzystywanego podczas generowania muzyki.

Na rys. 1.23 przedstawiona została płaszczyzna modelu Woollera opisana za pomocą funkcji właściwości algorytmu generującego muzykę (analityczna, transformacyjna, generatywna) oraz szerokości kontekstu muzycznego wykorzystywanego podczas generowania kompozycji. Na płaszczyźnie ulokowano analizowane rozwiązania oraz autorską propozycję systemu generowania muzyki.

Płaszczyzna modelu Woollera jest opisana przez funkcję szerokości kontekstu muzycznego względem głównej właściwości algorytmu wykorzystanego do kompo-



Rys. 1.23. Płaszczyzna pozycjonowania algorytmów komponowania muzyki zaproponowana przez Woollera z ulokowanym autorskim rozwiązaniem (GEBMS). Opracowanie własne.

nowania muzyki. Zakres szerokości kontekstu zawiera się w umownym przedziale od wąskiego do szerokiego, natomiast zakres właściwości algorytmu został podzielony na trzy obszary: analityczny, transformacyjny i generatywny.

Wg pracy [104] analityczny charakter algorytmu komponującego muzykę polega na redukcji potencjalnego rozmiaru danych wejściowych (struktury muzycznej) oraz ekstrahowaniu specyficznych właściwości struktury muzycznej, które później mogą być wykorzystane w procesie komponowania. Przykład takiego algorytmu może być opisany jako proces przetwarzania muzyki, który na wejściu otrzymuje zbiór sekwencji muzycznych a na wyjściu - zbiór nut wraz z ich relacjami.

Transformacyjny charakter algorytmu polega na zmianie informacji bez istotnej zmiany reprezentacji struktury wejściowych danych muzycznych (w tym także ich rozmiaru). Przykładem takiego algorytmu może być proces polegający na transpozycji poszczególnych nut w sekwencji muzycznej lub zmiana porządku nut we frazie.

Generatywny algorytm komponowania muzyki jest procesem polegającym na wytwarzaniu złożonych muzycznych sekwencji wyjściowych z wykorzystaniem zredukowanych danych wejściowych. Przykładem może być proces, który jako dane wejściowe pobiera pojedyncze nuty z informacją na temat ich rozmieszczenia we frazie, a na wyjściu tego procesu generowana jest sekwencja muzyczna złożona z wylosowanych na wejściu nut.

W [104] zdefiniowany został również kontekst muzyczny jako informacja otaczająca wpływająca na obliczenia algorytmu, a tym samym na cały system algorytmicznego komponowania muzyki. Obrazowo szerokość kontekstu muzycznego może być przedstawiona jako nuta ulokowana na pięciolinii wraz z kołem narysowanym wokół tej nuty, definiującym bezpośrednie jej sąsiedztwo. Zakres kontekstu jest w tym wypadku związany z otaczającym nutę kołem. Im większy jest promień koła otaczającego nutę, tym szerszy jest kontekst muzyczny - więcej nut leży w bezpośrednim sąsiedztwie analizowanej nuty. Autorzy w [104] przyznają jednak, że

szerokość kontekstu muzycznego nie zależy jedynie od liczby nut ulokowanych w bezpośrednim sąsiedztwie analizowanej nuty, ale także od liczby wymiarów (parametrów) definiujących nutę. W tym znaczeniu należy zauważyć, że algorytm biorący pod uwagę cztery parametry nuty (dla przykładu: wysokość, głośność, czas trwania oraz czas rozpoczęcia) będzie miał szerszy kontekst muzyczny niż podobny algorytm analizujący jedynie wysokość dźwięku.

Mając na uwadze przedstawione definicje autor dokonał próby ulokowania swojego rozwiązania na tle innych systemów, co zostało przedstawione na rys. 1.23. W dalszej części tego rozdziału dokonano analizy podobnych rozwiązań ulokowanych przez autora na płaszczyźnie modelu Woollera.

Jednym z pierwszych rozwiązań w dziedzinie algorytmicznego komponowania muzyki przy użyciu maszyny był system MUSICOMP autorstwa Hillera i Isaacsona, zaprezentowany w [61]. W swoich eksperymentach autorzy skupili się na modyfikacjach stochastycznej muzyki przedstawionej jako biały szum. Modyfikacje struktury muzycznej wykorzystywały reguły harmonii, a część z przetwarzanych struktur muzycznych była transformowana poprzez dodanie akordów lub reguły ograniczające liczbę możliwych kolejnych powtórzeń danej frazy muzycznej. Podczas eksperymentów z MUSICOMP wykorzystany został proces Markowa, kontrolujący wystąpienia dysonansów i konsonansów poprzez wpływ na funkcję dystrybucji prawdopodobieństwa ich pojawienia się w wynikowej frazie. Wg Woollera wykorzystanie systemu tonalnego w modelu Hillera i Isaacsona poszerza kontekst muzyczny, w którym porusza się algorytm odpowiedzialny za generowanie kompozycji. W ogólności eksperymenty opisane w [61] zostały przez Woollera zdefiniowane jako generatywny system algorytmicznego komponowania muzyki wykorzystujący szeroki kontekst muzyczny w procesie generowania muzyki (rys. 1.23).

Innym rozwiązaniem ulokowanym przez Woollera w [104] był projekt systemu autorstwa Davida Cope'a o nazwie EMI (Experiments in Musical Intelligence) przedstawiony w [40] i [34]. Eksperymenty opisane przez Cope'a skupiały się na wykorzystaniu rekombinacji i gramatyki opisujących kreatywny proces tworzenia muzyki. EMI analizowała wejściowe struktury muzyczne, manualnie dostarczane i przygotowane przez Cope'a. Na wyjściu systemu za pomocą sieci ATN (Augmented Transition Networks) generowane były frazy oraz struktura muzyczna wysokiego poziomu (forma), natomiast za pomocą sieci MATN (Micro-Augmented Transition Networks) generowany był poziom organizacji pojedynczych dźwięków. W ramach projektu EMI utworzony został złożony system gramatyki - SPEAC. System SPEAC został utworzony aby dostarczać ciekawych motywów i łączyć je w sposób gładki i z zachowaniem logicznego uporządkowania [37]. Aplikacja działała jako bazodanowy słownik muzycznych próbek, które były łączone zgodnie z logiczną składnią języka zdefiniowanego przez Cope'a. Wyjściowe formy muzyczne poprzez sprzężenie z bazą danych próbek mogły być w dalszej kolejności wykorzystywane do generowania innych ciekawych rozwiązań. Złożoność reguł komponowania oraz wykorzystanie wielu wymiarów danych wejściowych algorytmu, spowodowało ulokowanie EMI w [104] jako analityczno-transformacyjno-generatywnego systemu algorytmicznego komponowania muzyki o szerokim kontekście muzycznym (rys. 1.23).

Kolejnym rozwiązaniem analizowanym przez Woollera był system Concert zaprezentowany przez Mozera w [72]. Eksperymenty Mozera dotyczyły predykcji wyjściowej frazy nuta po nucie. Głównym założeniem autora było stwierdzenie, że model, bez jakiejkolwiek jawnej reprezentacji formy muzycznej (budowy całego utworu),

może nauczyć się trendów występujących w strukturze, które mogą być następnie wykorzystane do generowania złożonych kompozycji muzycznych. W systemie Concert wykorzystany został łańcuch Markowa trzeciego rzędu, odpowiedzialny za generowanie wyjściowej kompozycji. Początkowo system trenowany był sekwencjami w celu predykcji kolejnych występujących po sobie dźwięków. Proces trenowania systemu wykorzystywał rekursywne sprzężenie zwrotne, w którym pierwszy dźwięk w sekwencji był dany, natomiast kolejne predykcje dźwięków bazowały na analizie wcześniejszych predykcji. Woller ulokował system Concert w [104] jako rozwiązanie podobne we właściwościach algorytmu do systemu EMI, natomiast o węższym kontekście muzycznym (rys. 1.23).

W roku 1999 IBM wypromował system algorytmicznego komponowania muzyki o nazwie Music Sketcher, przedstawiony przez Abramsa i innych w [18] oraz [17]. Music Sketcher był meta-sekwencerem, który generował muzykę przy użyciu małych wcześniej skomponowanych fragmentów danych nazwanych rifami. Użytkownik systemu mógł swobodnie transformować parametry muzyczne (wysokość, długość lub głośność dźwięku) poprzez rysowanie grafów kontrolujących zachowanie się prymitywnych operacji arytmetycznych wykorzystywanych do modyfikacji struktury utworu. Główny proces komponowania muzyki bazował na procedurach transformacyjnych: muzyczne wzorce (rify) mogły być przearanżowane za pomocą dużej liczby funkcji bez naruszania rozmiarów danych wejściowych. Wg Wollera Music Sketcher jest ulokowany na płaszczyźnie modelu jako system transformacyjny o dość wąskim kontekście muzycznym [104] (rys. 1.23).

Microsoft w roku 2001 zaprezentował narzędzie o nazwie DirectMusic Producer (DMP) przeznaczone do komponowania nieliniowej muzyki. Opis tego systemu zaprezentował Buttram w [90] przedstawiając DMP jako system do tworzenia interaktywnej muzyki na potrzeby dynamicznych treści jakimi są gry komputerowe. W systemie DMP podstawową strukturą muzyczną jest "segment", który definiuje pewną część muzyki renderowaną na różne sposoby podczas odtwarzania, w zależności od ustawień użytkownika. Segment jest kontenerem dla "ścieżek", które mogą należeć do różnych zdefiniowanych w systemie klas. Ścieżka może przechowywać liniową sekwencję muzyczną (klasa "Sequence Track") lub zbiór zdefiniowanych przez użytkownika wariacji tworzących wzorzec muzyczny (klasa "Pattern Track"). W DMP muzyczny algorytm komponowania posiada wg Woollera dosyć szeroki kontekst muzyczny, z powodu systemu gramatyk łączącemu na wysokim poziomie abstrakcji dane muzyczne z algorytmem, który je wykorzystuje do przetwarzania. System DMP nie analizuje danych muzycznych, a jedynie dokonuje przejść pomiędzy losowymi schematami akordów oraz pozwala na wybór wariacji wzorca muzycznego. Algorytm muzyczny wykorzystuje rekombinacje wzorców w procesie komponowania muzyki, dlatego też system DMP został przez Woollera sklasyfikowany jako rozwiązanie generatywne z małą funkcjonalnością w zakresie transformacji danych [104] (rys. 1.23).

Kolejnym rozwiązaniem ulokowanym na płaszczyźnie modelu Woollera jest The Continuator autorstwa Pacheta, zaprezentowany w pracy [53] i [52]. System został zaprezentowany jako narzędzie do generowania improwizacji muzycznych, które uczy się modeli Markowa możliwych sekwencji nut na podstawie wbudowanej bazy danych muzyki. Główną cechą tego rozwiązania jest wykorzystanie "funkcji przystosowania" wpływającą na prawdopodobieństwo pojawienia się danego dźwięku w wynikowej improwizacji w czasie rzeczywistym. Pachet zaproponował algorytm, który generuje polifoniczne kompozycje z wykorzystaniem klasteryzacji dźwięków muzycznych

znajdujących się w tym samym tymczasowym regionie utworu. Proces klasteryzacji bazuje na określonych i nieokreślonych regułach przetwarzania. Dla przykładu, określone reguły klasteryzowania definiują stałą metryczną strukturę sekwencji, podczas gdy niejawne reguły wykorzystują dany parametr do określenia przetwarzania sekwencji (np. punkt odcięcia pomiędzy nachodzącymi na siebie dźwiękami i akordami). Wg Woollera The Continuator jest systemem o właściwościach analitycznych, transformacyjnych i generatywnych oraz posiada szeroki zakres kontekstu muzycznego z tytułu zróżnicowanych wejściowych danych muzycznych oraz dużej liczby parametrów sterujących procesem generowania improwizacji [104] (rys. 1.23).

Znaczącym systemem w historii algorytmicznego komponowania muzyki jest GenJam utworzony przez Bilesa i opisany w pracy [13] i [14], chociaż pierwsze prace nad systemem zaczęły się dużo wcześniej [12]. GenJam generuje improwizacje jazzowe oraz dostarcza możliwości tworzenia kompozycji praktycznie w czasie rzeczywistym, na podstawie wykonywanej przez muzyka partii instrumentalnej (np. gry na trąbce). Baza danych systemu zawiera muzyczne struktury i motywy dla całych kompozycji (skomponowanych wg standardów jazzowych). Dane te dostarczają głównemu algorytmowi generującemu muzykę wcześniej zdefiniowaną globalną strukturę utworu, wokół której dokonywane są różne wariacje. Główny mechanizm systemu bazuje na interaktywnym algorytmie genetycznym (IGA - Interactive Genetic Algorithm), w którym lokalna struktura muzyczna jest generowana na podstawie rekombinacji i selekcji, tak, aby pasowała do aktualnie wykonywanego przez muzyka motywu. Wysokości dźwięków muzycznych są mapowane na progresję akordów akompaniujących motywowi oraz skalę muzyczną, w której odgrywany jest aktualnie fragment improwizacji. GenJam może w sposób losowy dokonywać rekombinacji fraz generowanej improwizacji lub wykorzystać specjalną funkcję do określenia najbardziej poprawnego rytmicznie punktu krzyżowania osobników (motywów). Biles wykorzystuje wiele metod heurystycznych w celu poprawienia muzykalności wyjściowej improwizacji, wprowadzając tym samym praktycznie nieograniczone możliwości kombinacji wyjściowych kompozycji. Wooller sklasyfikował system Bilesa jako mocno transformacyjny z elementami analitycznymi (analiza odgrywanego przez człowieka motywu) i generatywnymi o raczej szerokim wykorzystywanym kontekście muzycznym, z uwagi na wykorzystaną bazę danych struktur i motywów muzycznych [104] (rys. 1.23).

W Polsce badania nad możliwością zalgorytmizowania procesu komponowania muzyki należą raczej do niszy naukowej. Autor postanowił jednak przytoczyć przykładowe rozwiązanie powstałe w wyniku prac Zabierowskiego i opisane w [108], a następnie poszerzone o badania zawarte w [101]. W przytoczonych podejściach zastosowane zostały sztuczne sieci neuronowe, które generują progresję akordów z uwzględnieniem zasad harmonii muzycznej. Reguły komponowania zostały zdefiniowane w postaci możliwych następstw funkcji harmonicznych zawartych w triadzie harmonicznej: toniki (T), subdominanty (S) i dominanty (D). Autor tej pracy ulokował badania Zabierowskiego jako rozwiązania o charakterze generatywnym z wąskim kontekstem muzycznym (rys. 1.23).

Rozwiązanie opisane w	Metoda	Bazuje na jawnych regułach	Bazuje na niejawnych regułach
[73]	automaty komórkowe	X	-
[20]	gramatyki	X	X
[12], [13], [14]	algorytm genetyczny	X	-
[82]	algorytm genetyczny	X	-
[84]	algorytm genetyczny	X	-
[58]	algorytm genetyczny	X	-
[24]	algorytm roju cząstek (PSO)	X	-
[60]	algorytm mrówkowy	X	-
[76], [96]	wirtualny model systemu chemicznego	X	-
[108], [101]	sztuczne sieci neuronowe	X	-
[80]	drzewa dezycyjne	-	X
[50], [51]	parsowanie przyrostowe i drzewa PST	-	X
[35]	modele statystyczne	-	X
[46]	ukryte modele Markowa (HMM)	-	X
[56]	łańcuchy Markowa	-	X
[99]	łańcuchy Markowa	-	X
[30], [49]	sztuczne sieci neuronowe	-	X
[38], [37], [39], [40], [41], [34]	sieci ATN i MATN	X	X

Tab. 1.9. Podział systemów algorytmicznego komponowania muzyki ze względu na jawność i niejawność wykorzystanych reguł komponowania. Opracowanie własne.

Systemy algorytmicznego komponowania muzyki w ujęciu reguł komponowania

Systemy algorytmicznego komponowania muzyki można pogrupować, uwzględniając rodzaj reguł komponowania, wykorzystanych w procesie generowania muzyki. Wg [89] systemy te można podzielić na dwie grupy. Pierwszą stanowią rozwiązania z regułami jawnymi, w których kompozytor definiuje z góry zasady komponowania utworu. Drugą grupę stanowią systemy z regułami niejawnymi, w których reguły komponowania pozyskiwane są z przykładów (fragmentów bądź całych utworów muzycznych) za pomocą wybranych metod przetwarzania i pozyskiwania informacji.

W tabeli 1.9 przedstawiono klasyfikację systemów algorytmicznego komponowania muzyki analizowanych przez autora pod względem jawności i niejawności reguł komponowania zastosowanych w algorytmie generującym muzykę.

Systemy komponowania muzyki z regułami jawnymi

Metody komponowania zorientowane na reguły zdefiniowane przez twórcę modelują reguły kompozycji jako funkcję rozkładu podobieństwa występowania danych sekwencji muzycznych opisanych za pomocą wybranych parametrów muzycznych. Najbardziej znanym podejściem jest wykorzystanie ukrytego modelu markowa (z

ang. *Hidden Markov Model*, HMM), w którym przyszłe zdarzenia (np. pojawienie się danej nuty) zależą jedynie od zdarzeń poprzednich. Przykładowe rozwiązanie - system CAMUS - przedstawiono w [85].

W dziedzinie komponowania opartego na zdefiniowanych regułach wyróżnić można również rozwiązania wykorzystujące automaty komórkowe oraz algorytmy genetyczne. Automaty komórkowe zdefiniowane jako macierz komórek, zmieniających się w czasie wg ustalonych reguł i generujących wzorce, stanowią model postaci propagacji wzorców. Komponowanie może być potraktowane jako problem optymalizacyjny, w którym mechanizm tworzenia muzyki stanowi algorytm genetyczny. W uproszczeniu algorytm genetyczny jest procesem, zainicjowanym przez losową populację osobników (np. wzorców muzycznych), która w wyniku reprodukcji i mutacji wybranych osobników, ewoluuje w czasie do kolejnej generacji rozwiązań. Krytycznym elementem tego procesu jest funkcja oceny przystosowania osobników, której zdefiniowanie jest problemem niezwykle złożonym, przez co częściej stosuje się podejście interaktywnej oceny generacji przez człowieka. Przykłady wykorzystania algorytmów genetycznych opisano w [12] i [82].

Ciekawym podejściem jest opisanie reguł komponowania za pomocą metod lingwistycznych oraz kaskadowych sztucznych sieci neuronowych. W takim rozwiązaniu mechanizm generowania kompozycji muzycznej bazuje na informacjach dostarczanych z teorii muzyki - zasad harmonii. Zabierowski i Napieralski opisali ten problem w pracach [107] oraz [101].

Reguły komponowania mogą być modelowane przy użyciu sieci Petriego oraz kodu opisującego generowane struktury muzyczne. Goffredo i Sametti definują w [59] obiekty muzyczne (tzn. pojedyncze dźwięki, fragmenty utworu bądź całe partie instrumentalne) modelowane za pomocą sieci Petriego oraz wyrażeń definiujących strukturę kompozycji przy użyciu specyficznej składni. Wynikiem tych badań jest aplikacja ScoreSynth, syntezująca muzykę w technologii *MIDI*.

Systemy komponowania muzyki z regułami niejawnymi

Metody komponowania algorytmicznego zorientowane na wyodrębnianie reguł muzycznych z przykładowych utworów stanowią podgrupę rozwiązań z dziedziny machine learning. S. Dubnov zaproponował wykorzystanie parsowania przyrostowego (*incremental parsing*) oraz drzewa PST (*prediction suffix tree*) jako mechanizmu wyodrębniania stylu muzycznego z podanych utworów wzorcowych (proces został opisany szerzej w pracach [50] i [51]).

W literaturze komponowanie zostało również przedstawione w postaci drzewa decyzyjnego. Praca [80] Reagana traktuje o zaadaptowaniu klasyfikatora w postaci drzewa decyzyjnego, który poszukiwał pewnych wspólnych właściwości utworów muzycznych z przykładów przedstawionych w postaci nut. Następnie wyszukane właściwości utworu były wykorzystywane do generowania nowej kompozycji.

Ciekawym problemem jest imitowanie stylu muzycznego. Styl muzyczny jest pojęciem złożonym, a jego istota polega na zastosowaniu odpowiednich podobnych układów rytmicznych, melodycznych i harmonicznych w różnych utworach wyrażających ten styl. Conklin w [35] zastosował model statystyczny do pozyskania cech stylu muzycznego z gotowych przykładów w postaci utworów. W pracy zbadano wiele metod pozyskiwania informacji muzycznych - od modelu statystycznego po metody wykorzystujące wyszukiwanie wzorców. Najlepsze rezultaty uzyskano wykorzystując ukryty model markowa (HMM). Kolejnym przykładem wykorzystania

HMM jest system Cybernetic Composer przedstawiony w [46], który generuje muzykę w różnych stylach muzycznych. W większości podejść wykorzystujących modele HMM każdy stan jest na ogół pojedynczą nutą. Dopiero ostatnie prace ukazują modelowanie stanu jako wzorca będącego zbiorem nut i powtarzającego się w utworze w postaci sekwencji. Przykładem wykorzystania HMM w tej postaci jest praca [99].

Innym rozwiązaniem jest wykorzystanie szeroko stosowanych sztucznych sieci neuronowych jako uproszczonego modelu ludzkiego mózgu (przez co ich wykorzystanie w dziedzinie sztucznej twórczości muzycznej wydaje się być intuicyjne), który wyodrębnia wzorce z dostarczanych mu struktur danych. W tej dziedzinie ostatnie badania zostały opisane w pracach [30] i [49].

Wreszcie wspomnieć należy o pracach prof. David Cope'a [38]-[41], którego badania zostały oparte na idei muzycznej gry w kości Mozarta. Rozwiązanie Cope'a polega na poszukiwaniu i wyodrębnianiu prostych sygnatur muzycznych z prac klasyków muzyki (Bach, Mozart) z wykorzystaniem metod dopasowania wzorców. Następnie wzorce zostają zreplikowane oraz zrekombinowane tak aby utworzyć nową melodię.

1.4. Sformułowanie problemu badawczego

Celem pracy jest opracowanie modelu komputerowego systemu komponowania muzyki użytkowej (posiadającej wartość rynkową), w którym do generowania oryginalnych utworów muzycznych wykorzystuje się metody autorskie oparte na metodach sztucznej inteligencji.

Po analizie dotychczasowych rozwiązań w zakresie algorytmicznego komponowania utworów oraz konstruowania maszyn do generowania muzyki sztucznej a także na podstawie badań autorskich algorytmów stawia się hipotezę, że, bazując na metodach sztucznej inteligencji, możliwe jest opracowanie takich algorytmów komponowania utworów muzycznych, których efektem będą utwory atrakcyjne w odbiorze i posiadające strukturę poprawną w aspekcie teorii muzyki. W powyższej hipotezie badawczej zakłada się, że system komponowania bazował będzie na wzorcowych utworach stanowiących muzyczną bazę wiedzy oraz na metodach wyszukiwania i modyfikacji sygnatur muzycznych (komponentów utworu), wykorzystywanych w kompozycji wynikowej. Przez model w ujęciu metodologicznym autor rozumie narzędzie kognitywne umożliwiające badanie i opisywanie rozmaitych aspektów struktury i funkcjonalności systemu komponowania muzyki. Model taki będzie krokiem pośrednim do syntezy metody lub metod generowania muzyki sztucznej prowadzącej w przyszłości do budowania złożonego systemu informatycznego. Przez system ten rozumie z kolei rozbudowaną instalację sprzętowo-programową, której budowa nie jest celem niniejszej pracy. Praca ta może być inspiracją, pierwszym krokiem do budowy i doskonalenia takiego systemu.

W przypuszczeniu, że utwory będą atrakcyjne w odbiorze kryje się wiele subiektywizmu i ta cecha utworów będzie wymagała specyficznego potwierdzenia (generalnie trudno zdefiniować bezwzględną atrakcyjność utworu muzycznego, z pewnością jest to niemożliwe, można jednak posiłkować się miarami pośrednimi – statystycznie mierzonym zainteresowaniem lub rynkowymi np. wartością sprzedaży. Drugi wymieniony istotny postulat – zgodność z kanonami teorii muzyki, jest znacznie prostszy do dowiedzenia i zapewnienia.

Rozdział 2

Koncepcja systemu algorytmicznego komponowania muzyki

W nawiązaniu do przedstawionej analizy różnych podejść z dziedziny algorytmicznego komponowania muzyki oraz postawionego celu pracy zasadne wydaje się wskazanie pewnej luki w prezentowanych rozwiązaniach jaką jest tzw. muzyka tła, która jest jednym z najbardziej interesujących obszarów stosowania muzyki użytkowej. Prezentowany system z założenia funkcjonuje jako narzędzie generujące tzw. muzykę tła (*background music*) z wykorzystaniem mini-modeli zdefiniowanych w dalszej części tego rozdziału. Muzyka tła odnosi się do "*muzyki jakiegokolwiek rodzaju, która jest odtwarzana podczas danej czynności, w taki sposób, aby człowiek aktywnie nie angażował się w słuchanie tej muzyki*" (na podstawie [45]. Ten typ muzyki odgrywa drugorzędną rolę w życiu człowieka, jednak według różnych badań ma wpływ na wiele aspektów naszego zachowania.

Badania prowadzone przez Kämpfe i innych w [65] skupione były na analizie wpływu muzyki na różne zadania wykonywane przez człowieka. Autorzy zasugerowali, że chociaż globalne analizy nie wykazują znaczącego wpływu na efektywność zadań, to szczegółowe badania pozwalają jednak zidentyfikować zarówno pozytywne jak i negatywne oddziaływanie na analizowane czynności. Cockerton i inni w [32] śledzili pozytywny i negatywny wpływ słuchania muzyki podczas badania funkcji poznawczych (rozwiązywania testu kognitywnego). Także Chaudhury i inni w [29] zaprezentowali rolę muzyki w procesie przeprogramowania aktywności mózgu. Autorzy wywnioskowali, że muzyka może wywoływać różne mechanizmy funkcjonowania mózgu m.in. proces nauczania oraz pamiętania.

Przedstawione analizy są jedynie częścią badań prowadzonych na świecie, mających znaczenie nie tylko poznawcze ale również komercyjne. Autor przypuszcza, że system generujący muzykę tła adekwatną do zadanego rodzaju towarzyszącej jej czynności, byłby narzędziem użytecznym rynkowo, pozwalającym na generowanie odpowiedniego gatunku muzyki w zależności od aktualnych potrzeb klienta-słuchacza. Na bazie analizy istniejących rozwiązań opisanych w literaturze oraz niewypełnionej luki badawczej związanej z systemami algorytmicznego komponowania muzyki tła, powstała koncepcja systemu, którego strukturę opisano w dalszej części tego rozdziału.

2.1. Proces komponowania

Przed przystąpieniem do opisywania autorskiej koncepcji systemu algorytmicznego komponowania muzyki, należy w sposób formalny przedstawić szereg zadań składających się na proces komponowania muzyki, ujęty przez Bilesa w [75] na stronach 34-36. Zadania te zostały podzielone na grupy, których struktura została przedstawiona w tab. 2.1. Porządek przedstawionych zadań procesu komponowa-

Generowanie motywów melodii	Sekwencji wysokości dźwięków bez rytmu; Sekwencji rytmicznych bez określenia wysokości dźwięków; Sekwencji dźwięków z informacją o wysokości i rytmie
Rozszerzenie ideii melodycznych	Generowanie wariacji i melodii kontrapunktowych; Łączenie fragmentów melodii w dłuższe elementy
Harmonizacja	Generowanie partii harmonii; Generowanie zmian akordów
Aranżacja	Definiowanie indywidualnych partii dla danego muzyka; Definiowanie instrumentów przypadających na indywidualne partie
Struktura	Wykorzystywanie wysokiego poziomu opisu (top-down); Budowanie lub ewaluacja struktury od podstaw (bottom-up); Podejście hybrydowe w definiowaniu struktury (top-down i bottom-up)

Tab. 2.1. Podstawowe zadania procesu komponowania (na podstawie [75]). Opracowanie własne.

nia został przyjęty arbitralnie przez Bilesa i nie odzwierciedla struktury tworzenia muzyki według modelu kaskadowego.

Ważnym aspektem procesu komponowania w systemach algorytmicznego komponowania muzyki jest generowanie motywu melodii (tab. 2.1). Częstym podejściem jest wykorzystanie generatora melodii, który, bazując na zdefiniowanych bądź wyuczonych regułach generowania, wytwarza krótkie (najczęściej kilku- bądź kilkunastonutowe) fragmenty monofoniczne. Dodatkowo generowanie melodii może zostać rozbite na podzadania dotyczące tworzenia (w sposób niezależny) sekwencji wartości rytmicznych oraz wysokości dźwięków. Niektóre systemy oferują możliwość jednoczesnego generowania sekwencji rytmiczno-częstotliwościowych (np. [12]).

Część systemów algorytmicznego komponowania muzyki generuje jedynie sekwencje rytmiczne, najczęściej o długości jednego taktu, który jest następnie traktowany jako wzorzec umieszczany i powielany w różnych częściach kompozycji. Systemy takie swoją funkcjonalność opierają się na idei tzw. maszyny perkusyjnej (z ang. *drum machine*) - urządzeniu, które przechowuje w pamięci (lub syntezuje w czasie rzeczywistym) próbki dźwięków instrumentów perkusyjnych, wykorzystywanych następnie do tworzenia jedno- bądź wielotaktowych wzorców zapętlonych w celu uzyskania podkładu perkusyjnego. W takich systemach wysokości dźwięku nie są interpretowane jako częstotliwości lecz jako określone identyfikatory instrumentów wykorzystywanych podczas komponowania taktu.[75]

W procesie komponowania kolejnym ważnym zagadnieniem jest rozszerzanie idei melodycznych (tab. 2.1), które ogólnie można zdefiniować jako serie transformacji motywu początkowego, wykorzystanych do wygenerowania motywu docelowego. Zdefiniowane serie transformacji mogą generować wariacje motywu (np. [43], [23], [22]), wzorce rytmiczne ([27]) lub łączyć krótkie motywy w dłuższe linie melodyczne

([23], [22]). Punktem wejściowym w takich systemach jest motyw składający się z kilku lub kilkunastu nut, który jest następnie wykorzystany do tworzenia dłuższej struktury.

Harmonizacja (tab. 2.1) stanowi ważny element procesu komponowania muzyki i w ramach różnych systemów algorytmicznego komponowania polega najczęściej na tworzeniu głosu sopranowego, altowego, tenorowego oraz basowego dla danej melodii. Trudność polega na tym, że wszystkie wymienione głosy muszą być traktowane jako indywidualne melodie, z jednoczesnym zachowaniem harmonii wertykalnej (harmonii pomiędzy melodią oraz pozostałymi głosami). Można więc potraktować zagadnienie harmonizacji melodii jako proces planowania (problem optymalizacyjny) bazujący na zdefiniowanych wcześniej regułach następstwa dźwięków w czasie oraz ich rozmieszczenia w pionie (wybrzmiewania w tej samej chwili, z zachowaniem zasad harmonii). W przypadku gdy wejściem do systemu jest zarówno melodia (do której ma zostać dobrana harmonizacja) oraz progresja akordów, problem jest relatywnie prosty ([62]). Sytuacja komplikuje się, gdy wejściem systemu komponowania jest jedynie melodia. W takim przypadku stosuje się zazwyczaj metody bazujące na systemie reguł lub sieciach neuronowych, jak np. [83]. [75]

Aranżacja (tab. 2.1) stanowi problem doboru fragmentów utworu do określonych partii instrumentów muzycznych. Częściowo problem ten jest rozwiązywany w systemach rozbudowujących (rozszerzających) melodię oraz harmonizujących wejściowy materiał muzyczny. Istnieją jednak systemy ([23]), w których wyodrębniony został osobny moduł funkcjonalny odpowiedzialny za aranżację (tzw. aranżer).

Ostatnim i prawdopodobnie najtrudniejszym zagadnieniem procesu komponowania muzyki jest tworzenie logicznej struktury utworu (tab. 2.1). Niektóre systemy algorytmicznego komponowania muzyki poruszają się w płaszczyźnie niskiego poziomu opisu kompozycji (na ogół na poziomie opisu pojedynczej frazy utworu). Istnieją również systemy, które są w stanie generować w pełni rozwinięte kompozycje, poruszając się po różnych płaszczyznach opisu struktury utworu (np. od frazy po formę) [22]. Systemy takie mają najczęściej charakter interaktywny i pozwalają (lub często wymagają) zdefiniowania przez użytkownika struktury na najwyższym poziomie opisu kompozycji w sposób manualny ([92]) bądź częściowo zautomatyzowany ([23], [22]). Kształtowanie struktury w systemach komponowania jest realizowane również z wykorzystaniem różnych technik i metod obliczeń ewolucyjnych takich jak agenty, automaty komórkowe lub PSO (optymalizacja rojem cząsteczek) ([15], [57]).

Przedstawiona analiza procesu komponowania ujęta przez Bilesa w [75] pozwala autorowi dokonać próby ulokowania swojej koncepcji systemu algorytmicznego komponowania muzyki na tle innych rozwiązań jako systemu, w którym:

— sekwencja melodii generowana jest z uwzględnieniem informacji na temat wysokości i długości dźwięków ją tworzących;
— rozszerzenie melodii polega na łączeniu jej fragmentów w dłuższe elementy, z zastosowaniem zdefiniowanych reguł transformacji;
— generowanie harmonii bazuje na regułach zmian akordów związanych z melodią wiodącą;
— etap aranżacji ujęty jako zdefiniowanie instrumentów do określonych partii utworu pozostaje pod kontrolą użytkownika;

— struktura utworu definiowana jest z poziomu wysokiej abstrakcji opisu formy kompozycji (top-down), a możliwe połączenia fragmentów utworu w jedną kompozycję wynikają z analizy właściwości tych fragmentów (bottom-up).

W dalszej części pracy autor zaprezentował koncepcję systemu algorytmicznego komponowania muzyki, w której proces komponowania został opisany i przedstawiony w dwóch wariantach: komponowania bez definiowania struktury generowanego utworu oraz komponowania uwzględniającego zdefiniowaną formę generowanego utworu. Na rys. 2.1 przedstawiona została ogólna koncepcja autorskiego systemu algorytmicznego komponowania muzyki na podstawie wzorców. W ramach modelu rozróżnione zostały dwa etapy powstawania utworu muzycznego: generowanie wzorca oraz generowanie kompozycji.

Pierwszy wariant komponowania zakłada generowanie kompozycji muzycznej, bazujące na generowaniu wzorców muzycznych i łączeniu ich w jedną kompozycję. Wzorce muzyczne są generowane za pomocą parametrów wejściowych, takich jak długość wzorca, wektor opisujący sekwencję wysokości dźwięków motywu muzycznego, wektor opisujący możliwe rozmieszczenie dźwięków w ramach wzorca, itp. Kompozycja muzyczna tworzona jest natomiast z wygenerowanych wcześniej wzorców, których możliwość połączeń definiuje macierz, utworzona na podstawie analizy właściwości poszczególnych wzorców (ich podobieństwa). Podejście to zostało opisane w rozdziale 2.2.

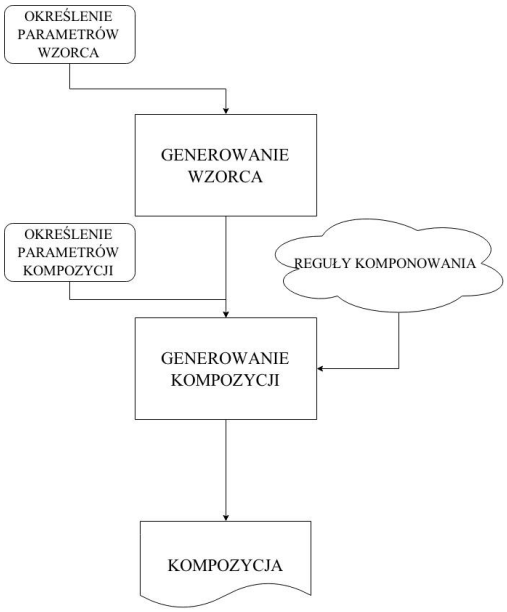
Drugi wariant komponowania bazuje na opisie formy utworu końcowego (liczbie i kolejności łączenia głównych fragmentów utworu) oraz procesie generowania kompozycji, wykorzystującym wygenerowane wcześniej wzorce muzyczne, których łączenie uwzględnia zdefiniowaną formę kompozycji. Proces generowania wzorców ma charakter odmienny, ponieważ, w odróżnieniu od pierwszego wariantu, głównym elementem nie jest motyw muzyczny, a wygenerowana (na podstawie reguł) progresja akordów, do której generowana jest melodia i dalsze partie instrumentalne. Podejście to zostało opisane w rozdziale 2.3.

2.2. Komponowanie bez definiowania formy generowanego utworu

Generowanie muzyki nieposiadającej skrystalizowanej formy (tzn. początku, rozwinięcia i zakończenia) jest zagadnieniem dość typowym dla muzyki tła, której cechą charakterystyczną są powtarzające się fragmenty muzyczne i ich wariacje. Taki sposób generowania muzyki może być porównany do generowania dużej tekstury obrazu z relatywnie małych fragmentów. [74]

Autor zaproponował w tym rozdziale koncepcję algorytmicznego komponowania muzyki, której forma nie jest ściśle skrystalizowana, co umożliwia tworzenie kompozycji zapętlonej w czasie. Na rys. 2.2 została przedstawiona w postaci graficznej struktura proponowanej koncepcji.

Aby zrozumieć zaproponowaną koncepcję systemu należy w pierwszej kolejności zdefiniować podstawowy element muzyczny, z jakiego tworzona jest kompozycja muzyczna.



Rys. 2.1. Ogólna architektura generatywnego systemu algorytmicznego komponowania muzyki na podstawie wzorców. Opracowanie własne.

Definicja wzorca muzycznego

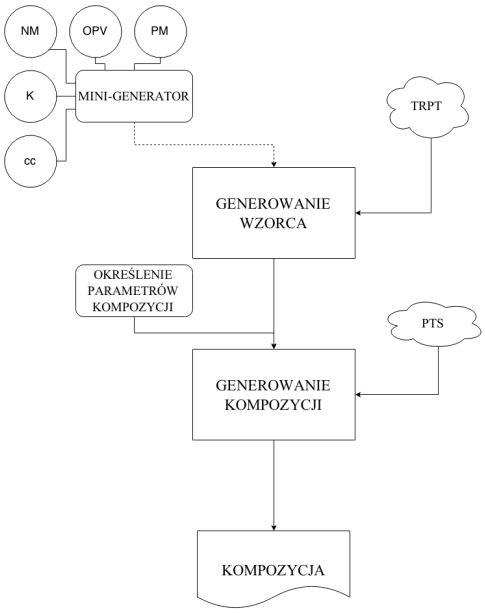
W przyjętej koncepcji systemu wyjściowa kompozycja muzyczna tworzona jest z mniejszych fragmentów muzycznych, zwanych wzorcami muzycznymi. Wzorzec jest w tym podejściu zdefiniowany jako macierz nut zaproponowana w [48] (przykład: tab. 1.4 na stronie 13), przy czym dodatkowo macierz ta posiada stałą i jednakową dla wszystkich wierszy wartość długości dur_W nuty (za równo w jednostkach bpm, jak i sekundach). Uproszczenie to jest podyktowane sposobem generowania wzorca, który został opisany w kolejnej części tego rozdziału oraz koniecznością definiowania parametru określającego długość wzorca.

Długość l_W wzorca wyrażona jest w jednostkach bpm i wpływa bezpośrednio na liczbę wierszy macierzy. Zależność ta wyrażona jest wzorem:

$$m_{MN} = \begin{cases} \frac{l_W}{dur_W} & \text{jeżeli } \frac{l_W}{dur_W} \in N \\ \left\lfloor \frac{l_W}{dur_W} \right\rfloor & \text{jeżeli } \frac{l_W}{dur_W} \notin N \end{cases}, \tag{2.1}$$

gdzie:
— m_{MN} oznacza liczbę wierszy macierzy nut (MN),
— l_W oznacza zadeklarowaną długość wzorca w jednostkach bpm,
— dur_W jest wartością długości nut zawartych w macierzy MN.
Na rys. 2.3 przedstawione zostały przykłady wzorca o długości 4 bpm z wartością długości $dur_W = 1$ bpm (cała nuta, 2.3A), $dur_W = 0.5$ bpm (pół nuta, 2.3B), $dur_W = 0.25$ bpm (ćwierć nuta, 2.3C) oraz $dur_W = 0.125$ bpm (ósemka, 2.3D).

Dla czytelności w przykładach z rys. 2.3 nuty zostały ulokowane w co drugiej pozycji w macierzy MN. W podanym przykładzie kolejne wzorce muzyczne posiadają następujące parametry:
— $l_W = 4$, $dur_W = 1$, $m_{MN} = 4$ (2.3A),
— $l_W = 4$, $dur_W = 0.5$, $m_{MN} = 8$ (2.3B),
— $l_W = 4$, $dur_W = 0.25$, $m_{MN} = 8$ (2.3C),



Rys. 2.2. Ogólna architektura generatywnego systemu algorytmicznego komponowania muzyki tła - podejście bez definiowania formy kompozycji. Opracowanie własne.

— $l_W = 4$, $dur_W = 0.125$, $m_{MN} = 16$ (2.3D).
Ostatecznie wzorzec muzyczny W jest zdefiniowany jako macierz nut MN, z uwzględnieniem założeń z (2.1):

$$W = MN_{m\times 7}, m = \begin{cases} \frac{l_W}{dur_W} & \text{jeżeli } \frac{l_W}{dur_W} \in N \\ \left\lfloor \frac{l_W}{dur_W} \right\rfloor & \text{jeżeli } \frac{l_W}{dur_W} \notin N \end{cases}. \tag{2.2}$$

Tak zdefiniowana budowa wzorca narzuca ograniczenie odnoszące się do danych muzycznych zawartych w macierzy MN - dane zapisane w macierzy mają postać skwantyzowaną. W cyfrowym przetwarzaniu muzyki w formacie *MIDI* kwantyzacja jest procesem wyrównującym rozmieszczenie dźwięków muzycznych wzdłuż linii czasu kompozycji do precyzyjnych lokalizacji zdefiniowanych przez siatkę kwantyzacji. Poszczególne nuty są przesuwane do najbliższych absolutnych wartości linii siatki kwantyzacji ulokowanej w funkcji czasu. W przyjętej definicji wzorca muzycznego każdy element macierzy MN jest związany z jedną linią siatki, przez co wejściowy materiał muzyczny jest generowany od początku w postaci skwantyzowanej.

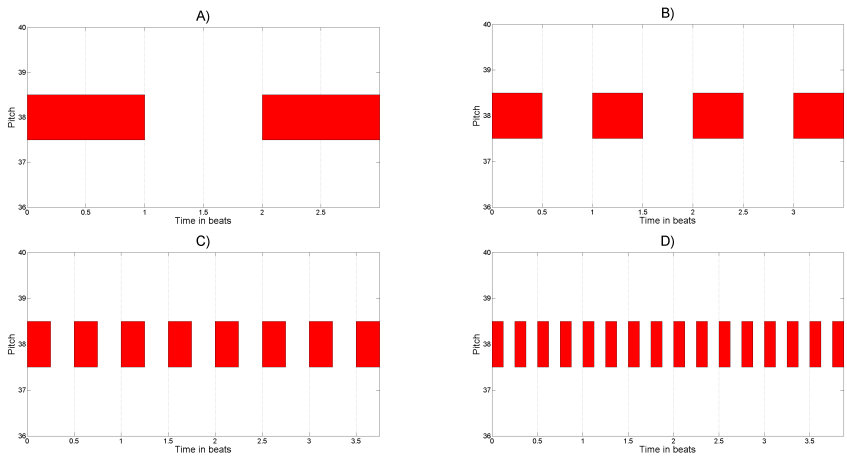
Istotnym elementem w prezentowanej koncepcji systemu jest możliwość konkatenacji wzorców w bardziej złożone wzorce muzyczne. Konkatenacja i-tego wzorca W z j-tym wzorcem W ma charakter wertykalny i została zdefiniowana jako:

$$\tilde{W} = W_i + W_j = \begin{bmatrix} W_i \\ W_j \end{bmatrix}, \tag{2.3}$$

gdzie $\tilde{W}$ oznacza nowy wzorzec po konkatenacji wzorca W_i ze wzorcem W_j, opisanych za pomocą macierzy nut odpowiednio MN_i i MN_j.

Konkatenacja wertykalna wzorców W jest możliwa tylko wtedy, gdy spełniony jest warunek:

$$W_i + W_j = \tilde{W} \iff m_i = m_j, \tag{2.4}$$



Rys. 2.3. Wykresy pianolowe przykładowego wzorca o długości 4 bpm i wartości długości nuty: 1 bpm (A), 0.5 bpm (B), 0.25 bpm (C) oraz 0.125 bpm (D). Opracowanie własne.

w którym m_i, m_j oznaczają odpowiednio liczbę kolumn macierzy nut MN_i i MN_j opisujących wzorce W_i oraz W_j w (2.3). Warunek konkatenacji zapewnia definicja wzorca W przedstawiona w (2.2), ponieważ $m_i = m_j = 7$.

Dla wzorca $\tilde{W}$ można określić najmniejszą możliwą wartość długości $dur_{\tilde{W}}$ nuty, która może być w nim przechowana. Wartość ta jest określana jako

$$dur_{\tilde{W}} = \min\{dur_{W_i}, dur_{W_j}\}, \tag{2.5}$$

gdzie dur_{W_i} oraz dur_{W_j} oznaczają najmniejszą wartość długości nuty odpowiednio we wzorcu W_i oraz W_j.

Ostatecznie (2.3) oraz (2.5) można uogólnić i sprowadzić do postaci:

$$\tilde{W} = W_0 + \ldots + W_k = \begin{bmatrix} W_0 \\ \vdots \\ W_k \end{bmatrix}, k \geq 1, \tag{2.6}$$

oraz

$$dur_{\tilde{W}} = \min\{dur_{W_0}, \ldots, dur_{W_k}\}, k \geq 1. \tag{2.7}$$

Nowy wzorzec $\tilde{W}$ jest macierzą nut $\tilde{MN}$ o rozmiarach $m_{\tilde{MN}} \times 7$, przy czym liczbę wierszy $m_{\tilde{MN}}$ określa warunek

$$m_{\tilde{MN}} = m_{MN_0} + \ldots + m_{MN_k}, k \geq 1. \tag{2.8}$$

Dla zdefiniowanego w (2.2) wzorca oraz operacji konkatenacji (2.6) określmy następnie parametry, które posłużą do jego wypełnienia, adekwatnie do potrzeb kompozycji muzycznej. W tym celu wprowadzono pojęcia mini-generatora, który na podstawie przyjętych założeń generuje wzorzec muzyczny, wykorzystywany w dalszej kolejności w procesie algorytmicznego komponowania muzyki. Zdefiniowana w (2.6) operacja konkatenacji jest wykorzystywana do łączenia wzorców generowanych przez różne mini-generatory w jeden wzorzec, będący docelowym fragmentem komponowanego utworu.

Definicja mini-generatora

Mini-generator (oznaczony jako *Mg*) jest podstawowym komponentem w opisywanej koncepcji autorskiego systemu algorytmicznego komponowania muzyki na podstawie wzorców muzycznych. Definicja mini-generatora *Mg* została przedstawiona poniżej jako:

$$Mg = \{K, MM, WPN, dur_W, l_W, t\}, \tag{2.9}$$

gdzie:
— K oznacza klasę mini-generatora,
— *MM* - motyw muzyczny na bazie którego generowany jest wzorzec muzyczny *W*,
— *WPN* - wektor pozycji (wierszy macierzy *MN*) nut, w których mają być rozmieszczone dźwięki z motywu *MM*,
— dur_W - minimalna długość nuty generowanego wzorca W określona w jednostkach bpm,
— l_W- długość generowanego wzorca W określona w jednostkach bpm,
— t - tempo generowanego wzorca W określone w jednostkach bpm.

Występujące w wyrażeniu (2.9) oznaczenie K definiuje klasę do jakiej należy mini-generator *Mg*. Przynależność do danej klasy K determinuje sposób wykorzystania mini-generatora *Mg* w procesie generowania wzorca muzycznego. W prezentowanym systemie zdefiniowane zostały następujące klasy mini-generatorów:
— generator monofoniczny,
— generator polifoniczny,
— generator perkusyjny.

Generator monofoniczny wykorzystywany jest do generowania monofonicznych fragmentów muzycznych takich jak np. melodia główna czy monofoniczna linia basowa. Generator polifoniczny generuje polifoniczne fragmenty muzyczne takie jak akord muzyczny. Generator perkusyjny to szczególna klasa mini-generatora, którego generowane sekwencje muzyczne (za równo monofoniczne jak i polifoniczne) są interpretowane jako sekwencje perkusyjne.

Kolejnym parametrem opisującym działanie mini-generatora *Mg* jest macierz nut *NM* (*Note Matrix*). Strukturę tą zaprezentowali Toiviainen i Eerola, opisując framework *MIDI* Toolbox dla środowiska Matlab w [48]. Szczegółową konstrukcję tej macierzy przedstawiono w rozdziale "Wprowadzenie", a jej przykład w tab. 1.4. Należy jednak zaznaczyć, że w przypadku macierzy nut *NM* wykorzystywanej przez mini-generator w podstawowej fazie eksperymentów wykorzystywana była stała zdefiniowana wartość długości dźwięków muzycznych.

Za rozmieszczenie dźwięków muzycznych w odpowiednich miejscach macierzy nut *NM* odpowiada wektor pozycji dźwięków *WPN*. Struktura wektora *WPN* jest generowana na podstawie wyrażenia:

$$WPN = \{n : rs : l\}, n, rs < l. \tag{2.10}$$

Parametr n jest dowolną liczbą naturalną mniejszą od zdefiniowanej liczby wierszy l macierzy nut *NM*. Stała *rs* jest liczbą naturalną określającą krok wypełnienia wektora *WPN*. Dla przykładu: mając zdefiniowaną liczbę wierszy l (dźwięków) w macierzy *NM* jako 32, parametr n ustawiony na wartość 1 oraz stałą *rs* ustawioną na 4, wektor *WPN* będzie miał postać $WPN = \{1, 5, 9, 13, 17, 21, 25, 29\}$, co oznacza, że tylko dla zdefiniowanych pozycjach *WPN* macierzy nut *NM* mini-generatora *Mg*

może osadzać dźwięki muzyczne podczas tworzenia wzorca muzycznego. Rozmieszczenie dźwięków w macierzy *NM* wpływa na rytmikę wzorca muzycznego generowanego przez mini-generator i wykorzystanego następnie do generowania wyjściowej kompozycji.

Mini-generator *Mg* do generowania muzyki wykorzystuje strukturę, która stanowi kombinację motywu muzycznego wcześniej zdefiniowanego przez użytkownika. W podstawowej wersji prezentowanego systemu motyw stanowi wektor liczb naturalnych zawartych w przedziale od 0 do 127, traktowanych jako wysokość dźwięku w standardzie *MIDI*. Zbiór możliwych kombinacji zdefiniowanego motywu muzycznego stanowi macierz *KMM*. Macierz *KMM* składa się z k-elementowych kombinacji C n-elementowego motywu muzycznego *MM*, rozłożonych w macierzy o wymiarach m wierszy (liczba kombinacji) na k kolumn (liczba elementów motywu muzycznego)

$$KMM = \begin{bmatrix} {}_1C_k^n \\ {}_2C_k^n \\ \vdots \\ {}_mC_k^n \end{bmatrix}, k <= n. \tag{2.11}$$

Z macierzy *KMM* losowana jest dla mini-generatora *Mg* kombinacja motywu muzycznego zdefiniowana jako *MM*. Mini-generator *Mg* może również przyjmować *MM* w postaci stałej zdefiniowanej liczby naturalnej. Tak sparametryzowany *Mg* generuje fragmentu muzyczne, w których wszystkie dźwięki posiadają tą samą wysokość równą stałej *MM*. W zależności od klasy K oraz rodzaju *MM* (wektor lub skalar) zdefiniowane zostały różne reguły generowania wzorców, przedstawione w rozdziale "Generowanie wzorców".

Ostatnim parametrem definiującym mini-generator *Mg* jest współczynnik *ms* definiujący liczbę dźwięków zawartych w *MM*, które mają być wykorzystane do produkcji wzorca muzycznego. Im większy jest współczynnik *ms*, tym więcej dźwięków zawartych w *MM* będzie wykorzystanych w wygenerowanym wzorcu. Relacja ta została zdefiniowana w następnej części tego rozdziału.

Należy zaznaczyć, że dowolna liczb mini-generatorów może wykorzystywać te same struktury *MM*, *WPN* oraz *ms*. Sytuacja ta została przedstawiona w części opisującej wyniki eksperymentów. Im większa liczba mini-generatorów zdefiniowanych w systemie, tym bardziej złożone będą wyjściowe kompozycje muzyczne budowane z wykorzystaniem wzorców generowanych przez mini-generatory. W dalszej kolejności opisany został proces generowania wzorców muzycznych, w którym przedstawiono reguły generowania, uzależnione od klasy K oraz zdefiniowanego dla *Mg* rodzaju *MM*.

2.2.1. Generowanie wzorców

Liczba mini-generatorów zdefiniowanych przez użytkownika w przedstawianym systemie ma wpływ na proces generowania wzorców, a dokładniej - na ich złożoność rytmiczno - harmoniczną. *Mg* może generować prostą sekwencję perkusyjną przy użyciu jednego instrumentu bądź grupy instrumentów, może także generować monofoniczną sekwencję interpretowaną jako melodia bądź sekwencję polifoniczną wykorzystywaną jako progresja akordów. Sposób generowania wzorca muzycznego jest związany z wykorzystaną przez mini-generator regułą komponowania.

Klasa mini-generatora K	MM	Reguła generowania wzorca
Generator monofoniczny	wektor	gr1
Generator monofoniczny	skalar	gr2
Generator polifoniczny	wektor	gr3
Generator polifoniczny	skalar	gr4
Generator perkusyjny	wektor	gr5
Generator perkusyjny	skalar	gr6

Tab. 2.2. Związek pomiędzy klasą K mini-generatora, rodzajem MM oraz zdefiniowaną regułą generowania wzorca. Opracowanie własne.

Reguły komponowania

W tab. 2.2 zaprezentowana została zależność pomiędzy klasą mini-generatora (K), rodzajem *MM* oraz regułą *gr* generowania wzorca wykorzystywaną przez mini-generator.

Podczas generowania wzorca przez dany mini-generator *Mg* obliczany jest współczynnik pokrycia *cc* według następującej reguły

$$cc = \frac{l}{ms}, ms <= l. \tag{2.12}$$

Współczynnik *cc* definiuje zbiór wybranych pozycji w macierzy *NM*, które mają zostać wypełnione zgodnie z wektorem *WPN*, wartościami z *MM*. W dalszej części przedstawiono zdefiniowane przez autora reguły generowania wzorców muzycznych.

Reguła *gr1* przedstawiona w tab. 2.2 została zdefiniowana w następujący sposób:

1. *Na początku wypełnij wybrane pozycje w* WPN *wartościami z wektora* MM*, wybierając pozycje* WPN *w zależności od parametru* cc.
2. *Jeżeli w wektorze* WPN *jest więcej pozycji niż elementów wektora* MM*, kontynuuj wypełnianie* WPN *zaczynając od początku wektora* MM.
3. *Po wypełnieniu przypisz wygenerowaną sekwencję do danego kanału MIDI.*

Zastosowanie reguły *gr1* powoduje wygenerowanie przez mini-generator *Mg* monofonicznej sekwencji zawierającej wybrane elementy wektora *MM*. Należy przy tym zaznaczyć, że wartość parametru *cc* równa 1, powoduje wybór wszystkich dostępnych elementów wektora *MM*, adekwatnie do liczby elementów wektora *WPN*.

Kolejna reguła *gr2* zdefiniowana została przez autora w następujący sposób:

1. *Wypełnij wybrane pozycje w* WPN *wartością* MM*, wybierając pozycje* WPN *w zależności od parametru* cc.
2. *Następnie przypisz wygenerowaną sekwencję do danego kanału MIDI.*

Zastosowanie reguły *gr2* powoduje generowanie przez mini-generator *Mg* monofonicznej sekwencji zawierającej jedną wartość wysokości dźwięku na wszystkich wybranych pozycjach macierzy *NM*.

Reguła *gr3* zdefiniowana przez autora na potrzeby badań służy do generowania sekwencji polifonicznych. Jej brzmienie zdefiniowano w następujący sposób:

1. *Na początku, utwórz dwie kopie danego wektora* MM.

2. *Następnie dokonaj transpozycji pierwszej kopii wektora* MM *o 4 lub 5 (w zależności od tonacji) półtonów w górę oraz transpozycji drugiej kopii wektora* MM *o 7 półtonów w górę.*
3. *Dalej, wypełnij wybrane pozycje w* WPN *wartościami z danego wektora* MM *oraz jego dwóch kopii, wybierając pozycje* WPN *w zależności od parametru* cc.
4. *Jeżeli w wektorze* WPN *jest więcej pozycji niż elementów wektora* MM *i jego kopii, kontynuuj wypełnianie* WPN *zaczynając od początku wektora* MM *oraz jego kopii.*
5. *Następnie przypisz wygenerowaną sekwencję do danego kanału MIDI.*

W podstawowych badaniach prowadzonych nad koncepcją systemu zdefiniowana reguła *gr3* generuje jedynie akordy molowe i durowe w ich wersjach podstawowych.

Kolejną regułą zdefiniowaną przez autora jest reguła *gr4* o następującym brzmieniu:

1. *Na początku, utwórz dwie kopie* MM.
2. *Następnie dokonaj transpozycji pierwszej kopii* MM *o 4 lub 5 (w zależności od tonacji) półtonów w górę oraz transpozycji drugiej kopii* MM *o 7 półtonów w górę.*
3. *Dalej, wypełnij wybrane pozycje w* WPN *otrzymanymi wartościami z* MM *oraz jego dwóch kopii, wybierając pozycje* WPN *w zależności od parametru* cc.
4. *Następnie przypisz wygenerowaną sekwencję do danego kanału MIDI.*

Zastosowanie reguły *gr4* powoduje generowanie przez mini-generator *Mg* sekwencji polifonicznej z jednym utworzonym akordem, naniesionym na wybrane pozycje *WPN* i macierzy *NM*.

Reguły *gr5* i *gr6* służą do generowania sekwencji perkusyjnych. Autor zdefiniował regułę *gr5* w następujący sposób:

1. *Na początku wypełnij wybrane pozycje w* WPN *wartościami z wektora* MM, *wybierając pozycje* WPN *w zależności od parametru* cc.
2. *Jeżeli w wektorze* WPN *jest więcej pozycji niż elementów wektora* MM, *kontynuuj wypełnianie* WPN *zaczynając od początku wektora* MM.
3. *Po wypełnieniu przypisz wygenerowaną sekwencję do 10-ego kanału MIDI.*

Natomiast reguła *gr6* otrzymuje brzmienie:

1. *Wypełnij wybrane pozycje w* WPN *wartością* MM, *wybierając pozycje* WPN *w zależności od parametru* cc.
2. *Następnie przypisz wygenerowaną sekwencję do 10-ego kanału MIDI.*

W regułach *gr3* i *gr4* pojawia się problem dotyczący szacowania tonacji generowanego fragmentu muzycznego. Na potrzeby badań autora wykorzystany został algorytm Krumhansl-Schmucklera znajdowania tonacji z wykorzystaniem tzw. profili durowych i molowych zaproponowanych przez Krumhansl i Kesslera w [31]. Reguły *gr3* i *gr4* generują sekwencję akordów dopasowanych pod względem tonacji do melodii głównej.

Metoda Krumhansl-Schmucklera bazuje na współczynniku korelacji R Pearsona zdefiniowanym jako

$$R = \frac{\sum_{i=1}^{n} (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n} (x_i - \bar{x})^2 \sum_{i=1}^{n} (y_i - \bar{y})^2}} \tag{2.13}$$

gdzie x jest histogramem klas wysokości dźwięku, powstałym w wyniku analizy fragmentu muzycznego, natomiast y stanowi listę ważonych wartości w dwunastostopniowej skali w danej tonacji. Wyrażenia $\bar{x}$ i $\bar{y}$ są średnimi wartości odpowiednio histogramu klas wysokości dźwięku x i listy y ważonych wartości dla danej tonacji. Wartość współczynnika R zawiera się w przedziale od -1 do +1, przy czym:

— 0 - oznacza brak liniowej zależności pomiędzy zmiennymi losowymi,
— +1 - oznacza dokładną zależność liniową pomiędzy zmiennymi, w której gdy jedna zmienna losowa rośnie, druga także rośnie,
— -1 - oznacza dokładną zależność liniową pomiędzy zmiennymi, w której gdy jedna zmienna losowa rośnie, druga maleje.

Zmienna x współczynnika korelacji R reprezentuje znormalizowany histogram dźwięków występujących w analizowanym fragmencie względem tzw. klas wysokości dźwięku (*pitch-class*) z uwzględnieniem długości nuty (tzw. modelu Parncutt'a opisanego w [86]). Histogram tworzy 12-sto elementowy wektor *pcd* (*pitch-class distribution*) zdefiniowany jako

$$pcd = \{p_1, p_2, ..., p_d\}, d \in \{C, C\#, D, D\#, E, F, F\#, G, G\#, A, A\#, H\}. \quad (2.14)$$

Prawdopodobieństwo p_d przynależności dźwięku muzycznego do danej klasy wysokości dźwięku wyliczane jest na podstawie wyrażenia

$$pcd(pc_k) = pcd(pc_k) + du(k), k \in \{1, 2, \ldots, l_{pc}\} \quad (2.15)$$

w którym przynależność do k-tej klasy wysokości dźwięku pc analizowanej k-tej wysokości dźwięku muzycznego w dana jest jako

$$pc_k = mod(w_k, 12) + 1 \quad (2.16)$$

natomiast du oznacza model Parncutt'a ważonego wpływu długości nuty na jej przynależności do k-tej klasy wysokości pc, co wyrażono jako

$$du = (1 - e^{\frac{-dur}{\tau}})^{ai}. \quad (2.17)$$

W modelu Parncutt'a dur oznacza długość dźwięku wyrażoną w milisekundach, parametr τ oznacza wartość saturacji długości nuty natomiast parametr ai wyznacza minimalną wartość dyskryminowaną długości nuty. Badania realizowane w [86] zdefiniowały optymalne wartości parametrów τ i ai odpowiednio na 0.5 oraz 2.0.

Postać znormalizowana histogramu dystrybucji klas wysokości dźwięku w analizowanym fragmencie zdefiniowana jest jako

$$pcd = \frac{pcd}{\sum(pcd + \epsilon)} \quad (2.18)$$

w którym ϵ oznacza bardzo małą liczbę (np. 10^{-12}), która wyklucza błąd dzielenia przez zero podczas normalizacji.

Zmienna y współczynnika korelacji R reprezentuje i-ty wiersz macierzy profili tonalnych Krumhansl-Kesslera przedstawionych w tab. 2.3. Macierz ta została podzielona na część górną, w której znajdują się wagi dla hierarchii tonacji durowych, natomiast dolna część zawiera wagi tonacji molowych. Każdy i-ty wiersz macierzy w części górnej i dolnej jest przekształceniem poprzedniego wiersza, będącym przesunięciem wartości w prawo o jedną kolumnę. Górna część macierzy w kolumnach

0.39	0.14	0.21	0.14	0.27	0.25	0.15	0.32	0.15	0.22	0.14	0.18
0.18	0.39	0.14	0.21	0.14	0.27	0.25	0.15	0.32	0.15	0.22	0.14
0.14	0.18	0.39	0.14	0.21	0.14	0.27	0.25	0.15	0.32	0.15	0.22
0.22	0.14	0.18	0.39	0.14	0.21	0.14	0.27	0.25	0.15	0.32	0.15
0.15	0.22	0.14	0.18	0.39	0.14	0.21	0.14	0.27	0.25	0.15	0.32
0.32	0.15	0.22	0.14	0.18	0.39	0.14	0.21	0.14	0.27	0.25	0.15
0.15	0.32	0.15	0.22	0.14	0.18	0.39	0.14	0.21	0.14	0.27	0.25
0.25	0.15	0.32	0.15	0.22	0.14	0.18	0.39	0.14	0.21	0.14	0.27
0.27	0.25	0.15	0.32	0.15	0.22	0.14	0.18	0.39	0.14	0.21	0.14
0.14	0.27	0.25	0.15	0.32	0.15	0.22	0.14	0.18	0.39	0.14	0.21
0.21	0.14	0.27	0.25	0.15	0.32	0.15	0.22	0.14	0.18	0.39	0.14
0.14	0.21	0.14	0.27	0.25	0.15	0.32	0.15	0.22	0.14	0.18	0.39
0.38	0.16	0.21	0.32	0.15	0.21	0.15	0.28	0.24	0.16	0.20	0.19
0.19	0.38	0.16	0.21	0.32	0.15	0.21	0.15	0.28	0.24	0.16	0.20
0.20	0.19	0.38	0.16	0.21	0.32	0.15	0.21	0.15	0.28	0.24	0.16
0.16	0.20	0.19	0.38	0.16	0.21	0.32	0.15	0.21	0.15	0.28	0.24
0.24	0.16	0.20	0.19	0.38	0.16	0.21	0.32	0.15	0.21	0.15	0.28
0.28	0.24	0.16	0.20	0.19	0.38	0.16	0.21	0.32	0.15	0.21	0.15
0.15	0.28	0.24	0.16	0.20	0.19	0.38	0.16	0.21	0.32	0.15	0.21
0.21	0.15	0.28	0.24	0.16	0.20	0.19	0.38	0.16	0.21	0.32	0.15
0.15	0.21	0.15	0.28	0.24	0.16	0.20	0.19	0.38	0.16	0.21	0.32
0.32	0.15	0.21	0.15	0.28	0.24	0.16	0.20	0.19	0.38	0.16	0.21
0.21	0.32	0.15	0.21	0.15	0.28	0.24	0.16	0.20	0.19	0.38	0.16
0.16	0.21	0.32	0.15	0.21	0.15	0.28	0.24	0.16	0.20	0.19	0.38

Tab. 2.3. Macierz hierarchii tonalnych Krumhansl-Kesslera. Opracowanie własne.

zawiera wagi dla tonacji durowych w danej (od prawej strony) kolejności: C, C#, D, D#, E, F, F#, G, G#, A, A#, H. Dolna część macierzy zawiera wagi dla tonacji molowych w kolejności: c, c#, d, d#, e, f, f#, g, g#, a, a#, h.

Ostatecznie tonacja t analizowanego fragmentu muzycznego spełnia warunek

$$t_k = \arg_k \max R(x, y_k), \tag{2.19}$$

gdzie:

— x oznacza histogram klas wysokości dźwięku,

— y_k jest określonym wierszem wag macierzy hierarchii tonalnych Krumhansl-Kesslera,

— k jest indeksem zmieniającym się po 24-ro wierszowej macierzy hierarchii tonalnych Krumhansl-Kesslera, przedstawionej w tab. 2.3.

Maksymalna wartość w 24-ro elementowym wektorze współczynników R korelacji pomiędzy histogramem x a profilem hierarchii tonalnych Krumhansl-Kesslera, wyznacza potencjalną tonację analizowanego fragmentu muzycznego, przy czym kolejność współczynników R interpretowana jest w kontekście tonacji w sposób przedstawiony w tab. 2.4.

C	C#	D	...	c	c#	...	h
1	2	3	...	13	14	...	24

Tab. 2.4. Pozycja współczynnika korelacji R w kontekście tonacji muzycznej. Opracowanie własne.

Nazwa reguły	Opis	Prawdopodobieństwo wybrania	Klasa mini-generatora
tr1	Transpozycja o 4 półtony w górę	P1	Monofoniczny, polifoniczny, perkusyjny
tr2	Transpozycja o 7 półtonów w górę	P2	Monofoniczny, polifoniczny, perkusyjny
tr3	Transpozycja o oktawę w górę	P3	Monofoniczny, polifoniczny, perkusyjny
tr4	Transpozycja o oktawę w dół	P4	Monofoniczny, polifoniczny, perkusyjny
tr5	Transpozycja o oktawę w górę, a następnie 4 półtony w górę	P5	Monofoniczny, polifoniczny, perkusyjny
tr6	Transpozycja o oktawę w górę, a następnie 7 półtonów w górę	P6	Monofoniczny, polifoniczny, perkusyjny
tr7	Transpozycja o oktawę w dół, a następnie 4 półtony w górę	P7	Monofoniczny, polifoniczny, perkusyjny
tr8	Transpozycja o oktawę w dół, a następnie 7 półtonów w górę	P8	Monofoniczny, polifoniczny, perkusyjny

Tab. 2.5. Tabela TRPT z przykładowymi zdefiniowanymi regułami transformacyjnymi. Opracowanie własne.

Reguły transformacyjne

Po wykorzystaniu reguły generowania wzorca, wybierana jest reguła transformacyjna, za pomocą której dokonywane są różne operacje na wygenerowanym przez mini-generatory wzorcu muzycznym. Zdefiniowane przez autora reguły transformacyjne zostały ujęte w postaci tabeli prawdopodobieństwa reguł transformacyjnych *TRPT* (*Transformational Rules Probability Table*). Tabela *TRPT* może być zinterpretowana jako rozszerzona wersja tabeli prawdopodobieństwa wykorzystywanej w kontekście muzycznym przez Mirande w [85] na stronie 68. Poza zdefiniowaniem prawdopodobieństwa wylosowania danej reguły transformacyjnej, autor dodał informację na temat klasy K mini-generatora Mg, który może korzystać z danej reguły transformacyjnej. Tab. 2.5 zawiera przykładową tabelę *TRPT* zawierającą reguły transformacyjne zdefiniowane przez autora.

Zastosowanie zdefiniowanej reguły transformacyjnej na wygenerowanym wzorcu muzycznym wprowadza do prezentowanego systemu transformacyjny charakter algorytmu, zdefiniowany przez Woollera i innych w [104].

Ostatecznie więc struktura k-tego wzorca jest opisana jako

$$w_k = \{l, MG, GR, TRPT\} \tag{2.20}$$

gdzie:
— l oznacza definiowaną długość wzorca wyrażoną w liczbie potencjalnych dźwięków muzycznych budujących dany wzorzec,
— MG stanowi zbiór mini-generatorów generujących k-ty wzorzec,
— GR stanowi zbiór reguł generowania wykorzystanych przez mini-generatory ze zbioru MG,
— $TRPT$ stanowi tabelę reguł transformacyjnych wykorzystywanych do przetworzenia k-tego wzorca.

Wygenerowane wzorce muzyczne ostatecznie tworzą strukturę zawierającą k macierzy nut, które są następnie wykorzystywane w procesie generowania kompozycji wyjściowej. Jednak, jak można usłyszeć w dołączonych do pracy utworach muzycznych wygenerowanych przez autorski system, kompozycje wygenerowane z wykorzystaniem mini-generatorów zbudowane są ze wzorców o zróżnicowanej strukturze. Ta cecha kompozycji spowodowana jest zaproponowanym przez autora sposobem budowy zbioru MG mini-generatorów tworzących wzorzec muzyczny oraz definiowaniem klas wzorców zbudowanych w oparciu o taki sam zbiór mini-generatorów.

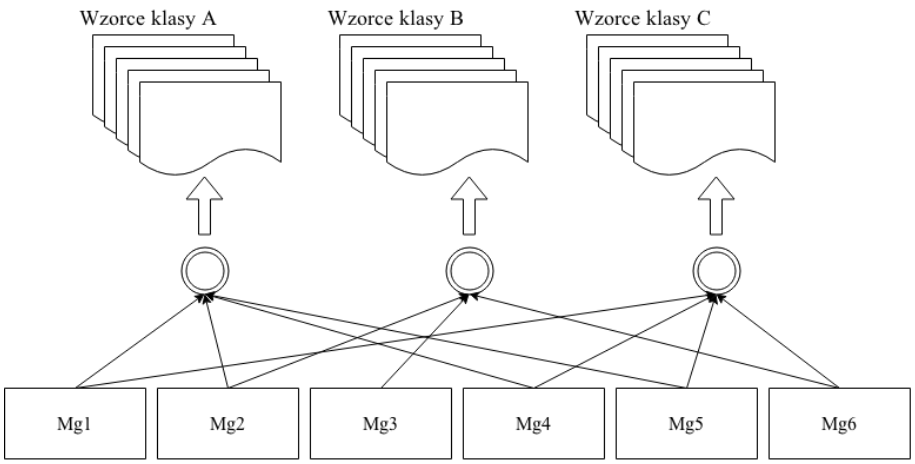
Na rys. 4.2 przedstawiona została przykładowa architektura budowy zbiorów MG mini-generatorów dla trzech przykładowych klas wzorców: A, B i C. Każda klasa zbudowana jest ze wzorców muzycznych, które są generowane przez wybrane mini-generatory. W przykładzie z rys. 4.2 zdefiniowane zostały następujące zbiory mini-generatorów dla poszczególnych klas wzorców muzycznych:
— $MG_A = \{Mg1; Mg2; Mg4; Mg5\}$, dla klasy A;
— $MG_B = \{Mg2; Mg3; Mg6\}$, dla klasy B;
— $MG_C = \{Mg1; Mg4; Mg5; Mg6\}$, dla klasy C.

Założenia te oznaczają, że wzorce muzyczne klasy A będą budowane w oparciu o zbiór MG_A mini-generatorów, czyli mini-generatory: $Mg1, Mg2, Mg4, Mg5$. Wzorce klasy B będą generowane na podstawie zbioru MG_B, natomiast w klasie C zawarte będą wzorce muzyczne utworzone na podstawie danych wygenerowanych przez mini-generatory zbioru MG_C.

Generowanie wzorców muzycznych w poszczególnych klasach przebiega w ten sam sposób z wykorzystaniem operacji konkatenacji zdefiniowanej we wzorze (2.6). Macierze nut wygenerowane przez poszczególne mini-generatory danego zbioru są łączone w jedną macierz nut tworzącą wzorzec muzyczny. Konfiguracja konkatenacji macierzy nut leży w gestii użytkownika systemu (nie jest mechanizmem automatycznym), a przykłady implementacji tego mechanizmu zostały opisane w rozdziale trzecim. W dalszej części tej pracy omówiony zostanie etap tworzenia kompozycji muzycznej na podstawie utworzonych przez mini-generatory wzorców należących do różnych klas (zbudowanych na podstawie różnych zbiorów MG).

2.2.2. Generowanie kompozycji wyjściowej

Proces generowania kompozycji muzycznej w prezentowanej koncepcji związany jest doborem wcześniej utworzonych wzorców muzycznych, które są kolejno łączone w sposób szeregowy, do momentu uzyskania kompletnego utworu. Do selekcji odpowiednich wzorców muzycznych zastosowana została miara euklidesowa w jedno-



Rys. 2.4. Przykładowa architektura generowania wzorców. Opracowanie własne.

wymiarowej przestrzeni podobieństwa pomiędzy i-tym i j-tym wzorcem muzycznym, których podobieństwo określa funkcja dystrybucji klas wysokości dźwięków, uwzględniająca model Parncutt'a [86].

Odległość pomiędzy dwoma wzorcami muzycznymi zdefiniowano jako

$$l = \mid x_i - x_j \mid, \tag{2.21}$$

w którym x_i oraz x_j oznaczają histogramy i-tego oraz j-tego wzorca muzycznego, wyrażone jako wektor 12-sto elementowy przedstawiony we wzorze 2.14.

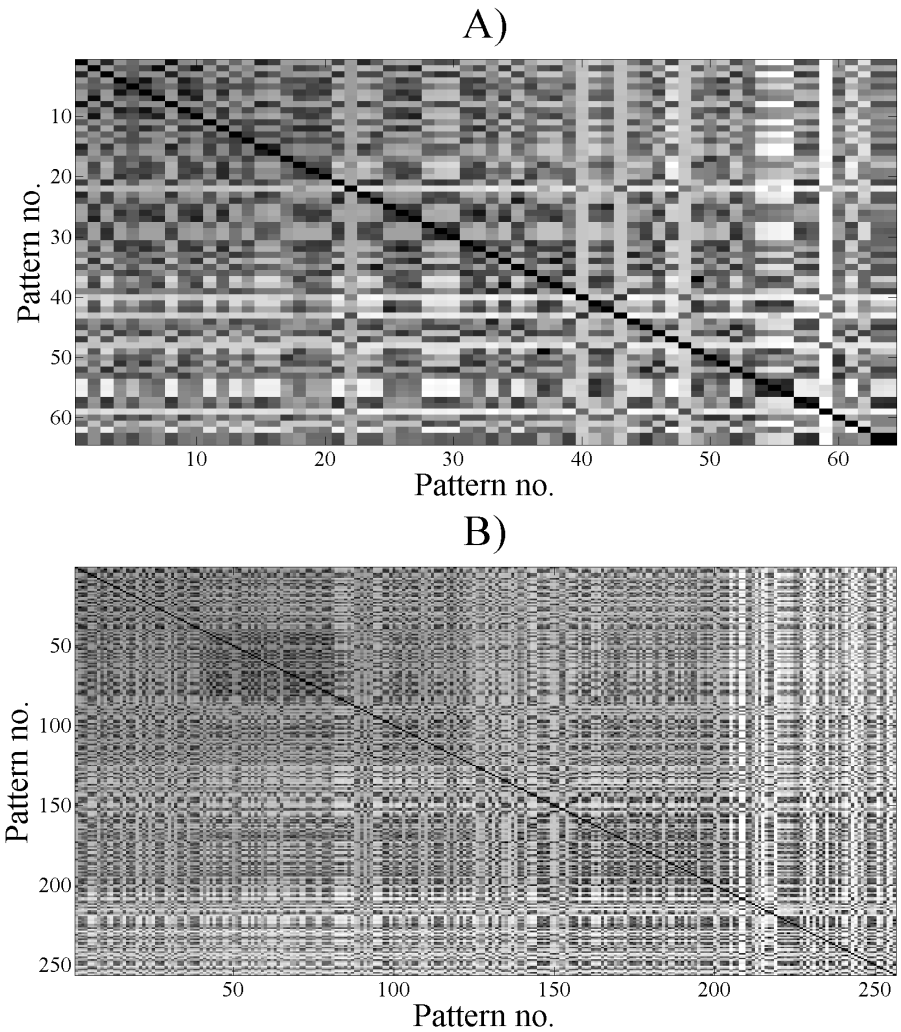
Dla wszystkich wygenerowanych i przetworzonych wzorców muzycznych tworzona jest macierz *PST* (*Pattern Transition Structure*) wykorzystywana podczas wyboru kolejnego wzorca muzycznego tworzącego kompozycję muzyczną. Przykładowe macierze *PST* dla 64-ech i 256-ciu wygenerowanych wektorów muzycznych zostały przedstawione na rysunku 2.5 .

Macierz *PST* zawiera wartości euklidesowej miary przedstawionej w wyrażeniu 2.21, których wartość jest na rysunku 2.5 interpretowana poprzez odcień szarości. Kolor czarny reprezentuje zerową odległość (l=0) pomiędzy i-tym a j-tym wzorcem muzycznym opisanym histogramem klas wysokości dźwięków. Kolor biały reprezentuje maksymalną odległość ($l = max$) pomiędzy i-tym a j-tym wzorcem w kontekście i-tego wiersza macierzy *PST*. Należy przy tym zaznaczyć, że im większa jest odległość pomiędzy danymi wzorcami muzycznymi tym bardziej różnią się one pod względem analizowanych cech (rozkładu klas wysokości dźwięku).

Pierwszy wzorzec muzyczny występujący w kompozycji wyjściowej jest wybierany losowo. W dalszej części z macierzy PST wybierany jest i-ty wiersz, przechowujący odległości pomiędzy pierwszym wylosowanym wzorcem, a wszystkimi innymi wygenerowanymi w systemie. Wartości i-tego wiersza są następnie sortowane rosnąco, tak by pierwszy element reprezentował najmniejszą wartość l. Wybór wzorca na podstawie uszeregowanych rosnąco wartości l ogranicza warunek

$$l_i <= \alpha, \tag{2.22}$$

w którym α wyznacza próg podobieństwa pomiędzy bieżącym i-tym wzorcem muzycznym a kolejnym j-tym wzorcem, który ma być dodany do kompozycji wyjściowej.



Rys. 2.5. Przykładowe macierze PST dla 64-ech (A) i 256-ciu (B) wygenerowanych wzorców muzycznych. Opracowanie własne

Ostatecznie kompozycja wyjściowa powstała w wyniku agregacji wzorców muzycznych jest opisana jako

$$U = \{s, I, W, n, rt\}, \tag{2.23}$$

gdzie:

— s oznacza liczbę kanałów *MIDI*, wykorzystanych w kompozycji U,
— I oznacza zbiór instrumentów *MIDI*, przypisanych do zdefiniowanych w kompozycji kanałów *MIDI*,
— W oznacza zbiór wzorców muzycznych o mocy n,
— n jest daną liczbą całkowitą określającą liczbę wzorców muzycznych, które mają być wykorzystane w kompozycji muzycznej,
— rt jest tablicą, zawierającą liczby całkowite interpretowane w procesie komponowania jako liczba powtórzeń wylosowanego wzorca ze zbioru W w kompozycji U.

Należy przy tym dodać, że podczas komponowania liczba powtórzeń wybranego wzorca ze zbioru W jest losowana z tablicy rt. Na długość wynikowej kompozycji

muzycznej wpływają: długość wzorców muzycznych wykorzystanych w kompozycji oraz łączna liczba wylosowanych powtórzeń z tablicy rt.

Ostatecznie długość kompozycji U przyjmuje wartość daną wyrażeniem

$$d_U = \sum_{i=1}^{n} d_i * j, j \in rt, \tag{2.24}$$

gdzie:

— n oznacza zdefiniowaną liczbę wzorców muzycznych przedstawioną we wzorze 2.23 ,

— d_i oznacza długość i-tego wzorca muzycznego, wyrażoną w sekundach bądź jednostkach bpm,

— j oznacza liczbę powtórzeń wylosowaną z tablicy rt i zastosowaną do i-tego wzorca w kompozycji U.

Wygenerowana przez prezentowany system kompozycja muzyczna może być następnie zapisana do formatu pliku *MIDI* i dalej przetwarzana.

Ostatnim krokiem w uzyskaniu wyniku końcowego jest instrumentacja i przetworzenie kompozycji przez syntezator sprzętowy lub programowy. Ponieważ instrumentacja utworu jest elementem wspólnym dla obu przedstawionych koncepcji, autor ulokował jej opis w rozdziale 2.4.

2.3. Komponowanie z uwzględnieniem opisu formy generowanego utworu

Koncepcja generowania kompozycji przedstawiona w poprzednim rozdziale, posiada następujące ograniczenia:

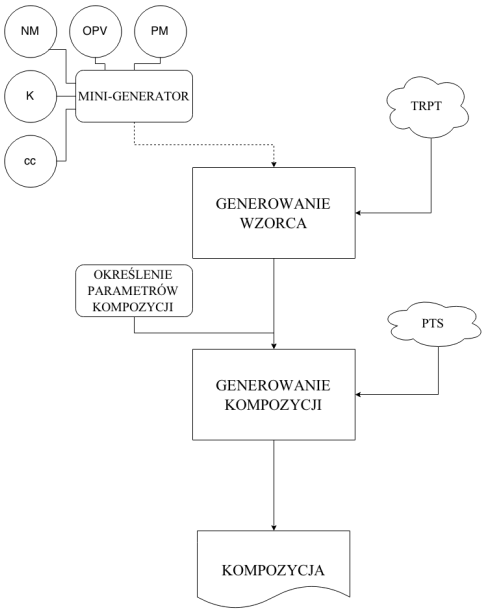
— brak możliwości definiowania struktury kompozycji na wysokim poziomie abstrakcji (formy utworu, uwzględniającej występujące w nim wyraziste fragmenty muzyczne, takie jak wstęp, rozwinięcie czy zakończenie),

— brak możliwości generowania partii harmonii, zgodnie z przyjętymi zasadami teorii muzyki (np. z wykorzystaniem *kadencji*),

— wzorce muzyczne generowane przez mini-generatory, posiadają stałe wartości rytmiczne w ramach konkretnego wzorca.

Analiza (przedstawiona w rozdziale 4) kompozycji generowanych przez system wykazuje, że pomimo przyjętych powyższych ograniczeń, kompozycje muzyczne utworzone przez system spełniają założenia dotyczące generowania kompozycji algorytmicznych na podstawie wzorców z gatunku muzyki tła. Jednakże w wyniku konsultacji autora ze środowiskiem naukowym i muzycznym, na drodze debat, konsultacji i recenzji, zrodził się pomysł poszerzenia omówionej wcześniej koncepcji systemu o funkcjonalności związane z przedstawionymi ograniczeniami.

System algorytmicznego komponowania muzyki, przedstawiony wcześniej przez autora w rozdziale 2.2, przyjął teraz strukturę zaprezentowaną na rys. 2.6. Najważniejsza zmiana dotyczy sposobu generowania wzorców, z których tworzona jest wyjściowa kompozycja muzyczna. Dodatkowo autor zrezygnował z etapu oceny podobieństwa wygenerowanych wzorców, z racji przyjętego mechanizmu generowania i łączenia wzorców w ramach kompozycji. Z tego względu w kolejnym rozdziale zaprezentowano koncepcję generowania wzorców, w której głównym elementem jest generowanie progresji akordów.



Rys. 2.6. Ogólna architektura generatywnego systemu algorytmicznego komponowania muzyki tła - podejście z uwzględnieniem opisu formy generowanego utworu. Opracowanie własne.

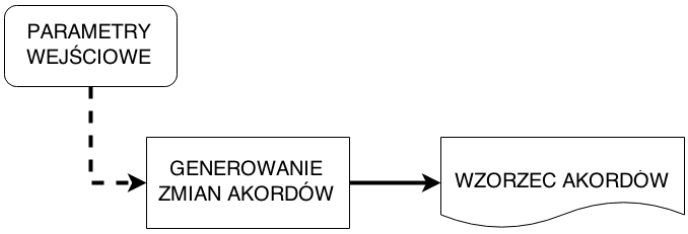
2.3.1. Generowanie wzorców

Przed zaprezentowaniem koncepcji generowania wzorców należy cofnąć się do opisu procesu komponowania zaproponowanego przez Bilesa w [75] i ujętego w tab. 2.1. Biles w swoich rozważaniach na temat modelowania procesu komponowania muzyki poprzez zdefiniowanie elementarnych zadań, które kształtują ten proces, wspomniał , że rozmieszczenie zadań ulokowanych w tab. 2.1 zostało wykonane w sposób arbitralny. System, który w sposób mniej lub bardziej kompleksowy generuje kompozycje muzyczne, może bazować na różnych zadaniach procesu komponowania, nie koniecznie w kolejności przedstawionej w tab. 2.1. Z tego względu autor w tej części pracy zaproponował odmienne podejście do problemu algorytmizacji procesu komponowania muzyki, bazując na harmonizacji jako wejściowym etapie generowania wzorców.

Na rys. 2.7 zaprezentowano w sposób uproszczony proces powstawania wzorca zawierającego wygenerowaną sekwencję akordów

$$W_A = \{A_p, n_A, l_A, S, R_z\}. \tag{2.25}$$

Do parametrów wejściowych (rys. 2.7) kształtujących proces generowania wzorca sekwencji akordów W_A zaliczone zostały:
— A_p - wygenerowany akord początkowy,
— n_A - liczba akordów, które mają być wygenerowane w sekwencji,
— l_A - długość akordu (w jednostkach bpm),
— S - skala muzyczna, na podstawie której generowana będzie sekwencja,
— R_z - zbiór reguł tworzenia zmian (następczości) generowanych akordów.



Rys. 2.7. Generowanie zmian akordów. Opracowanie własne.

Akord początkowy A_p jest macierzą nut MN zdefiniowaną jako

$$MN = \begin{bmatrix} p_1 m \\ \vdots \\ p_i n \end{bmatrix}, \tag{2.26}$$

przy czym p określa pojedynczy sparametryzowany dźwięk muzyczny w macierzy MN, natomiast m i n oznaczają stopnie wykorzystanej skali muzycznej (wtedy $m, n \leq 7$) oraz i wyznacza liczbę dźwięków wykorzystanych ($i \geq 3$) w początkowym akordzie A_p.

Parametr n_A określa liczbę kolejnych akordów A wygenerowanych w ramach wzorca W_A. Długość akordu l_A wyraża rozciągnięcie w czasie pojedynczego akordu A w jednostkach bpm.

Szerszego wyjaśnienia wymaga przedstawiony we wzorze (2.25) zbiór reguł R_z na podstawie którego budowana jest sekwencja akordów W_A. Zasady generowania sekwencji zostały zapisane w postaci reguł, bazujących na kadencjach, które wykorzystują pojęcie *funkcji harmonicznej*, budującej lub wygaszającej napięcie w utworze muzycznym.

W tab. 2.6 przedstawione zostały przykładowe kadencje zbudowane w tonacji C-dur. W kolumnie "Sekwencja akordów (stopnie)" znajdują się sekwencje akordów oznaczonych za pomocą cyfr rzymskich, które odpowiadają stopniom skali. Kolumna "Sekwencja akordów (nazwy)" zawiera przykłady sekwencji akordów dla tonacji C-dur.

Akordy zbudowane na kolejnych stopniach skali w systemie dur-moll tworzą funkcje harmoniczne, wśród których najczęściej występującymi są: *tonika*, *subdominanta* i *dominanta*. Obok wymienionych funkcji w systemie dur-moll występują także tzw. funkcje poboczne, czyli funkcje zbudowane na stopniach skali innych niż I, IV i V. Do tej grupy funkcji należą *medianta* i *submedianta* oraz akordy poboczne zbudowane na II i VII stopniu skali.

Zbiór reguł R_z ujęty we wzorze 2.25 został przez autora zdefiniowany jako macierz tranzycji (2.7), w której następstwo pojawienia się określonego akordu j (funkcji harmonicznej) uwarunkowane jest zdefiniowanym prawdopodobieństwem p_{ij} oraz aktualnie wygenerowanym akordem i. W nagłówkach kolumn i wierszy macierzy tranzycji znajdują się cyfry rzymskie, określające funkcje harmoniczne przedstawione w tab. 2.6. Dla każdego przejścia czyli wygenerowania akordu j będącego następnikiem akordu i, definiowane jest prawdopodobieństwo p_{ij} wystąpienia takiego zdarzenia.

Sekwencja akordów (stopnie)	**Sekwencja akordów (nazwy)**
I →IV (kadencja plagalna)	C-dur →F-dur
I →V (kadencja doskonała)	C-dur →G-dur
I →IV →V (triada harmoniczna)	C-dur →F-dur →G-dur
I →IV →V →IV	C-dur →F-dur →G-dur →F-dur
I →V →vi →IV	C-dur →G-dur →a-mol →F-dur
I →ii →IV →V	C-dur →d-mol →F-dur →G-dur
I →ii →IV	C-dur →d-mol →F-dur
I →VI →ii →V	C-dur →A-dur →d-mol →G-dur
I →VI →IV →V	C-dur →A-dur →F-dur →G-dur
IV →I →IV →V	F-dur →C-dur →F-dur →G-dur

Tab. 2.6. Przykładowe kadencje oparte na tonacji C-dur. Opracowanie własne.

	I	II	III	IV	V	VI	VII
I	p_{11}	p_{12}	p_{13}	p_{14}	p_{15}	p_{16}	p_{17}
II	p_{21}	p_{22}	p_{23}	p_{24}	p_{25}	p_{26}	p_{27}
III	p_{31}	p_{32}	p_{33}	p_{34}	p_{35}	p_{36}	p_{37}
IV	p_{41}	p_{42}	p_{43}	p_{44}	p_{45}	p_{46}	p_{47}
V	p_{51}	p_{52}	p_{53}	p_{54}	p_{55}	p_{56}	p_{57}
VI	p_{61}	p_{62}	p_{63}	p_{64}	p_{65}	p_{66}	p_{67}
VII	p_{71}	p_{72}	p_{73}	p_{74}	p_{75}	p_{76}	p_{77}

Tab. 2.7. Macierz tranzycji generowania sekwencji akordów. Opracowanie własne.

Budowa kadencji bazująca na przyjętej koncepcji macierzy tranzycji sekwencji akordów pozwala definiować zarówno typowe konfiguracje sekwencji (np. triada harmoniczna: tonika → subdominanta → dominanta), jak i sekwencje niestandardowe (np. funkcje poboczne oraz tzw. wtrącenia, czyli dodanie dominanty do akordu z istniejącej sekwencji).

Kolejnym krokiem w etapie generowania wzorca wykorzystywanego w przyjętej koncepcji jest konkatenacja wzorca W_A zdefiniowanego we wzorze 2.25 ze wzorcem melodii W_M wygenerowanej na podstawie sekwencji akordów W_A. Wzorzec melodii W_M jest macierzą nut wygenerowaną na podstawie parametrów przedstawionych w postaci

$$W_M = \{W_A, \{d_1, d_2, \ldots, d_n\}, \{o_1, o_2, \ldots, o_m\}, \{v_1, v_2, \ldots, v_s\}\}, \tag{2.27}$$

gdzie:

— W_A - wzorzec sekwencji akordów, na podstawie którego wygenerowana będzie melodia,

— $\{d_1, d_2, \ldots, d_n\}$ - wektor możliwych długości dźwięków występujących w melodii,

— $\{o_1, o_2, \ldots, o_m\}$ - wektor możliwych przesunięć dźwięków występujących w melodii,

— $\{v_1, v_2, \ldots, v_s\}$ - wektor możliwych głośności dźwięków występujących w melodii.

Algorytm 2.1 Generowanie melodii na podstawie sekwencji akordów. Opracowanie własne.

1. Oblicz długość l_A wzorca W_A,
2. Zdefiniuj pustą macierz nut dla wzorca melodii W_M i linii basowej W_B,
3. Ustaw indeks przeszukiwania i wzorca W_A na początek analizowanego fragmentu: $i = 0$,
4. Zdefiniuj okno analizy wzorca O_A: początek okna: $t_p \leftarrow i$, koniec okna: $t_k \leftarrow i + \Delta t$,
5. Dopóki $i + \Delta t \leq l_A$:
   a) Wybierz z okna analizy wzorca zbiór występujących we fragmencie wysokości dźwięków $\{h_1, h_2, \ldots, h_k\}$,
   b) Ustaw indeks przeszukiwania j okna O_A na: $j \leftarrow t_p$,
   c) Ustaw indeks położenia generowanego dźwięku melodii $r_{D_M} \leftarrow t_p$,
   d) Dopóki $j \neq t_k$:
      i. Dopóki $r_{D_M} \leq t_k$:
         A. Wylosuj z $\{h_1, h_2, \ldots, h_k\}$ wysokość dźwięku h,
         B. Wylosuj z $\{d_1, d_2, \ldots, d_n\}$ długość dźwięku d,
         C. Wylosuj z $\{o_1, o_2, \ldots, o_m\}$ przesunięcie dźwięku o,
         D. Wylosuj z $\{v_1, v_2, \ldots, v_s\}$ głośność dźwięku v,
         E. Utwórz dźwięk muzyczny D_M o wysokości h, długości d, głośności v oraz definiowanym przesunięciem w czasie o,
         F. Dodaj dźwięk muzyczny D_M do macierzy nut wzorca melodii W_M: $W_M \leftarrow \{W_M; D_M\}$,
      ii. Zwiększ indeks r_{D_M} o wartość położenia dźwięku muzycznego o: $r_{D_M} \leftarrow r_{D_M} + o$,
   e) Generuj wzorzec linii basowej W_B na podstawie zawartości wzorca W_M i aktualnego położenia okna O_A,
   f) Zwiększ indeks przeszukiwania j: $j \leftarrow j + i$,
   g) Przesuń indeks przeszukiwania i: $i \leftarrow i + \Delta t$,
   h) Przesuń okno analizy wzorca O_A: $t_p \leftarrow i$, $t_k \leftarrow i + \Delta t$,
6. Zwróć macierz nut wzorca W_M.

Algorytm 2.1 przedstawia koncepcję generowania melodii bazującą na analizie wzorca W_A sekwencji akordów. Mocnym ograniczeniem narzuconym przez autora jest wykorzystanie podczas tworzenia melodii wyłącznie dźwięków wchodzących w skład analizowanych akordów. Głównym elementem algorytmu 2.1 jest okienkowanie sygnału (wzorca W_A) oknem O_A o zadanej rozpiętości czasowej $< t_p, t_k >$, które zawiera wydzielony fragment wzorca. Podczas przesuwania okna wzdłuż wzorca W_A generowane są kolejne dźwięki D_M melodii, która ostatecznie przybiera postać macierzy nut wzorca W_M. Jest to podejście częściowo stochastyczne, częściowo deterministyczne, ponieważ dźwięki melodii losowane są ze zbioru możliwych dźwięków występujących w akordach, natomiast ich rozłożenie w czasie jest z góry ograniczone występowaniem akordów w macierzy wzorca W_A.

Uzupełnieniem nowej koncepcji generowania wzorca muzycznego jest wprowadzenie mechanizmu tworzenia wzorca linii basowej (oznaczonego jako W_B) na podstawie generowanego wzorca W_M. Wymienioną funkcjonalność, o której wspomniano w kroku 5*e* w algorytmie 2.1, przedstawiono szerzej w algorytmie 2.2.

Algorytm 2.2 Generowanie linii basowej na podstawie melodii. Opracowanie własne.

1. Wybierz fragment wzorca melodii W_M zgodnie z położeniem okna analizy O_A,
2. Zdefiniuj parametr k jako liczbę losową z przedziału $< 1, \ldots, l >$, gdzie l stanowi liczbę wszystkich dźwięków muzycznych w wybranym fragmencie wzorca W_M,
3. Z wybranego fragmentu macierzy nut wzorca melodii wybierz n dźwięków, pomijając co k-ty element,
4. Transponuj o m oktawy w dół dźwięki wybrane w punkcie 3,
5. Dodaj transponowane dźwięki do macierzy wzorca W_B linii basowej.

Generowanie linii basowej polega na fragmentacji linii melodycznej wzorca W_M oraz transponowaniu o m oktawy w dół. Ten prosty zabieg powoduje poszerzenie spektrum częstotliwości dźwięków muzycznych w wygenerowanej kompozycji, a linia basowa jest ściśle związana z melodią główną. Należy przy tym dodać, że dla każdego okna analizy proces generowania linii basowej rozpoczyna się na nowo, co w konsekwencji powoduje urozmaicenie rytmiczne występujących we wzorcu W_B dźwięków.

W ramach przedstawionego procesu generowania wzorców autor założył również możliwość stosowania zdefiniowanego wcześniej (2.9) mini-generatora, którego idea została opisana szerzej w rozdziale 2. 2. Należy jednak przedstawić zasadniczą różnicę pomiędzy wykorzystaniem mini-generatora w koncepcji generowania utworu muzycznego, w którym nie została zdefiniowana forma kompozycji, a koncepcją, w której opis formy utworu stanowi centralny punkt procesu komponowania.

W rozdziale 2. 2. mini-generatory zostały przedstawione jako mechanizm generowania wszelkich wzorców muzycznych, wykorzystywanych później w procesie generowania kompozycji wyjściowej. Koncepcja komponowania utworu z uwzględnieniem definiowania jego formy wykorzystuje mini-generatory, w których parametr MM przedstawiony we wzorze (2.9) przyjmuje postać skalara - wartości odpowiadającej konkretnemu instrumentowi perkusyjnemu zdefiniowanemu w standardzie General Midi. Dodatkowo parametr K oznaczający klasę mini-generatora jest wspólny dla wszystkich mini-generatorów wykorzystanych podczas generowania wzorca i oznaczony jako klasa generatora perkusyjnego ze zdefiniowanymi regułami generowania wzorca $gr5$ i $gr6$ (tab. 2.2 na stronie 58).

Kolejnym elementem przedstawianej w tym rozdziale koncepcji jest generowanie kompozycji wyjściowej, przy czym w dalszej części pracy autor przedstawił redefinicję pojęcia kompozycji wyjściowej oraz procesu jej generowania. Zmiany opisane dalej są spowodowane przyjęciem odmiennych założeń w stosunku do procesu komponowania oraz odmienną budową wzorca muzycznego.

2.3.2. Generowanie kompozycji wyjściowej

Istotną zmianą w stosunku do przedstawionej w rozdziale 2.2. koncepcji generowania kompozycji wyjściowej jest pominięcie etapu selekcji wzorców i generowanie utworu muzycznego na podstawie modelu kompozycji $\hat{U}$

$$\hat{U} = \{PW, F = \{i_{f_1}; i_{f_2}; \ldots; i_{f_n}\}, S, I, \{MG_{f_1}; MG_{f_2}; \ldots; MG_{f_n}\}\}, \tag{2.28}$$

w którym:
— PW - oznacza zbiór parametrów wejściowych kompozycji,
— $F = \{i_{f_1}; i_{f_2}; \ldots; i_{f_n}\}$ - oznacza zbiór identyfikatorów zdefiniowanych fragmentów muzycznych $f_1 \ldots f_n$ opisujący ich rozmieszczenie w kompozycji,
— S - oznacza zbiór kanałów *MIDI* wykorzystanych w kompozycji,
— I - oznacza zbiór instrumentów *MIDI* przypisanych do zdefiniowanych w kompozycji kanałów *MIDI*,
— $\{MG_{f_1}; MG_{f_2}; \ldots; MG_{f_n}\}$ - oznacza kolekcję zbiorów MG mini-generatorów przypisanych do poszczególnych fragmentów kompozycji $f_1 \ldots f_n$, przy czym n stanowi liczbę zdefiniowanych fragmentów kompozycji.

Do parametrów wejściowych PW kształtujących kompozycję wyjściową systemu należą:
— tempo utworu (liczba całkowita, wyrażająca szybkość odtwarzania kompozycji w jednostkach bpm),
— przyjęta skala muzyczna, w której tworzona będzie kompozycja,
— definicje fragmentów muzycznych, z których będzie generowana kompozycja,
— długości i liczby akordów wchodzących w skład zdefiniowanych fragmentów kompozycji.

Skala muzyczna została zdefiniowana jako wektor 8-mio elementowy, w którym znajdują się wartości interwałów kolejnych stopni skali, wyrażone w liczbie półtonów. Pierwszym elementem tego wektora jest liczba 0, która opisuje pierwszy stopień skali - wartość odniesienia dla pozostałych stopni. Ostatni element wektora został zdefiniowany jako interwał oktawy wyrażony wartością 12 półtonów. Przykładowo dla naturalnej skali durowej zdefiniować można następujący wektor interwałów: $< 0, 2, 4, 5, 7, 9, 11, 12 >$. W takim wypadku dla dźwięku $C4$, dla którego wysokość wyrażona wg standardu *MIDI* wynosi 60, otrzymujemy naturalną skalę C-dur w następujący sposób:

$60+ < 0, 2, 4, 5, 7, 9, 11, 12 >=< 60, 62, 64, 65, 67, 69, 71, 72 >$, przy czym ostatnim elementem tak utworzonego wektora jest dźwięk $C5$, należący już do kolejnej oktawy.

Fragment muzyczny stanowi zdefiniowaną część kompozycji przedstawioną w postaci macierzy nut zbudowanej na podstawie wybranych wzorców muzycznych. Formalny zapis fragmentu jest zdefiniowany jako

$$f = \{i_f, S_f, I_f, W_{A_f}, W_{M_f}, W_{B_f}, MG_f\}, \tag{2.29}$$

gdzie:
— i_f jest identyfikatorem danego fragmentu f,
— S_f jest zbiorem kanałów midi wykorzystanych do wygenerowania fragmentu f,
— I_f stanowi zbiór instrumentów *MIDI* przypisanych do zdefiniowanych we fragmencie f kanałów *MIDI*,
— W_{A_f} jest wzorcem sekwencji akordów przypisanym do fragmentu f,
— W_{M_f} jest wzorcem melodii wygenerowanej na podstawie W_{A_f} i przypisanym do fragmentu f,
— W_{B_f} jest wzorcem linii basowej wygenerowanej na podstawie W_{M_f} i przypisanym do fragmentu f,
— MG_f stanowi zbiór mini-generatorów przypisanych do fragmentu f.

Identyfikator i_f fragmentu służy do jego lokalizowania w zbiorze F (2.28) opisującym formę generowanej przez system kompozycji muzycznej. Przykładowo dla zdefinio-

wanych fragmentów kompozycji f_1, f_2, f_3 i f_4 można zdefiniować ich identyfikatory: $i_1 = 1$, $i_2 = 2$, $i_3 = 3$ oraz $i_4 = 4$. Dla tak zdefiniowanych danych zbiór F może przykładowo przybrać postać $F = \{1, 1, 2, 2, 2, 2, 3, 3, 2, 2, 2, 2, 3, 3, 4, 4\}$. Zbiór ten należy analizować od lewej do prawej strony, przy czym kolejność elementów zbioru jest istotna. W przedstawionym przykładzie zbioru F można zauważyć, że fragment f_1 o identyfikatorze $i_1 = 1$ występuję dwukrotnie na początku kompozycji. Następnie naprzemiennie występują dwie grupy fragmentów: czterokrotnie powtórzony fragment f_2 ($i_2 = 2$) oraz dwukrotnie powtórzony fragment f_3 ($i_3 = 3$). Kompozycja kończy się dwukrotnym wystąpieniem fragmentu f_4 o identyfikatorze $i_4 = 4$.

S_f i I_f (2.30) stanowią podzbiory występujących kolejno w (2.28) zbiorów: S kanałów *MIDI* oraz I instrumentów *MIDI* wykorzystanych w kompozycji $\hat{U}$. Na tym etapie badań autor założył występowanie w danym fragmencie jednego wzorca sekwencji akordów W_{A_f}, na którym bazuje mechanizm generowania melodii i linii basowej. Istnieje natomiast możliwość generowania więcej niż jednej melodii i linii basowej dla wygenerowanego wzorca W_{A_f}, co zostało przedstawione w części pracy poświęconej eksperymentom i wynikom badań (rozdział 4). W ramach rozdziału 4 zaprezentowane zostały również różnorodne możliwości definiowania zbioru mini-generatorów MG_f dla danego fragmentu.

Ostatnimi parametrami ujętymi w zbiorze parametrów wejściowych PW są długości akordów (wyrażone w jednostkach bpm) zdefiniowane niezależnie dla każdego fragmentu kompozycji oraz liczby akordów w ramach wzorców sekwencji akordów W_{A_f} związanych z fragmentami kompozycji. Wymienione parametry służą do określenia długości wzorca sekwencji akordów l_{WA_f} wyrażonej wzorem

$$l_{WA_f} = l_A * n, \tag{2.30}$$

gdzie:
— l_A - długość akordu (w bpm),
— n - liczba akordów generowanych we wzorcu W_{A_f}.
Długość wzorca sekwencji akordów W_{A_f} jest wykorzystywana podczas generowania melodii i lini basowej (algorytm 2.1). Odpowiednio dobrane l_A oraz n we wzorcach sekwencji umożliwiają generowanie kompozycji w różnym metrum (l_A) oraz z różnym poziomem złożoności fragmentów kompozycji (n).

Końcowym etapem generowania kompozycji dla koncepcji prezentowanych w rozdziałach 2.2 oraz 2.3 jest dobór instrumentów wykorzystanych w wygenerowanej kompozycji oraz synteza utworu muzycznego. Z tego względu ta wspólna część procesu komponowania została omówiona w dalszej części rozdziału 2.

2.4. Instrumentacja i synteza kompozycji

Autor w swoich badaniach nie napotkał na pracę jednoznacznie rozwiązującą problem automatycznego doboru instrumentu do wygenerowanej przez maszynę linii melodycznej bądź akompaniamentu. Nawet w przytoczonych analizowanych rozwiązaniach z literatury światowej problem ten jest traktowany marginalnie, tzn. albo dana praca nie porusza tego tematu, albo jasno stwierdza się fakt sztywnego przypisania instrumentu do ścieżki muzycznej.

W przedstawionym podejściu zakłada się, że kompozycje wygenerowane przez system są zapisywane w postaci plików *MIDI*, a ostateczny dobór instrumentów

wykonujących leży po stronie użytkownika. Podstawowa wersja kompozycji posiada sztywne przypisanie domyślnego pierwszego instrumentu z banku brzmień *MIDI* (instrument *Piano* ze standardowego banku *GM - General MIDI*) do kanałów *MIDI* zdefiniowanych w kompozycji. Wyjątkiem jest kanał 10-ty zarezerwowany domyślnie dla instrumentów perkusyjnych. Dalsza instrumentacja może się odbywać w zewnętrznej aplikacji oferującej możliwość prostej edycji plików *MIDI*.

Ostatecznie kompozycja muzyczna otrzymuje brzmienie, które ściśle zależy od środowiska syntezatora programowego bądź sprzętowego. Jakość odbioru może być zachowana poprzez syntezę kompozycji za pomocą programowego syntezatora oferującego wysokiej jakości brzmienia instrumentów, a następnie zachowanie odtwarzanego utworu w formacie MP3 bądź WAV. Wyniki pracy prezentowane przez autora są przechowywane w formacie FLAC, tak aby Czytelnik mógł zapoznać się z kompozycjami wygenerowanymi i syntezowanymi za pomocą urządzeń i oprogramowania dostępnego autorowi podczas badań.

Podczas publikowania swoich osiągnięć autor często zmierzał się z problemem odbioru wygenerowanych przez system kompozycji w kontekście jakości brzmienia. Należy więc uzmysłowić Czytelnikowi różnicę pomiędzy kompozycją a syntezą tej kompozycji przez urządzenie bądź program komputerowy.

Koncepcje prezentowanego systemu dotyczą algorytmicznego podejścia do komponowania utworu muzycznego. Komponowanie w tym kontekście polega na opracowaniu procedury systematyzującej problemy łączenia pewnych struktur muzycznych w całość tworzącą kompozycję, z uwzględnieniem faktu ograniczeń narzuconych przez przyjęty model. Natomiast problem syntezy tej kompozycji przez urządzenie bądź program komputerowy jest tematem pobocznym tej pracy. W następnym rozdziale omówiona została implementacja prezentowanych w rozdziałach 2.2 i 2.3 koncepcji w środowisku Matlab, uwzględniająca założenia i ograniczenia wysunięte w tym rozdziale przez autora.

Rozdział 3

Implementacja systemu algorytmicznego komponowania muzyki

Przedstawione w rozdziale 2 koncepcje systemu algorytmicznego komponowania muzyki z wykorzystaniem wzorców muzycznych zostały zaimplementowane w środowisku Matlab z wykorzystaniem frameworka *MIDI* Toolbox zaprezentowanego w [48]. W rozdziale zostały omówione implementacje koncepcji generowania kompozycji bez definiowania końcowej formy utworu oraz z podaną definicją struktury kompozycji. Kolejność prezentowania wymienionych rozwiązań została przedstawiona w oparciu o ewolucje koncepcji przedstawionej w rozdziale 2.2 w koncepcję zaprezentowaną w rozdziale 2.3.

3.1. Komponowanie bez definiowania formy kompozycji

Pierwszą w kolejności rozwijaną przez autora w rozdziale 2.2 koncepcją systemu algorytmicznego komponowania muzyki na podstawie wzorców jest system komponowania nie uwzględniający opisu formy końcowej kompozycji muzycznej. Przykładową pełną implementację tego podejścia przedstawiono na listingu 5.16. Na podstawie wymienionej implementacji autor w tej części pracy opisał najważniejsze składowe systemu komponowania. Przykładowe dalsze modyfikacje konfiguracji systemu i jego poszczególnych implementacji modułów opisano dokładniej w rozdziale 4.

Początkowym etapem komponowania jest zdefiniowanie parametrów i struktur wykorzystywanych do generowania wzorców muzycznych. Do tej grupy elementów należą:

— długość wzorca,
— liczba generowanych wzorców,
— liczba wygenerowanych wzorców, które będą wykorzystywane do generowania kompozycji,
— długości nut dla generowanych wzorców,
— struktura przechowująca wektory możliwych lokalizacji dźwięków w poszczególnych macierz nut generowanych wzorców,
— macierze nut, do których "wstawiane" będą wysokości dźwięku z wybranego motywu *MM*,
— macierz *KMM* - zbiór możliwych kombinacji motywu *MM*.

Definicja wymienionych parametrów została przedstawiona na listingu 3.1. W linii 1 została zdefiniowana długość wzorca *pattern_l*. W liniach 2 i 3 zdefiniowano odpowiednio liczbę generowanych wzorców jako zmienną n oraz liczbę wygenerowanych wzorców wykorzystanych do generowania kompozycji jako zmienną *seq_patterns*.

W dalszej kolejności w liniach 4-7 na listingu 3.1 zdefiniowane zostały szablony macierzy nut wykorzystane w procesie generowania wzorców muzycznych. Do generowania macierzy nut autor wykorzystał wbudowaną w *MIDI* Toolbox

List. 3.1. Fragment implementacji z listingu 5.16 - definiowania parametrów wejściowych w komponowaniu bez definiowania formy utworu. Środowisko Matlab. Opracowanie własne.

```
1  pattern_l = 16;
2  n = 32; %liczba generowanych wzorców
3  seq_patterns = 128; %liczba wzorców wykorzystanych w kompozycji
4  basepattern = createnmat(zeros(1,pattern_l),0.125,127,10);
5  bass_pattern = createnmat(zeros(1,pattern_l),0.25,127,1);
6  bass_pattern2 = createnmat(zeros(1,pattern_l),0.25,127,2);
7  bass_pattern3 = createnmat(zeros(1,pattern_l),0.25,127,3);
8  bassoctave = KMM()
9  [rhytmic_step, rhytmic_value] = rhytmicStructure3(pattern_l);
10 temppattern = basepattern;
11 pcdistribution = zeros(n,12);
12 pattern_struct = [];
13 CC_size = size(bassoctave,1);
14 ...
```

funkcję *createnmat*. Funkcja pobiera cztery parametry: wektor wysokości dźwięków, długość dźwięków w sekundach, wartość głośności dźwięków oraz numer kanału *MIDI* wykorzystywanego przez dźwięki macierzy nut. Na podstawie tych parametrów po wywołaniu funkcji *createnmat* otrzymujemy macierz nut. Wywołania funkcji *createnmat* przedstawione w liniach 4-7 na listingu 3.1 powodują utworzenie macierzy nut *basepattern*, *bass_pattern*, *bass_pattern*2 oraz *bass_pattern*3 , zawierających $zeros(1, pattern_l)$ wierszy - dźwięków muzycznych, o długości 0.125 bpm (macierz *basepattern*) oraz 0.25 bpm, głośności 127, ulokowanych w następujących kanałach *MIDI*: 10 - macierz *basepattern*, 1 - macierz *bass_pattern*, 2 - macierz *bass_pattern*2 oraz 3 - macierz *bass_pattern*3. W wygenerowanej macierzy nut, w kolumnie przechowującej wysokości dźwięków, znajdować się będą elementy zerowe, ponieważ w wywołaniu funkcji *createnmat* jako wektor wysokości dźwięków został podany wektor wygenerowany za pomocą wbudowanej w środowisko Matlab funkcji *zeros*.

W liniach 8 i 9 na listingu 3.1 zdefiniowane zostały przykładowe struktury macierzy KMM (funkcja KMM) oraz wektorów WPN dla kanałów *MIDI* wykorzystanych przy generowaniu macierzy nut dla wzorców muzycznych (funkcja *rhytmicStructure*3). Pełne implementacje przykładowych funkcji KMM i *rhytmicStructure*3 zostały zamieszczone odpowiednio na listingach 5.17 i 5.18 w części "Dodatek A. Listingi kodów".

Funkcja KMM z listingu 5.17 generuje macierz *bassoctave* przechowującą permutacje trzech motywów, których strukturę podano w postaci wektorów 6-cio elementowych: $< 24, 26, 28, 31, 33, 36 >$, $< 36, 38, 40, 43, 45, 48 >$ i $< 24, 26, 28, 31, 48, 52 >$. Permutacje wektorów zostały wygenerowane za pomocą wbudowanej w środowisko Matlab funkcji *perms*, a wyniki jej działania zostały skonstatowane pionowo do postaci macierzy *bassoctave*. Następnie macierz *bassoctave* została poszerzona o kolejnych sześć kolumn, w których umieszczone zostały transponowane o oktawę wersje permutacji wymienionych wcześniej wektorów. Ostatecznie macierz *bassoctave* przyjmuje postać jak na (3.1).

$$bassoctave = \left[\begin{array}{cccccc|cccccc} 36 & 33 & 31 & 28 & 26 & 24 & 48 & 45 & 43 & 40 & 38 & 36 \\ 36 & 33 & 31 & 28 & 24 & 26 & 48 & 45 & 43 & 40 & 36 & 38 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 24 & 52 & 48 & 31 & 28 & 26 & 36 & 64 & 60 & 43 & 40 & 38 \end{array}\right] \tag{3.1}$$

Po lewej stronie macierzy zawarte są permutacje wektorów natomiast prawa strona zawiera transponowane wersje permutacji motywów. Dla tak zdefiniowanej macierzy KMM pojedynczy wiersz stanowi motyw muzyczny MM wykorzystywany do generowania wzorca muzycznego.

Wywołana w linii 9 na listingu 3.1 funkcja *rhytmicStructure*3 tworzy struktury opisujące rytmiczny rozkład dźwięków w ramach wzorca muzycznego. Funkcja zwraca dwie struktury: *rhytmic_step* - macierz jednokolumnową $n \times 1$ oraz *rhytmic_value* - macierz komórek składającą się z n wektorów kolumnowych.

Macierz jednokolumnowa *rhytmic_step* przechowuje wartości wyznaczające współczynnik *cc* opisany (2.12) na stronie 58. Liczba wierszy tej macierzy wyznacza potencjalnie różne wartości obliczanych współczynników *cc*, a więc i teoretycznie liczbę mini-generatorów zdefiniowanych w systemie, przy założeniu jednokrotnego wykorzystania współczynnika *cc* przez każdy z mini-generatorów. Dodatkowo wartości macierzy *rhytmic_step* są zdefiniowane na podstawie długości wzorca, oznaczonej jako *pattern_l* w linii 1 na listingu 3.1 oraz wykorzystanej przez funkcję *rhytmicStructure*3 w linii 9. W przykładowej implementacji funkcji *rhytmicStructure*3 przedstawionej na listingu 5.18 zdefiniowano 11 niezależnych wartości, które mogą być wykorzystane do generowania współczynników *cc* dla mini-generatorów. Dla przejrzystości generowana struktura *rhytmic_step* została utworzona z wykorzystaniem wcześniej zdefiniowanych 11 zmiennych o nazwach: *bass_step*, *snare_step*, *closedhihat_step* ... *clap_step*. Następnie zmienne te zostały połączone za pomocą konkatenacji wertykalnej w jedną macierz jednokolumnową.

Dla implementacji przedstawionej na listingu 5.18 oraz długości wzorca $pattern_l = 16$ macierz *rhytmic_step* przybiera postać jak na (3.2). Wartości przechowywane w macierzy *rhytmic_step* nie muszą należeć do zbioru liczb naturalnych, ponieważ w dalszej kolejności są wykorzystywane do wyznaczenia współczynników *cc*, w którym zastosowano przybliżanie wartości w kierunku zera.

$$rhytmic_step = \begin{bmatrix} 4 \\ 4 \\ 8 \\ 8 \\ 4 \\ 4 \\ 4 \\ 5.33 \\ 8 \\ 8 \\ 8 \end{bmatrix} \tag{3.2}$$

Drugą ze zwracanych przez funkcję *rhytmicStructure*3 struktur jest macierz komórek *rhytmic_value* zawierająca dokładnie tyle samo elementów co macierz jednokolumnowa *rhytmic_step*. Każda z komórek przechowuje macierz jednowierszową, w której zdefiniowane są potencjalne pozycje dźwięków muzycznych wykorzystane następnie do generowania wektora WPN opisanego (2.10) na stronie 56.

Dla implementacji przedstawionej na listingu 5.18 oraz długości wzorca $pattern_l = 16$ struktura macierzy komórek *rhytmic_value* przybiera postać jak na (3.3). Każda komórka macierzy zawiera wektor składający się z liczb całkowitych interpretowanych jako kolejne lokalizacje dźwięków muzycznych w ramach generowanego wzorca.

$$rhytmic_value = \begin{bmatrix} [1,5,9,13] \\ [2,5,8,11,14] \\ [1,4,7,10,13,16] \\ [8,10,12,14,16] \\ [8,10,12,14,16] \\ [1,5,9,13] \\ [9,13] \\ [1,4,7,10,13,16] \\ [1,2,\ldots,16] \\ [2,3,4,5,6,7,8] \\ [1,4,7,10,13,16] \end{bmatrix} \tag{3.3}$$

Struktury przedstawione w (3.2) i (3.3) są następnie wykorzystane w procesie generowania wzorców muzycznych. W dalszej części tego rozdziału opisano proces tworzenia wzorców, wykorzystując przy tym implementację z listingu 5.16 umieszczoną w części "Dodatek A: Listingi kodów". Autor dokonał analizy fragmentów implementacji interpretowanych jako mini-generatory tworzące różne elementy generowanej przez system kompozycji muzycznej.

3.1.1. Generowanie wzorców muzycznych

Proces generowania wzorców muzycznych przez system jest realizowany w sposób iteracyjny w głównej pętli programu, której przykład realizacji przedstawiono w liniach 16 - 159 na listingu 5.16. Założenia teoretyczne dotyczące mini-generatorów wykorzystanych w komponowaniu utworu bez definiowania jego ostatecznej formy, przedstawiono w (2.9) na stronie 56. W przyjętej koncepcji zakłada się przynależność mini-generatora do jednej z trzech zdefiniowanych klas: generatora monofonicznego, polifonicznego oraz perkusyjnego. Implementacja z listingu 5.16 wykorzystuje mini-generatory klasy perkusyjnej i monofonicznej.

Na listingu 3.2 przedstawiono fragment implementacji generowania wzorców muzycznych z wykorzystaniem mini-generatorów perkusyjnych. Do generowania wykorzystane zostały struktury przedstawione w (3.2) i (3.3).

W głównej pętli programu tworzonych jest n wzorców muzycznych (linia 1, na listingu 3.2). W liniach 2 - 10 przedstawiono generowanie wzorców dla dwóch instrumentów perkusyjnych: bębna basowego (bass) oraz werbla (snare). Procedura polega na wyznaczeniu współczynnika *step* dla generowanego wzorca na podstawie zdefiniowanego dla instrumentu współczynnika z macierzy *rhytmic_step*, a następnie

List. 3.2. Fragment implementacji z listingu 5.16 - generowanie wzorców muzycznych przez mini-generatory perkusyjne w komponowaniu bez definiowania formy utworu. Środowisko Matlab. Opracowanie własne.

```
1  for i=1:n
2  %bass
3          step = fix( pattern_l/ fix(rand(1)*rhytmic_step(1)+1) );
4          temppattern(rhytmic_value{1}(1:step:end) , 4) = 36;
5          bass = temppattern;
6  %snare
7          temppattern = basepattern;
8      step = fix( pattern_l/ fix(rand(1)*rhytmic_step(2)+1) );
9          temppattern(rhytmic_value{2}(1:step:end) , 4) = 40;
10         snare = temppattern;
11 ...
12 %ornaments
13         temppattern = basepattern;
14         step = fix( pattern_l/ fix(rand(1)*rhytmic_step(7)+1) );
15         vec = rhytmic_value{8}(1:step:end);
16         temppattern(vec,4) = [60:1:60+length(vec)-1];
17         harmonization = temppattern;
18         temppattern = createnmat(zeros(1,pattern_l*2),0.25,127,10);
19         step = fix( pattern_l/ fix(rand(1)*rhytmic_step(7)+1) );
20         vec = rhytmic_value{7}(1:step:end);
21         temppattern(vec,4) = [60:1:60+length(vec)-1];
22         harmonization2 = temppattern;
23         temppattern = basepattern;
24 ...
25 end
```

wykorzystania go do ulokowania dźwięku *MIDI* danego instrumentu perkusyjnego (36 dla bębna basowego oraz 40 dla werbla) w ramach pustego wzorca *temppattern* przy wykorzystaniu wektora *rhytmic_value*. Analogicznie proces ten jest powtarzany dla pozostałych instrumentów perkusyjnych z listingu 5.16 z wyłączeniem mini-generatorów, które generują serie wystąpień różnych instrumentów perkusyjnych (zmienne *harmonization* i *harmonization*2 na listingu 3.2) z wykorzystaniem zadanego wektora jako motywu MM (definicja mini-generatora (2.9) na stronie 56.

W liniach 12 - 23 listingu 3.2 przedstawiono przykładowy fragment procesu generowania wzorców muzycznych przez mini-generatory perkusyjne bazujące na motywie MM zdefiniowanym jako wektor dźwięków *MIDI*, interpretowanych jako różne instrumenty perkusyjne. Początek tworzenia wzorców *harmonization* i *harmonization*2 jest analogiczny do tego zaprezentowanego dla mini-generatorów perkusyjnych generujących wzorce dla jednego instrumentu (wtedy motyw MM mini-generatora jest skalarem). W liniach 16 i 22 listingu 3.2 następuje wypełnienie wzorca muzycznego dźwiękami z wektora $< 60 : 1 : 60 - length(vec) - 1 >$ w lokalizacjach zdefiniowanych przez wektor *vec*, przy czym wektor ten definiowany jest niezależnie dla obu wzorców (w liniach 15 i 20).

List. 3.3. Fragment implementacji z listingu 5.16 - generowanie wzorców muzycznych przez mini-generator monofoniczny w komponowaniu bez definiowania formy utworu. Środowisko Matlab. Opracowanie własne.

```
1  for i=1:n
2  ...
3  %mini-generator monofoniczny 1
4          temppattern = bass_pattern;
5          step = fix( pattern_l / fix(rand(1)*rhytmic_step(9)+1) );
6          vec = rhytmic_value{9}(1:step:end);
7          mm_index = fix(rand(1)*CC_size+1);
8          mm = bassoctave(mm_index,:);
9          basspiano_vec = zeros(1,length(vec));
10         k=1;
11         for j=1:length(vec)
12                 basspiano_vec(j) = mm(k);
13                 if (k < length(mm))
14                         k=k+1;
15                 else
16                         k=1;
17                 end
18         end
19         if ( mod(i,2) == 1)
20                 basspiano_vec = basspiano_vec + 12;
21         end
22         temppattern(vec , 4) = basspiano_vec;
23         basspiano = temppattern;
24 ...
25 end
```

Odrębnego omówienia wymaga proces generowania wzorców muzycznych przez mini-generatory monofoniczne. Na listingu 3.3 przedstawiono przykładowy fragment implementacji generowania linii melodycznej (monofonicznej) przez mini-generator z wykorzystaniem struktur przedstawionych w (3.2) i (3.3).

Procedury wyznaczania współczynnika *step* oraz związanego z nim wektora *vec* lokalizacji dźwięków w ramach wzorca muzycznego są analogiczne do tych przedstawionych w przykładach implementacji mini-generatorów perkusyjnych. Odrębnie natomiast przebiega proces wypełniania wzorca muzycznego dźwiękami z motywu *mm*. Wzorzec *mm* losowany jest ze zbioru *bassoctave* wygenerowanych wcześniej kombinacji motywu, a jego dźwięki wstawiane są do odpowiednich lokalizacji wzorca muzycznego w obrębie pętli zaprezentowanej na listingu 3.3 w liniach 11 - 18. Jeżeli długość motywu *mm* jest mniejsza niż długość wektora *vec* lokalizacji dźwięków, dźwięki wstawiane są do wektora muzycznego od początku motywu *mm*. Dodatkowo wprowadzona została reguła transpozycji o oktawę w górę co drugiego wygenerowanego wzorca monofonicznego (linie 19 - 21 na listingu 3.3).

Ostatnim etapem generowania wzorców muzycznych jest łączenie wyjściowych wzorców muzycznych generowanych przez mini-generatory w jeden wzorzec przy-

List. 3.4. Fragment implementacji z listingu 5.16 - konkatenacja wzorców muzycznych w obrębie zdefiniowanych klas wzorców w komponowaniu bez definiowania formy utworu. Środowisko Matlab. Opracowanie własne.

```
1  for i=1:n
2  ...
3  if ( (i/n*100) <= 25 ) %klasa A
4      pattern = [bass; snare; closedhihat; low_mid_tom;
5          high_mid_tom; tambourine; harmonization; basspiano;
6          basspiano1; basspiano2 ];
7  end
8  if ( (i/n*100) > 25 && (i/n*100) <= 50 ) %klasa B
9          pattern = [bass; snare; closedhihat; low_mid_tom;
10         high_mid_tom; tambourine; basspiano2; basspiano1];
11 end
12 if ( (i/n*100) > 50 && (i/n*100) <= 75 ) %klasa C
13         pattern = [bass; snare; closedhihat; low_mid_tom;
14         high_mid_tom; tambourine; fill; basspiano;
15         basspiano1; basspiano2 ];
16 end
17 if ( (i/n*100) > 75 && (i/n*100) <= 100 ) %klasa D
18         pattern = [bass; closedhihat; tambourine; fill; basspiano ];
19 end
20 ...
21 end
```

należący do zdefiniowanej przez użytkownika klasy, w ramach reguły konkatenacji zdefiniowanej w (2.6) na stronie 55.

Przykład implementacji konkatenacji wzorca muzycznego oraz definiowania klas wzorców muzycznych przedstawiono na listingu 3.4. W przykładzie zdefiniowane zostały cztery klasy, a ich wypełnienie wzorcami muzycznymi zostało ujęte w warunkach logicznych opisanych w liniach 3, 8, 13 i 18. Autor w przedstawionej implementacji założył równomierne rozłożenie generowanych wzorców w ramach klas, badając zmienną i - iterator pętli generowania wzorców oraz zmienną n definiująca ogólną liczbę wygenerowanych wzorców. W prezentowanym przykładzie każda z generowanych klas zawiera taką samą liczbę wzorców muzycznych, a każdy z nich zdefiniowany jest jako macierz *pattern*, będąca wynikiem konkatenacji wzorców muzycznych generowanych przez różne mini-generatory opisane w poprzedniej części tego rozdziału. Składowe wzorców generowanych przez mini-generatory zostały dokładnie przedstawione w implementacji na listingu 5.16 w części "Dodatek A: Listingi kodów".

Definiowanie klas wzorców muzycznych ma charakter subiektywny, dlatego w implementacji 3.4 autor zaproponował następujące struktury klas:

— klasa A zbudowana jest ze wzorców muzycznych zbudowanych na podstawie wzorców: *bass*, *snare*, *closedhihat*, *low_mid_tom*, *high_mid_tom*, *tambourine*, *harmonization*, *basspiano*, *basspiano*1 i *basspiano*2,

List. 3.5. Przykład implementacji tworzenia macierzy podobieństwa wzorców muzycznych w komponowaniu bez definiowania formy utworu. Środowisko Matlab. Opracowanie własne.

```
1  ...
2  pcdistribution(i,:) = pcdist1(pattern);
3  ...
4  %macierz podobienstwa
5  pattern_differences = [];
6  for i=1:n
7          pattern_to = repmat(pcdistribution(i,:),n,1);
8          d =  sqrt(sum((pattern_to - pcdistribution).^2));
9          e = [sum(d,2) , [1:n]'];
10         pattern_differences{i} = sortrows(e);
11 end
12 ...
```

— klasa B zbudowana jest ze wzorców generowanych na podstawie wzorców: *bass*, *snare*, *closedhihat*, *low_mid_tom*, *high_mid_tom*, *tambourine*, *basspiano*1 i *basspiano*2,
— klasa C zbudowana jest ze wzorców powstałych na podstawie wzorców: *bass*, *snare*, *closedhihat*, *low_mid_tom*, *high_mid_tom*, *fill*, *basspiano*, *basspiano*1 i *basspiano*2,
— klasa D zawiera wzorce muzyczne wygenerowane na podstawie wzorców: *bass*, *closedhihat*, *tambourine*, *fill* i *basspiano*.

Warto w tym miejscu zaznaczyć, że w przypadku wykorzystanego w implementacji narzędzia *MIDI* Toolbox, podana kolejność konkatenacji wzorców nie ma znaczenia, ponieważ podczas przetwarzania wynikowej macierzy nut wygenerowanej kompozycji muzycznej, jej dane są analizowane i przetwarzane na wiadomości *MIDI* w kolejności ich wystąpienia w czasie (kolumna 1 i 6 macierzy nut) od początku pliku.

W następnej części tego rozdziału opisany został etap tworzenia kompozycji muzycznej na podstawie wygenerowanych wcześniej wzorców. Autor dokonał analizy kolejnych składowych tego procesu według implementacji z listingu 5.16.

3.1.2. Generowanie kompozycji muzycznej

Przed przystąpieniem do opisywania implementacji generowania końcowej sekwencji muzycznej, autor przedstawił przykładową implementację generowania macierzy podobieństwa wzorców muzycznych, która wykorzystywana jest do wyznaczania kolejnych wzorców występujących w kompozycji muzycznej.

Na listingu 3.5 przedstawiono przykład implementacji macierzy podobieństwa dla opisywanego przypadku generowania kompozycji według listingu 5.16. Podobieństwo pomiędzy wzorcami muzycznymi w przyjętej implementacji zostało zdefiniowane jako odległość Euklidesowa pomiędzy i-tym wzorcem a resztą wzorców ze zbioru wszystkich wygenerowanych wzorców w dziedzinie ważonej funkcji dystrybucji klas wysokości dźwięków z wykorzystaniem modelu Parncutt'a opisanego wyrażeniami: (2.14), (2.15), (2.16), (2.17) i (2.18) ujętymi na stronach 60 - 60. Wartości funkcji dystrybucji klas wysokości dźwięków są wyznaczane za pomocą wbudowanej w *MIDI*

List. 3.6. Przykład implementacji generowania kompozycji w komponowaniu bez definiowania formy utworu. Środowisko Matlab. Opracowanie własne.

```
1  sequence = [];
2  shift_value = basepattern(end,1) + basepattern(end,2);
3  for i=1:seq_patterns
4          if (i==1)
5                  state =  fix(rand(1)*n+1);
6                  temppattern = pattern_struct{state}.data;
7          end
8          if (i>1)
9                  choice = fix(rand(1)*0.40*n + 1);
10                 state = pattern_differences{state}(choice,2)
                       ;
11                 temppattern = pattern_struct{state}.data;
12                 temppattern = shift(temppattern, 'onset',
                       shift_value);
13                 shift_value = shift_value + basepattern(end
                       ,1) + basepattern(end,2);
14         end
15         sequence = [sequence; temppattern];
16 end
17 writemidi_java(sequence,'sequence.mid',120,100);
```

Toolbox funkcji *pcdist*1, która jako parametr wejściowy wymaga podania macierzy nut (w tym przypadku - macierzy *pattern*, reprezentującej wygenerowany wzorzec muzyczny - linia 2 na listingu 3.5). W macierzy *pcdistribution* przechowywane są wartości rozkładu uzyskanego z funkcji *pcdist*1 dla poszczególnych wzorców.

W pętli przedstawionej na listingu 3.5 obliczane są odległości kolejnego i-tego wzorca od wszystkich wygenerowanych wzorców, przy czym im mniejsza odległość tym większe podobieństwo danego wzorca w dziedzinie klas wysokości dźwięków. Odległości i-tego wzorca przechowywane są w strukturze *pattern_differences* posortowanej od najbardziej do najmniej podobnego wzorca. Struktura ta jest wykorzystywana w dalszej części omawianego etapu komponowania do wyboru występujących w generowanej kompozycji wzorców muzycznych.

Na listingu 3.6 przedstawiony został ostatni etap generowania kompozycji muzycznej na podstawie wygenerowanych wzorców. Głównym ograniczeniem w prezentowanej implementacji jest założenie stałej wartości przesunięcia w czasie wstawianych w wyjściową kompozycję wzorców. Wynika to z faktu przyjęcia koncepcji pustego wzorca o zdefiniowanej wcześniej strukturze, do którego wstawiane są dźwięki muzyczne w określonych lokalizacjach wzorca.

Wartość przesunięcia przechowywana jest w zmiennej *shift_value* zainicjowanej w linii 2 wartościami czasu pojawienia się i długości ostatniego dźwięku w bazowym wzorcu muzycznym. W przedstawionej na listingu 3.6 pętli generowana jest kompozycja muzyczna w postaci macierzy nut *sequence* inicjowanej jako pusta macierz, która następnie w sposób dynamiczny jest konkatenowana pionowo z wylosowanym wzorcem muzycznym.

W pierwszej iteracji losowany jest początkowy wzorzec muzyczny, zawarty w strukturze $pattern_struct\{state\}.data$, który jest następnie konkatenowany z początkowo pustą macierzą nut *sequence* (linia 15). W kolejnych iteracjach (linie 8 - 14) losowany jest identyfikator jednego ze wzorców wg. wyrażenia $choice = fix(rand(1) * 0.40 * n + 1)$, definiującego wybór wzorca ze zbioru 40 procent najbardziej podobnych wzorców muzycznych, których liczność wynosi n. Zmienna *choice* jest następnie wykorzystywana do wybrania wylosowanego wzorca ze struktury $pattern_differences$. Wybrany wzorzec muzyczny przechowywany jest w zmiennej *temppatern*. Przed złączeniem wzorców, wybrany wzorzec *temppattern* jest modyfikowany poprzez przesunięcie wartości rozpoczynających zawartych w nim dźwięków muzycznych o wartość $shift_value$, będącą offsetem dla kolejnych wybieranych wzorców (linia 12). Wartość $shift_value$ jest następnie zwiększana o przyjęty rozmiar czasowy wzorca (linia 13), a wzorzec *temppattern* jest konkatenowany z macierzą *sequence*. Finalna kompozycja zapisywana jest do pliku $MIDI$, który jest następnie przetwarzany w sposób opisany w rozdziale 3.3.

Przedstawiona implementacja systemu komponowania muzyki pomija możliwość definiowania opisu przyszłej struktury kompozycji. Takie uproszczenie prowadzi do ograniczeń w postaci konieczności definiowania dużej liczby klas wzorców oraz liczby powtórzeń procesu komponowania ze zmianą parametrów początkowych w celu dostosowania wynikowej kompozycji do kanonów kryterium akceptowalności utworu muzycznego w sensie artystycznym. W kolejnej części tego rozdziału autor wprowadził do koncepcji systemu algorytmicznego komponowania muzyki funkcjonalność definiowania formy utworu muzycznego, tworząc przy tym wyższy poziom abstrakcji opisu procesu komponowania muzyki w przyjętej koncepcji.

3.2. Komponowanie z uwzględnieniem opisu formy kompozycji

Autorski system komponowania muzyki na podstawie wzorców w podstawowej postaci zakładał wykorzystanie definiowanych wzorców muzycznych jako "budulca" w procesie generowania kompozycji wyjściowej. Za podstawowe ograniczenie "twórcze" w tej kwestii autor uznał niekontrolowany i często pseudolosowy rozkład złączanych wzorców w kompozycji wyjściowej, co przekładać się może na jej małą atrakcyjność muzyczną oraz niskie walory artystyczne. W założeniach muzyki tła, na której autor oparł mechanizm generowania muzyki, pojawia się akceptacja dla dużej liczby repetycji i modyfikacji tego samego wzorca oraz brak skończonej formy utworu, z uwagi na jego "niekończącą się" strukturę (ograniczeniem utworu jest liczba wykorzystanych wzorców, które nie tworzą zamkniętej kompozycji). Postanowiono jednak wprowadzić pewien opis abstrakcji formy kompozycji oraz mechanizmy generowania poszczególnych jej części (wzorców), tak, aby zwiększyć atrakcyjność generowanych utworów muzycznych oraz upodobnić ich w swej strukturze do dzieł skomponowanych przez człowieka wg przyjętych kanonów muzyki.

Przykładową implementację działającego systemu algorytmicznego komponowania muzyki przedstawiono na listingu 5.19 w rozdziale "Dodatek A. Listingi kodów". W dalszej części tego rozdziału omówione zostały parametry kontrolujące zmodyfikowany proces komponowania muzyki, mechanizm wykorzystania kadencji

muzycznych, definiowania poszczególnych części kompozycji a także zmieniony sposób generowania wzorców przez mini-generatory.

3.2.1. Parametry kontrolujące proces komponowania

Na listingu 5.19 w liniach 4 - 54 znajduje się przykładowa implementacja parametrów kontrolujących proces komponowania wraz z wykorzystaniem zmiennych pomocniczych stosowanych w dalszych fragmentach implementacji. Istotną zmianą w stosunku do poprzedniej implementacji systemu jest wprowadzenie parametrów opisujących strukturę wynikowej kompozycji. Do najważniejszych parametrów zalicza się:

— wektor opisujący strukturę kompozycji,
— tempo kompozycji (w jednostkach bpm),
— wektor opisujący skalę muzyczną, w której wygenerowana zostanie kompozycja,
— wektor (lub wektory) możliwych lokalizacji dźwięków we wzorcach muzycznych,
— wektor (lub wektory) możliwych przesunięć dźwięków w czasie,
— wektor (lub wektory) możliwych wartości długości dźwięków,
— zmienne opisujące liczbę oraz czas trwania akordów, wykorzystanych w danej części kompozycji.

W implementacji z listingu 5.19 zdefiniowano 4-ry różne części kompozycji różniące się strukturą muzyczną przechowywaną w zmiennych $part1$, $part2$, $part3$ i $part4$ i traktowane jak wzorce muzyczne przedstawione w koncepcji komponowania w rozdziale 3.1. Zdefiniowane zmienne posiadają swoje identyfikatory w postaci liczbowej ($part1$ - 1, $part2$ - 2, $part3$ - 3 i $part4$ - 4), aby w wygodny sposób wykorzystywać je do opisu struktury generowanej kompozycji. Wektor opisujący strukturę przechowywany jest w zmiennej $form$, w postaci szeregu identyfikatorów zmiennych $part$. Na listingu 5.19 $form$ przyjmuje postać [1,1,3,3,2,2,1,1,4,4,3,3,2,2,1,1,4,4,2,2,1,1], definiującą strukturę kompozycji, w której następują po sobie kolejno wymienione części $part1$, $part1$, $part3$, $part3$, ..., $part1$.

Skala muzyczna wykorzystywana w kompozycji opisana jest w postaci wektora zawierającego wartości różnic kolejnych stopni skali wyrażonych w półtonach. Pierwszy element wektora posiada wartość 0, ponieważ jest to pierwszy stopień skali, dla którego zostaną obliczone kolejne według wartości elementów wektora skali. Na listingu 5.19 wektor skali *skala* przyjmuje postać skali molowej naturalnej zdefiniowanej w postaci [0,2,3,5,7,8,10,12]. Dla prezentowanego przykładu dźwiękiem bazowym wykorzystanym do utworzenia skali jest dźwięk o wartości 60 wg standardu $MIDI$ (dźwięk $C4$). Gotowe wartości dźwięków skali zostały utworzone za pomocą wbudowanej we framework $MIDI$ Toolbox funkcji createnmat, która przyjmuje jako parametr wejściowy wektor dźwięków skali wykorzystanych do utworzenia wynikowej macierzy nut. Autor wykorzystał strukturę macierzy nut w celu usprawnienia opisanego w dalszej części procesu generowania poszczególnych części kompozycji. Wycinek macierzy nut utworzonej na podstawie skali został użyty go wygenerowania bazowego akordu (zmienna *chord*) wykorzystanego podczas generowania progresji akordów.

Kolejnymi istotnymi z punktu widzenia procesu komponowania parametrami są wektory definiujące możliwe lokalizacje ($possible_location$, $possible_location_snare$), długości ($possible_duration$, $possible_duration_mel1$, $possible_duration_mel2$) oraz przesunięcia ($possible_offset$, $possible_offset_snare$, $offset_mel$, $offset_mel2$) dźwięków muzycznych

występujących w generowanej kompozycji. Wektory te mogą być definiowane oddzielnie dla każdego instrumentu wykorzystanego w kompozycji, np. *possible_location_snare* jest użyty do generowania ścieżki instrumentalnej dla werbla (z ang. *snare*). Mogą być również definiowane globalnie dla wszystkich dźwięków (*possible_duration*) bądź dźwięków należących do określonej grupy, np. melodii (*possible_duration_mel*1). Wykorzystanie poszczególnych wektorów zostało opisane w kolejnych częściach tego rozdziału.

Proces generowania kompozycji zamknięty jest w głównej pętli aplikacji (linie 55 - 231 na listingu 5.4), w której proces tworzenia poszczególnych części kompozycji podzielono na:
— generowanie akordów,
— generowanie melodii,
— generowanie linii basowej,
— generowanie linii perkusyjnej.
Koncepcyjnie pierwszym etapem tworzenia wzorców muzycznych kompozycji jest generowanie progresji akordów, stanowiące podstawę do dalszego budowania struktury kompozycji. Następny w kolejności proces generowania melodii został połączony z procesem generowania linii basowej i opisany we wspólnym rozdziale tej pracy. Autor przedstawił również rozszerzenie pojęcia mini-generatora w rozdziale poświęconym generowaniu linii perkusyjnej wzorca.

3.2.2. Generowanie progresji akordów

Podstawowym krokiem w autorskiej koncepcji generowania kompozycji bazującej na abstrakcyjnym opisie formy jest zdefiniowanie przestrzeni możliwych dźwięków wykorzystywanych podczas tworzenia wzorca muzycznego. Autor w tym względzie odwrócił nieco twórczy proces powstawania utworu, zaczynając od wygenerowania postępujących po sobie akordów, stanowiących podstawowy zbiór dźwięków wykorzystanych do generowania melodii.

Mechanizm generowania progresji akordów został zaimplementowany w postaci funkcji *progresje* przedstawionej na listingu 5.20. Funkcja przyjmuje następujące parametry:
— *chord* - jest to bazowy akord (w postaci macierzy nut) wykorzystywany do generowania kolejnych akordów wzorca muzycznego,
— *steps* - określa liczbę akordów, które zostaną wygenerowane,
— $offset$ - to wartość przesunięcia pomiędzy kolejnymi akordami wzorca muzycznego,
— $shift_value$ - aktualna wartość położenia w czasie znacznika definiującego początek kolejnego akordu,
— $chord_l$ - czas trwania akordu,
— *skala* - wektor zdefiniowanej skali.
Główna pętla funkcji *progresje* (linia 6 na listingu 5.20) iteruje według zdefiniowanej liczbie akordów, które mają zostać wygenerowane w wyjściowym wzorcu zwracanym przez funkcję w zmiennej *out*. W zależności od aktualnej wartości iteratora k pętli, wykorzystywane są odmienne reguły tworzenia akordu związane z daną kadencją (tab. 2.6 na stronie 69). Dodatkowo w niektórych regułach brany jest pod uwagę pierwszy element kadencji oraz element $(k-1)$. Podczas generowania progresji za każdym obrotem głównej pętli tworzona jest tymczasowa macierz nut zawierająca bazowy akord *chord* przesunięty w czasie o wartość $shift_value$ za pomocą wbu-

dowanej funkcji *shift* (linia 7 na listingu 5.20). Podczas generowania progresji wykorzystywana jest pomocniczy wektor *prog*, który przechowuje aktualnie wybrany stopień skali.

Dla przykładu w podanej na listingu 5.20 implementacji dla $k = 1$ losowo wybierana jest jedna z reguł:

— z prawdopodobieństwem 0.2 aktualnie generowany akord zostanie przetransponowany w górę o liczbę półtonów odpowiadającą IV stopniu wybranej skali;
— z prawdopodobieństwem 0.2 aktualnie generowany akord zostanie przetransponowany w górę o liczbę półtonów odpowiadającą V stopniu wybranej skali;
— z prawdopodobieństwem 0.6 aktualnie generowany akord pozostanie bez zmian (przyjmie wartości tymczasowej macierzy *chord*).

Odmienne reguły są stosowane dla $k = 2$, $k = 3$, $k = 4$ oraz $k > 4$. Przedstawiony mechanizm generowania progresji akordów został zbudowany bazując na kilku najpopularniejszych rodzajach kadencji przedstawionych w tab. 2.6 na stronie 69 i może zostać rozbudowany o kolejne reguły tworzenia nowych sekwencji.

Po każdym obrocie głównej pętli funkcji *progresje* następuje modyfikacja wartości *shift_value* wg wzoru przedstawionego w linii 91 na listingu 5.20 oraz złączenie aktualnie generowanego akordu *chord* ze zwracaną macierzą nut *out* zawierającą progresję akordów. Funkcja *progresje* zwraca również ostatnią zmodyfikowaną wartość *shift_value*, która jest następnie wykorzystywana do oznaczenia aktualnej pozycji czasowej w generowanej kompozycji.

Autor w przykładowej implementacji na listingu 5.19 generował progresje składające się z 4 akordów, każdy o długości 4 bpm (parametry $lp1$, $lp2$, $lp3$ i $lp4$ podobnie jak parametry $step1$, $step2$, $step3$ i $step4$ przyjmują wartość 4).Wszystkie akordy znajdujące się w utworzonej progresji są skalowane o wartość parametru *steps* za pomocą wbudowanej w *MIDI* Toolbox funkcji *setvalues* (np. linia 60 na listingu 5.19). Zabieg ten ma na celu zdefiniowanie przestrzeni czasowej dla dźwięków muzycznych tworzonych podczas procesu generowania melodii i linii basowej. Dodatkowo z prawdopodobieństwem 0.3 dokonywany jest tzw. *przewrót* pierwszego akordu z progresji za pomocą transponowania dźwięku podstawowego o oktawę w górę (np. linia 66 na listingu 5.19). Kolejnym ważnym elementem procesu komponowania muzyki jest utworzenie melodii bazującej na przygotowanej wcześnie progresji akordów.

3.2.3. Generowanie melodii i linii basowej

Na podstawie struktury *chordprogression* (listing 5.19) przygotowywane są linia melodyczna i basowa, które są następnie dodawane do wzorca muzycznego stanowiącego odrębną część kompozycji (macierz $part1$, $part2$, $part3$ lub $part4$). Za ten mechanizm odpowiada utworzona przez autora funkcja *makemelody* zaimplementowana w postaci przedstawionej na listingu 5.6. Funkcja przyjmuje następujące parametry:

— *chordprogression* - macierz nut zawierająca progresję akordów,
— *ch* - numer kanału *MIDI* przypisany do generowanej melodii,
— *possible_duration* - wektor możliwych długość dźwięków melodii,
— *offset* - wektor możliwych przesunięć w czasie dźwięków melodii,
— *makebass* - flaga kontrolująca generowanie linii basowej bazujące na linii melodycznej.

W głównej pętli funkcji *makemelody* (linie 9 - 44, listing 5.20) przebiega proces podziału progresji akordów na okna czasowe (zmienna *window*), utworzone za pomocą

wbudowanej w *MIDI* Toolbox funkcji *onsetwindow* pobierającej trzy parametry wejściowe:
— macierz nut, z której pobrany ma być fragment w postaci okna (w prezentowanym przykładzie - macierz *chordprogression*),
— minimalna wartość jednostki czasu dla okna (początek, wartość wyrażona domyślnie w bpm),
— maksymalna wartość jednostki czasu dla okna (koniec okna).
W implementacji z listingu 5.21 autor przyjął wartość rozpiętości okna 1 bpm, co powoduje analizowanie macierzy *chordprogression* ze skokiem co 1 bpm. Tak więc początkowe wywołanie funkcji *onsetwindow* przybiera postać $onsetwindow(chordprogression, 0, 1)$.

Dla aktualnie wybranego okna *window* tworzony jest wektor *pitch* możliwych do wykorzystania w melodii dźwięków, z którego w sposób losowy wybierana jest wartość wysokości dźwięku (zmienna *selectedpitch*). Liczba dźwięków generowanych w przedziale czasowym okna nie jest z góry znana, a warunkuje ją warunek opisany w linii 17 na listingu 5.21. Dla tworzonego dźwięku wybierana jest z wektora *possible_duration* wartość długości *duration* oraz wybierana jest wartość opcjonalnego przesunięcia w postaci zmiennej $seloffset$. W konsekwencji powstaje struktura *note* reprezentująca pojedynczy dźwięk melodii w postaci macierzy nut. Dodatkowo w celu urozmaicenia dynamiki dla niektórych generowanych dźwięków melodii losowana jest wartość dynamiki z przedziału <110,127> (linia 31 na listingu 5.21).

Utworzona macierz *note* jest następnie konkatenowana z macierzą $melfragment$, przechowującą fragment melodii wygenerowany w ramach okna czasowego *window*. Jeżeli ustawiona jest flaga *makebass* dodatkowo generowana jest linia basowa z wykorzystaniem wbudowanych w *MIDI* Toolbox funkcji $shift$ i *quantize*.

Funkcja $shift$ przesuwa elementy macierzy nut o wybraną wartość w danym wymiarze. Przyjmuje trzy parametry wejściowe:
— macierz nut dla której ma być przeprowadzone przesunięcie,
— wymiar,
— wartość przesunięcia,
— opcjonalnie reprezentacja czasowa elementów.
Wymiar może przyjąć jedną ze zdefiniowanych wartości: *onset* - położenie dźwięków w czasie, *dur* - długość dźwięków, *pitch* - wysokość dźwięków, *chan* - kanał *MIDI* oraz *vel* - głośność. Opcjonalnie definiowana reprezentacja czasowa może przyjmować wartość *beat* (długość dźwięków wyrażona w bpm) lub *sec* - długość dźwięków w sekundach. Ten dodatkowy parametr może być wykorzystany do przesuwania dźwięków macierzy nut w dziedzinie położenia (*onset*) i długości (*dur*). Funkcja $shift$ zwraca macierz nut przetworzoną wg przyjętych wartości parametrów wejściowych. Na listingu 5.21 w linii 40 wywołanie funkcji $shift$ przyjmuje postać:

```
bassline = [bassline;
shift(melfragment([1:fix(rand(1)*8+1):end],:),'pitch',-24)];
```

Macierz *bassline* zawiera generowaną linię basową, która jest iteracyjnie zwiększana o macierz nut utworzoną za pomocą funkcji $shift$ z wybranych losowo dźwięków macierzy $melfragment$, transponowanych w dół o dwie oktawy (24 półtony).

Kolejną wykorzystaną w generowaniu linii basowej funkcją jest *quantize*, która kwantyzuje początki i długości dźwięków w podanej macierzy nut oraz opcjonalnie

filtruje z niej dźwięki krótsze niż podany próg wartości. Funkcja pobiera kolejno maksymalnie cztery parametry:
— wejściowa macierz nut ulegająca przetworzeniu,
— wartość progowa czasów rozpoczęcia dźwięków w procesie kwantyzacji (domyślnie: 1/8),
— (opcjonalnie) wartość progowa długości dźwięków w procesie kwantyzacji (domyślnie: dwu krotność progu wartości początkowych),
— (opcjonalnie) wartość progowa dźwięków, które mają zostać odfiltrowane z wejściowej macierzy (według wartości długości dźwięku).

Podczas kwantyzacji macierzy nut, wszystkie jej dźwięki zostają przesunięte do zaokrąglonej wartości czasu rozpoczęcia (wielokrotności wartości progowej czasu rozpoczęcia) a ich długość skwantyzowana zgodnie z parametrem wartości progowej długości dźwięków. Na listingu 5.21 w linii 46 funkcja *quantize* wywołana jest w następujący sposób:

```
bassline = quantize(bassline,1/4,1/16);
```

W konsekwencji macierz *bassline* zawiera przetworzoną linię basową, w której wszystkie dźwięki zaczynają się w chwili będącej wielokrotnością wartości $\frac{1}{4}$ bpm a ich długość jest skwantyzowana do wielokrotności $\frac{1}{16}$ bpm. Gotowa linia melodyczna z opcjonalnie utworzoną linią basową zwracana jest w postaci macierzy *melody*.

Na tym etapie generowany wzorzec muzyczny posiada już progresję akordów oraz pasującą do niej linię melodyczną. Ostatnim krokiem w procesie algorytmicznego komponowania muzyki za pomocą przyjętej koncepcji jest wykorzystanie mini-generatorów do budowy linii instrumentów perkusyjnych.

3.2.4. Generowanie linii instrumentów perkusyjnych

Wzorce muzyczne stanowiące odrębne części kompozycji powstały na bazie wzorców generowanych przez wcześniej opisane funkcje *progresje* i *makemelody*. Posiadają osadzoną rytmicznie strukturę melodii oraz linii basowej. Aby jednak rozszerzyć strukturę muzyczną kompozycji o dodatkowy aspekt związany z rytmiką i instrumentami perkusyjnymi, autor zastosował mechanizm opisywanych wcześniej mini-generatorów, w nieco odmiennej formie.

W zmodyfikowanej koncepcji systemu algorytmicznego komponowania muzyki mini-generatory zostały przekształcone do postaci funkcji. Na listingu 5.22 w rozdziale "Dodatek A. Listingi kodów" przedstawiona została autorska implementacja mini-generatora w środowisku Matlab. Funkcja *minigenerator* przyjmuje następujące parametry:
— K - nazwa klasy, do której przynależy mini-generator,
— MM - motyw muzyczny na bazie którego generowany jest wzorzec muzyczny W,
— cc - parametr definiujący krok wypełnienia dźwiękami wzorca muzycznego W,
— $offset$ - parametr wprowadzający przesunięcie wypełnienia wzorca muzycznego W,
— res - długość dźwięku (w sekundach) stała dla wszystkich dźwięków wzorca W,
— l - docelowa długość generowanego wzorca W,
— $tempo$ - tempo w jakim mają być odtwarzane dźwięki wzorca W,
— ch - numer kanału *MIDI* wykorzystanego przez wzorzec W.

Parametr K przyjmuje następujące wartości: C, D oraz M. Wartość C oznacza mini-generator polifoniczny, natomiast D generator perkusyjny. Dla wartości M funkcja przyjmuje mechanizm mini-generatora monofonicznego.

Parametr MM jest wektorem opisującym motyw muzyczny w postaci kolejnych wartości wysokości dźwięków. Wektor MM może przybrać również postać skalara - w takim przypadku wszystkie występujące we wzorcu W dźwięki muzyczne posiadają taką samą wartość wysokości równą wartości parametru MM. Mechanizm selekcji motywu muzycznego został przedstawiony w dalszej części tego rozdziału.

Krok wypełnienia cc określa poziom upakowania dźwięków muzycznych we wzorcu W w zależności od jego docelowej długości l. Kiedy cc przyjmie wartość 1, poziom upakowania dźwięków będzie największy, tzn. w generowanym wzorcu W dźwięki muzyczne zostaną rozlokowane we wszystkich możliwych lokalizacjach macierzy nut (nie będzie pustych wierszy, a więc pauz pomiędzy dźwiękami). Wartość cc musi być mniejsza niż docelowa długość wzorca.

Dodatkowo autor wprowadził w implementacji możliwość wypełniania wzorca z zadanym przesunięciem (parametrem $offset$). Dzięki temu możliwe jest generowanie wzorca, w którym początkowa jego część pozostanie pusta. W takim przypadku podczas odtwarzania kompozycji wzorce wygenerowane z zadaną niezerową wartością parametru $offset$ będą charakteryzować się początkową pauzą.

Parametr res jest wartością długości dla dźwięków muzycznych wchodzących w skład generowanego wzorca W. Wartość długości dźwięku res jest wyrażona w jednostkach bpm (a więc względnych w stosunku do wartości tempa utworu). W takiej samej jednostce wyrażone są parametry docelowej długości wzorca (l) oraz tempa wzorca ($tempo$). Ostatnim parametrem przyjmowanym przez funkcję *minigenerator* jest numer kanału *MIDI*, przyjmujący wartości od 1 do 16 zgodnie ze standardem *MIDI*.

Wykorzystana w implementacji z listingu 5.22 funkcja *createnmat*, pochodząca z *MIDI* Toolboxa, do budowy macierzy nut potrzebuje parametru długości dźwięku wyrażonej w jednostkach bezwzględnych - sekundach. Aby wykorzystać dostępną funkcję *createnmat*, w linii nr 4 kodu funkcji *minigenerator* z listingu 5.22 dokonuje się konwersji wartości res z jednostki bpm do sekund wg wzoru

$$dur_{sek} = \frac{dur_{bpm} * 60}{tempo}, \tag{3.4}$$

gdzie:

— dur_{sek} to szukana wartość długości dźwięku wyrażona w sekundach,

— dur_{bpm} to dana wartość długości dźwięku wyrażona w jednostce bpm (parametr res),

— $tempo$ to dane tempo utworu w jednostce bpm.

Proces generowania wzorca W przez mini-generator został przedstawiony w algorytmie 3.1. W początkowym etapie generowania wzorca W tworzona jest macierz nut, zawierająca zerowe elementy w kolumnie opisującej wysokości poszczególnych dźwięków muzycznych. Kolejnym ważnym krokiem jest wybór dźwięków muzycznych z macierzy nut, do których przypisane zostaną odpowiednie wysokości dźwięków z motywu MM. Jeżeli motyw jest skalarem, to jego wartość zostanie przypisana wszystkim wybranym dźwiękom macierzy nut. Jeśli natomiast motyw tworzy wektor elementów, jego wartości będą kolejno przypisywane do wybranych pozycji wyselekcjonowanych dźwięków macierzy nut. W przypadku gdy liczba wybranych dźwięków

Algorytm 3.1 Generowanie wzorca W za pomocą funkcji *minigenerator*. Opracowanie własne.

1. Wczytaj parametry wejściowe:K, MM, cc, $offset$, res, l, $tempo$, ch,
2. Wykonuj następujące kroki gdy: $(K = C \vee K = D \vee K = M) \wedge res > 0 \wedge l > 0$:
   a) Dokonaj konwersji wartości res do postaci sekund,
   b) Utwórz macierz nut MN zawierającą dźwięki muzyczne w liczbie $fix(\frac{l}{res})$ o długościach równych wartości res i wysokościach równych wartości 0,
   c) Jeżeli $K = D$, ustaw dla każdego dźwięku macierzy MN wartość numeru kanału na 10, w przeciwnym wypadku na wartość parametru ch,
   d) Wybierz pozycje dźwięków muzycznych, dla których zostaną przypisane wartości wysokości z parametru MM, tworząc wektor v_1 według schematu: $[1 + offset : cc : size(MN, 1)]$,
   e) Jeżeli długość motywu MM jest równa 1 (skalar), wykonuj następujące kroki, w przeciwnym wypadku przejdź do punktu 2f:
      i. Przypisz na wybranych w punkcie 2d pozycjach macierzy nut, wartość MM,
      ii. Jeżeli $K = C$, zbuduj akordy bazujące na dźwięku MM,
      iii. Zakończ działanie funkcji
   f) Utwórz wektor v_2 o długości wektora v_1 zawierający zerowe elementy,
   g) Wypełnij wektor v_2 kolejno wartościami z parametru MM, zaczynając od pierwszego elementu MM, w przypadku, gdy długość v_2 jest mniejsza niż MM i w trakcie wypełniania osiągnięty zostanie ostatni element v_2,
   h) Wstaw kolejne elementy wektora v_2 w wybrane w punkcie 2d pozycje macierzy nut MN,
   i) Jeżeli $K = C$:
      i. Oszacuj tonację macierzy nut MN,
      ii. Na podstawie tonacji zbuduj akordy minorowe lub majorowe na bazie dźwięków MN,
      iii. Zakończ działanie funkcji
   j) Zakończ działanie funkcji

jest większa niż długość motywu MM, elementy motywu MM będą przypisywane do kolejnych pozycji, do momentu osiągnięcia ostatniego elementu motywu MM. Następnie pozycja wybierająca elementy z motywu MM zostanie ustawiona na początek motywu, a proces wypełniania wybranych pozycji macierzy nut jest kontynuowany od bieżącego elementu.

Dodatkowego komentarza wymaga wywołanie wbudowanych w *MIDI* Toolbox funkcji *createnmat* oraz *kkkey*. Omówienie wywołania funkcji *createnmat* zostanie przedstawione na przykładzie jej wywołania w linii nr 6 na listingu 5.22:

```
notematrix = createnmat(zeros(1,fix(l/res)),dur,127,10);
```

Funkcja pobiera cztery parametry: wektor wysokości dźwięków, długość dźwięków w sekundach, głośność dźwięków oraz numer kanału *MIDI* wykorzystywanego przez dźwięki macierzy nut. Na podstawie tych parametrów po wywołaniu funkcji *createnmat* otrzymujemy macierz nut. Poniższe wywołanie funkcji *createnmat* spowoduje utworzenie macierzy nut, zawierającą $fix(\frac{l}{res})$ wierszy - dźwięków muzycznych, o długości *dur*, głośności 127, ulokowanych w kanale *MIDI*

o numerze 10. Dodatkowo w wygenerowanej macierzy nut, w kolumnie przechowującej wysokości dźwięków, znajdować się będą elementy zerowe (elementy wektora $zeros(1, fix(l/res)))$.

Kolejną wbudowaną w $MIDI$ Toolbox funkcją wykorzystaną w mini-generatorze jest *kkkey*, za pomocą której szacowana jest tonacja macierzy nut MN, wykorzystywana jedynie podczas generowania akordu. Estymacja odbywa się na podstawie tzw. profili durowych i molowych zaproponowanych przez Krumhansl i Kesslera w [31], a dokładny jej opis znajduje się w rozdziale 2.2.1 w części poświęconej regułom komponowania. W ramach funkcji mini-generatora dokonuje się jedynie rozróżnienia na tonacje minorowe (molowe) i majorowe (durowe), co opisuje warunek z listingu 5.22 w linii 37. Jeżeli wartość otrzymana z funkcji *kkkey* jest mniejsza niż 13, to estymowana tonacja macierzy MN jest majorowa, w przeciwnym wypadku - minorowa. To w konsekwencji pociąga za sobą odmienne formuły generowania akordu, które nie są jednak wykorzystane w koncepcji komponowania muzyki z uwzględnieniem opisu formy kompozycji.

Mini-generatory w opisywanej koncepcji zostały wykorzystane do tworzenia partii muzycznych dla odrębnych instrumentów perkusyjnych. Przykład wywołania funkcji *minigenerator* przedstawiono na listingu 5.19 w linii 70:

```
bass = minigenerator('D',36,possible_location(floor(rand(1)*
    lpos+1)),possible_offset(floor(rand(1)*lof+1)),0.25,
    l_part1,tempo,10);
```

Funkcja mini-generatora została wywołana w celu utworzenia macierzy nut dla instrumentu perkusyjnego (klasa $'D'$ mini-generatora) o numerze 36 (wg standardu General $MIDI$ jest to *basskick* czyli kocioł centralny perkusji) według przyjętych wartości:

— możliwych pozycji dźwięków (*possible_location*),
— możliwych wartości przesunięć dźwięków (*possible_offset*),
— długości dźwięków równych 0.25 bpm,
— długości generowanego wzorca muzycznego (*l_part*1),
— wartości tempa (*tempo*),
— numerze kanału $MIDI$, w którym ma być wygenerowana macierz nut (10 - kanał zarezerwowany dla instrumentów perkusyjnych).

Funkcja *minigenerator* zwróci w tym wypadku macierz nut przypisaną do zmiennej *bass*, wykorzystanej w dalszej części procesu komponowania. Ostatnim krokiem tego procesu jest połączenie wszystkich wygenerowanych składowych struktury muzycznej w jedną macierz nut związaną z daną częścią kompozycji (wzorcem muzycznym) oraz zapis struktury kompozycji do postaci pliku $MIDI$.

3.2.5. Finalizacja procesu komponowania

Wygenerowane macierze nut progresji akordów, linii melodycznej, linii basowej oraz partii instrumentów perkusyjnych są finalnie konkatenowane w jeden wzorzec muzyczny przypisany do konkretnej części kompozycji. Przykład konkatenacji został omówiony na podstawie linii 86 z listingu 5.19:

```
part1 = [melody;bass;snare;clap;hihat;shift(chordprogression
    ,'pitch',-12)];
```

W przytoczonym fragmencie kodu budowany jest wzorzec muzyczny dla części *part*1. Składa się on z:
— macierzy nut *melody* zawierającej linię melodyczną i linię basową,
— macierzy nut *bass*, *snare*, *clap*, *hihat* wygenerowanych za pomocą mini-generatorów,
— macierzy nut *chordprogression* zawierającej progresję akordów.
Należy zauważyć, że autor dokonał transpozycji macierzy progresji akordów w wyniku czego melodia znajduje się na wysokości pierwotnej oktawy wybranej skali, progresja akordów umiejscowiona jest o jedną oktawę niżej, natomiast najniżej, bo o 2 oktawy, ulokowana jest linia basowa. Podobnie generowane są wzorce muzyczne dla pozostałych części kompozycji, przy czym różnią się macierzami nut wchodzącymi w skład ich struktury.

Gotowy wzorzec muzyczny (w opisywanym przypadku - *part*1) jest następnie konkatenowany z macierzą nut *composition*, zawierającą wyjściową kompozycję. Warto podkreślić, że wzorzec muzyczny jest generowany raz, a następnie wstawiany w miejscach wskazanych w zmiennej $form$ opisującej kolejność występowania poszczególnych części kompozycji. Za prawidłową lokalizację wzorca odpowiedzialna jest zmienna $shift_value$ stanowiąca znacznik końca aktualnie wygenerowanej części kompozycji.

Wygenerowana kompozycja muzyczna jest następnie konwertowana do postaci pliku $MIDI$ za pomocą funkcji *writemidi_java*, przyjmującej następujące parametry:
— macierz nut zawierającą macierz nut przeznaczoną do zapisania,
— nazwę wynikowego pliku $MIDI$,
— wartość tpq (z ang. *ticksperquarternote*) odzwierciedlająca najmniejszą wartość jednostki czasu, wykorzystaną do zapisu i odtworzenia dźwięku $MIDI$,
— tempo wyrażone w jednostkach bpm,
— licznik metrum,
— mianownik metrum,
— macierz opisująca przypisanie instrumentów $MIDI$ do poszczególnych kanałów $MIDI$.
Implementacja z listingu 5.19 w liniach 232 - 236 przedstawia przykład wykorzystania funkcji *writemidi_java*:

```
drumsp =
    [1,2,3,4,9,10,11,12,13,15,16,17,18,25,26,27,28,29,30,
31,32,33,34,41,49,51,57,128];
ldrumsp = length(drumsp);
nm = [1,floor(0+rand(1)*10);2,floor(10+rand(1)*10);3,floor
    (20+rand(1)*10);4,floor(30+rand(1)*10);5,floor(40+rand(1)
    *10);6,floor(50+rand(1)*10);7,floor(60+rand(1)*10);8,
    floor(70+rand(1)*10);9,floor(80+rand(1)*30);10,drumsp(
    floor(rand(1)*ldrumsp+1))];
writemidi_java(composition,'progression_19.mid',120,tempo
    ,4,4,nm);
```

W podstawowej wersji koncepcji komponowania autor zakładał przydział jednego wybranego instrumentu (był to domyślny instrument *Piano*) do wszystkich zdefiniowanych w kompozycji kanałów. Takie podejście spłaszcza fakturę kompozycji do

postaci mono-instrumentalnej z możliwością manualnego doboru instrumentów podczas etapu syntezy kompozycji. Autor wprowadził zatem losowy dobór instrumentów definiując pewne ograniczenia związane z wyborem instrumentu dla określonego kanału $MIDI$, jednocześnie wprowadzając uproszczoną automatyzację wyboru.

Macierz nm z listingu 5.19. opisuje przypisanie instrumentów do poszczególnych kanałów $MIDI$. W pierwszej kolumnie znajdują się numery kanałów $MIDI$, natomiast w drugiej - losowo wygenerowany numer instrumentu $MIDI$ należącego do danej grupy instrumentów. Autor zdefiniował następujące grupy instrumentów związane z określonym kanałem $MIDI$:
— dla kanału 1 - numery od 0 do 9,
— dla kanału 2 - numery od 10 do 19,
— dla kanału 3 - numery od 20 do 29,
— dla kanału 4 - numery od 30 do 39,
— dla kanału 5 - numery od 40 do 49,
— dla kanału 6 - numery od 50 do 59,
— dla kanału 7 - numery od 60 do 69,
— dla kanału 8 - numery od 70 do 79,
— dla kanału 9 - numery od 80 do 109,
— dla kanału 10 - zestaw perkusji losowany z wektora <1,2,3,4,9,10,11,12,13,15,16,17,18,25,26,27,28,29,30,31,32,33,34,41,49,51,57,128>
Każda nowa wygenerowana kompozycja otrzyma nowy losowy zestaw instrumentów. Autor celowo wprowadził tę funkcjonalność, aby wprowadzić różnorodność w ramach kompozycji generowanych przy tych samych ustawieniach systemu. Struktura macierzy nut kompozycji jest konwertowana do postaci pliku $MIDI$ z ustawieniami: tpq = 120 bpm, metrum = $\frac{4}{4}$ oraz tempo = 120 bpm.

W tym miejscu proces komponowania struktury muzycznej jest zakończony, a gotowy utwór może być odsłuchany w dowolnym odtwarzaczu plików $MIDI$. Aby zapewnić wysoką jakość wykonania muzyki, autor wprowadził do koncepcji systemu dodatkowy moduł odpowiedzialny za syntezę kompozycji do postaci cyfrowego audio, np. w formacie MP3. Narzędzia wykorzystane w tym celu zostały opisane w kolejnej części tego rozdziału.

3.3. Synteza wygenerowanej kompozycji

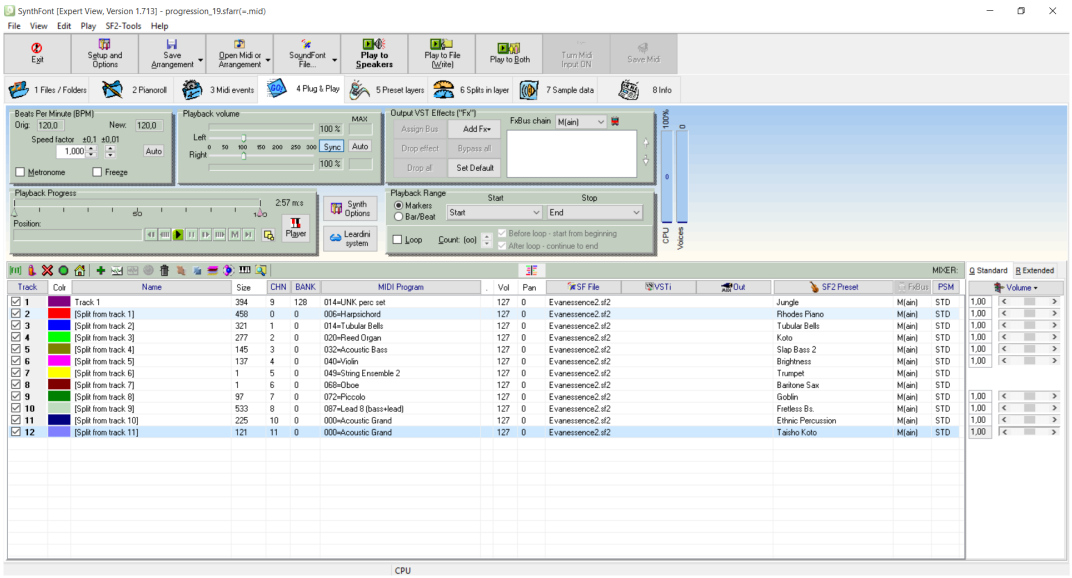
Niezależnie od przyjętej koncepcji komponowania, wynikowa kompozycja muzyczna jest zapisywana do postaci pliku $MIDI$. Jakość muzyki zapisanej w tym formacie zależy bezpośrednio od syntezatora wykorzystanego do jej odtworzenia. Aby zapewnić ujednoliconą jakość generowanych kompozycji oraz możliwość manualnej zmiany obsadzonych do poszczególnych ścieżek kompozycji instrumentów $MIDI$, autor wprowadził dodatkowy etap koncepcji komponowania jakim jest synteza kompozycji. Do tego etapu wykorzystane zostało zewnętrzne narzędzie jakim jest syntezator programowy SynthFont, którego przykładowe zrzuty ekranu przedstawiono na rys. 3.1 i 3.2 .

Aplikacja SynthFont pozwala syntezować muzykę zapisaną w formacie plików $MIDI$ do postaci cyfrowego audio z wykorzystaniem banków brzmień zawierających gotowe próbki instrumentów zapisane w formacie $SF2$. Na rys. 3.1 przedstawiono główny ekran aplikacji, w którego dolnej części znajduje się widok panelu ustawień ścieżek instrumentalnych wczytanej kompozycji. Do każdej ze ścieżek może zostać

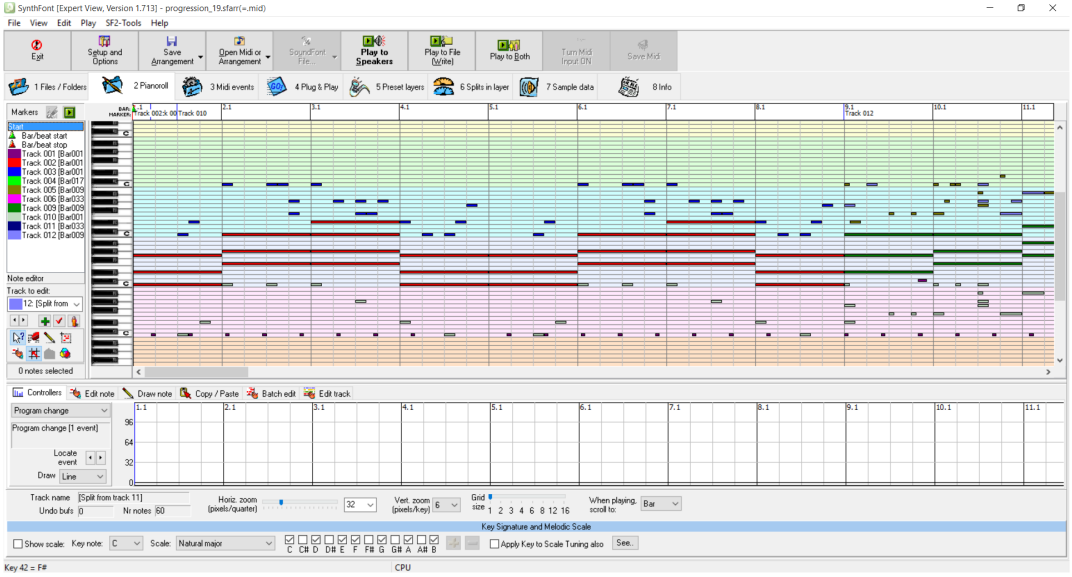
przypisany odrębny instrument z dowolnie wybranego banku brzmień $SF2$. Autor do syntezy wykorzystał darmowy i dostępny w Internecie zestaw brzmień $Evanssence2$ w wersji 2.1. Wynik syntezy może być odsłuchany w czasie rzeczywistym na wyjściu audio (opcja *Play to Speakers*), zapisany do pliku w formacie dźwięku cyfrowego (*Play to File(Write)*) lub osłuchany i zapisywany w tym samym czasie (*Play to Both*). Na dołączonej do pracy płycie CD znajdują się kompozycje zapisane w formacie $MP3$ i wygenerowane za pomocą dwóch opisywanych w rozdziale 2 koncepcji. Autor w uzyskanych kompozycjach dokonał manualnego doboru instrumentów według subiektywnych upodobań artystycznych, natomiast forma kompozycji pozostała w stanie nienaruszonym (pierwotnie wygenerowanym przez system).

Postać wizualna kompozycji jest w aplikacji SynthFont prezentowana w formie wykresu pianolowego, którego przykład przedstawiono na rys. 3.2. Każda ścieżka instrumentalna posiada własny kolor a domyślnym podziałem czasowym jest skala wyrażona w bpm. Dla ułatwienia orientacji w dziedzinie wysokości dźwięków, z lewej strony ekranu umieszczona została klawiatura z klawiszami pianina. Na rys. 3.2 pokazany został początkowy fragment kompozycji wygenerowanej na podstawie implementacji z listingu 5.4 systemu algorytmicznego komponowania muzyki z uwzględnieniem opisu formy.

Synteza komponowania jest ostatnim etapem komponowania w przyjętej koncepcji algorytmicznego komponowania muzyki na podstawie wzorców. Opis implementacji koncepcji zaprezentowany w tym rozdziale, powinien ułatwić Czytelnikowi nawigację pomiędzy poszczególnymi modułami systemu komponowania. Autor dokonał analizy wybranych fragmentów kodu źródłowego, wskazując i komentując rozwiązania opisane w rozdziale 2. W dalszej kolejności omówione zostały badania przeprowadzone przez autora, dotyczące wpływu różnych parametrów ustawień systemu na wynikową postać kompozycji.



Rys. 3.1. Ekran ogólnych ustawień aplikacji SynthFont



Rys. 3.2. Wykres pianolowy w aplikacji SynthFont

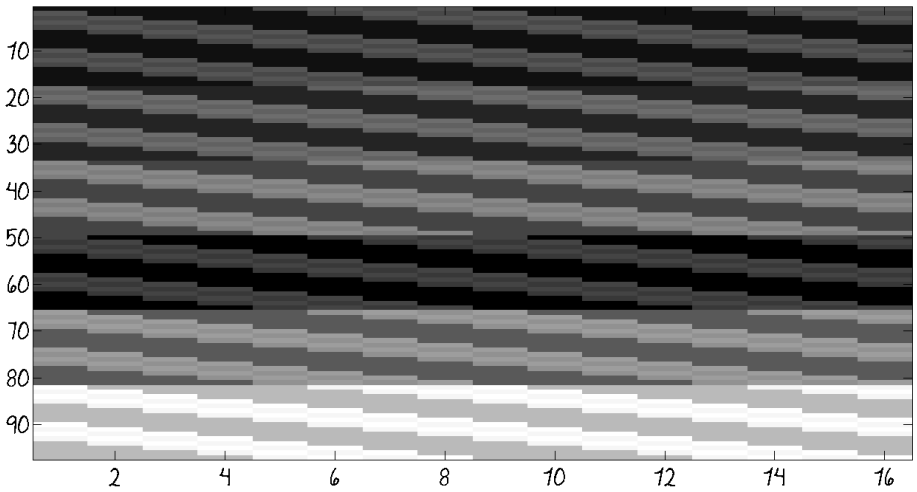
Rozdział 4

Eksperymenty i wyniki badań

W poprzednich rozdziałach autor zaprezentował dwie koncepcje algorytmicznego komponowania muzyki tła z wykorzystaniem wzorców. Pierwsza z omawianych zakładała generowanie kompozycji bez definiowania wynikowej struktury, natomiast druga wprowadzała możliwość opisu abstrakcji formy muzycznej. Naturalną więc wydaje się konieczność podziału opisywanych na dwie grupy tematyczne. W pierwszej kolejności autor przedstawił eksperymenty dotyczące generowania muzyki tła bez definiowania formy kompozycji. Następnie opisane zostały badania nad możliwościami konfiguracji systemu algorytmicznego komponowania muzyki, w którym użytkownik może definiować strukturę finalnej kompozycji. W ostatniej części tego rozdziału autor skonfrontował wyniki uzyskane na podstawie dwóch opisanych koncepcji systemu oraz dokonał podsumowania rezultatów.

4.1. Generowanie kompozycji bez opisu formy

W tej części rozdziału przeprowadzono analizę wpływu zmiany parametrów kontrolujących generowanie kompozycji na jej wynikową postać. Autor zdefiniował następujące parametry początkowe systemu:
— $dur = 0.25$ - długość dźwięku w macierzy NM wyrażona w sekundach,
— $pattern_l = 16$ - liczba wierszy (dźwięków) macierzy NM,
— $n_pattern = 32$ - liczba wzorców muzycznych wygenerowanych przez system,
— $u_pattern = 32$ - liczba wzorców muzycznych wykorzystanych w kompozycji,
— $rt = [4]$ - tablica rt powtórzeń.
Na potrzeby eksperymentu zdefiniowana została macierz KMM wygenerowana na podstawie implementacji pokazanej na listingu 5.23 w "Dodatku A. Listingi kodów". Wizualizacja wygenerowanej macierzy KMM została przedstawiona na rys. 4.1, na którym jaśniejsze komórki symbolizują wyższe dźwięki motywów muzycznych. Bazowym motywem muzycznym dla tej macierzy jest wektor wysokości dźwięków zdefiniowany w postaci $p =< 33, 33, 33, 33, 43, 45, 43, 45, 33, 33, 33, 33, 43, 45, 43, 45 >$, na którym wykonane zostały następujące przekształcenia z zachowaniem podanej kolejności:
— obrót elementów za pomocą wbudowanej funkcji $circshift$,
— dodanie do bazowego wektora p wartości 4 i obrót elementów za pomocą funkcji $circshift$,
— dodanie do bazowego wektora p wartości 5 i obrót elementów za pomocą funkcji $circshift$,
— odjęcie od bazowego wektora p wartości 12 i obrót elementów za pomocą funkcji $circshift$,
— dodanie do bazowego wektora p wartości 16 i obrót elementów za pomocą funkcji $circshift$,



Rys. 4.1. Wizualizacja macierzy KMM. Opracowanie własne.

— dodanie do bazowego wektora p wartości 17 i obrót elementów za pomocą funkcji *circshift*.

Funkcja *circshift* dokonuje przesunięcia podanej macierz lub wektora, zgodnie z parametrem podanym w postaci wektora przesunięć elementów w danym wymiarze. Przykładowo na listingu 5.23 w linii 6, po każdym obrocie pętli wektor p zostanie przesunięty w prawo o jedną pozycję, a jego zmodyfikowana wersja zostanie scalona z macierzą KMM:

```
for pp=1:length(p)
  kmm = [kmm; circshift(p,[1, pp])];
end
```

Ostatecznie macierz KMM posiada wymiarowość 97×16, a więc zbudowana jest z 97 motywów muzycznych, z których każdy zawiera 16 wysokości dźwięków tworzących pewną sekwencję.

W następnej kolejności na potrzeby opisywanych badań autor zdefiniował wektory *rhytmic_step* (4.1) i *rhytmic_value* (4.2), wykorzystywane podczas generowania wzorców muzycznych jako elementy struktury rytmicznej wzorca.

$$rhytmic_step = \begin{bmatrix} 4 \\ 8 \\ 16 \end{bmatrix} \tag{4.1}$$

$$rhytmic_value = \begin{bmatrix} [1,6,11,16] \\ [3,6,9,12,15] \\ [1,5,\ldots,13] \\ [8,11,14] \\ [8,13] \\ [9,12,15] \\ [1,5,9,13] \\ [2,4,\ldots,16] \\ [2,6,10,14] \\ [3,8,13] \\ [2,7,12] \end{bmatrix} \tag{4.2}$$

Kolejnym krokiem eksperymentu jest zdefiniowanie mini-generatorów wchodzących w skład systemu algorytmicznego komponowania muzyki. W tab. 4.1 autor zaprezentował parametry mini-generatorów zdefiniowanych w badanym przypadku. W pierwszej kolumnie podane zostały identyfikatory mini-generatorów odpowiedzialnych za generowanie poszczególnych wzorców muzycznych wchodzących w skład danej klasy wzorców. W drugiej kolumnie tabeli znajdują się oznaczenia klas mini-generatorów opisane w definicji (2.9) na stronie 56. Kolumny trzecia i czwarta zawierają odpowiednio wartości *rhytmic_step* oraz *rhytmic_value* przypisane poszczególnym mini-generatorom. W kolumnie piątej znajdują się motywy muzyczne PM, zdefiniowane w postaci skalarów i wektorów wygenerowanych na podstawie przykładowych implementacji bądź wylosowanych z utworzonej wcześniej macierzy KMM. Ostatnia kolumna zawiera numery kanałów $MIDI$ przypisane wzorcom muzycznym generowanym przez poszczególne mini-generatory.

Dla mini-generatora Mg_9 motyw PM przyjmuje postać wektora *pm*, zbudowanego na podstawie przedstawionego poniżej fragmentu implementacji.

```
step = fix( pattern_l/ (rand(1)*rhytmic_step(7)+1) );
vec = rhytmic_value{8}(1:step:end);
temppattern(vec,4) = [60:1:60+length(vec)-1]; %wektor pm
```

Liczba i wartości elementów wektora *pm* zależą od długości wektora *vec* powstałego na podstawie zmiennej *step* oraz 8-ego elementu wektora *rhytmic_step* (czyli <1,5,9,13>), symbolizującego rozmieszczenie dźwięków *pm* we wzorcu *temppattern*. Zmienna *step* wyliczana jest w sposób losowy na podstawie (4.3), gdzie wykorzystana została liczba wylosowana na podstawie rozkładu jednostajnego ciągłego z przedziału <0,1>.

$$step = int(\frac{16}{X \cdot 8 + 1}), X \sim U(0,1) \tag{4.3}$$

Podobnie dla mini-generatora Mg_{10} motyw PM przyjmuje postać $\widetilde{pm}$ obliczoną na podstawie implementacji:

```
step = fix( pattern_l/ (rand(1)*rhytmic_step(7)+1) );
vec = rhytmic_value{7}(1:step:end);
temppattern(vec,4) = [60:1:60+length(vec)-1]; %wektor pm
```

Mini-generator	Klasa	*rhytmic_step*	*rhytmic_value*	PM	Kanał *MIDI*
Mg_1	D	16	[1,6,11,16]	36	10
Mg_2	D	16	[2,7,12]	36	10
Mg_3	D	16	[3,6,9,12,15]	40	10
Mg_4	D	16	[1,5,9,13,1,5,9,13]	42	10
Mg_5	D	16	[3,8,13]	51	10
Mg_6	D	8	[8,11,14]	47	10
Mg_7	D	4	[8,13]	48	10
Mg_8	D	8	[1,6,11,16]	54	10
Mg_9	D	8	[1,5,9,13]	pm	10
Mg_{10}	D	8	[9,12,15]	$\widetilde{pm}$	10
Mg_{11}	D	8	[9,12,15]	57	10
Mg_{12}	M	16	[2,4,...,16]	z KMM	1
Mg_{13}	M	4	[2,6,10,14]	z Mg_{12}	2
Mg_{14}	M	16	[1,5,9,13,1,5,9,13]	z Mg_{12}	3
Mg_{15}	M	4	[2,6,10,14]	z Mg_{12}	4
Mg_{16}	M	16	[3,6,9,12,15]	z Mg_{12}	5
Mg_{17}	M	16	[2,4,...,16]	z Mg_{12}	6

Tab. 4.1. Parametry zdefiniowanych mini-generatorów. Opracowanie własne.

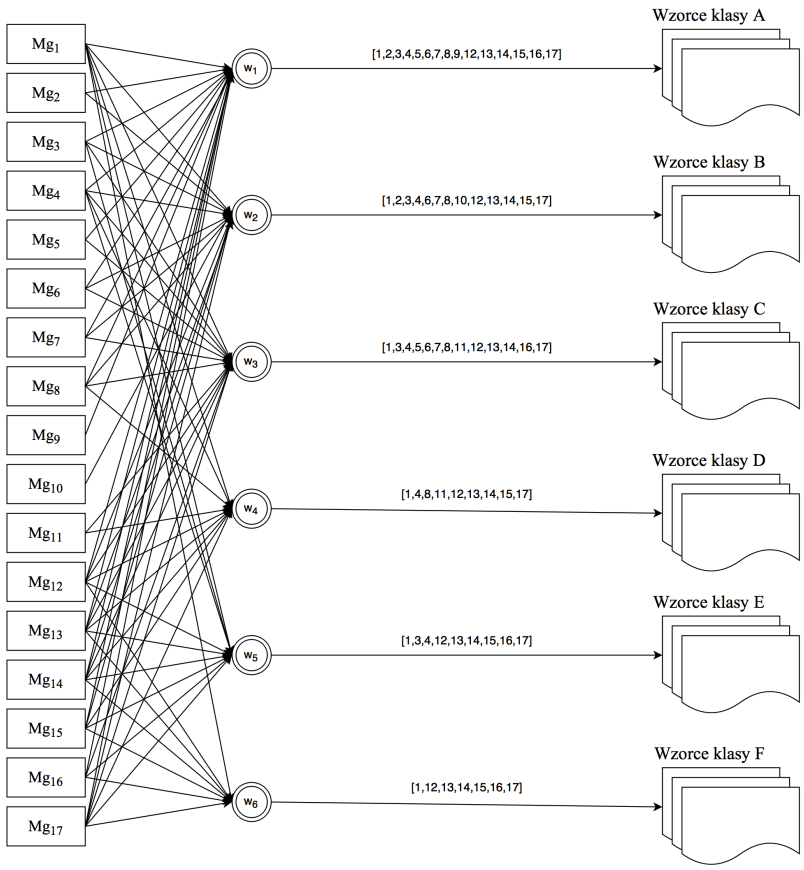
Zmienna *step* wyliczana jest w tym przypadku identycznie jak dla *pm* (4.3). Jedyną różnicą jest wykorzystany wektor *rhytmic_value*, który w tym wariancie przyjmuje postać $< 9, 12, 15 >$.

W eksperymencie autor zdefiniował architekturę klas wzorców, której schemat budowy przedstawiony został na rys. 4.2. Bloki oznaczone przez Mg_x symbolizują zdefiniowane mini-generatory biorące udział w procesie generowania wzorca muzycznego. Od każdego bloku odchodzi pewna liczba odcinków łączących się w jednym z sześciu węzłów w symbolizujących proces konkatenacji wzorca muzycznego.

Przynależność generowanego wzorca do jednej z klas wzorców $A - F$ określa wektor identyfikatorów mini-generatorów ujętych w nawiasie kwadratowym umiejscowiony nad strzałką łączącą węzeł w z symbolem klasy wzorców po prawej stronie rys. 4.2. Dla przykładu wzorzec w_1 generowany jest przez mini-generatory $[1, 2, 3, 4, 5, 6, 7, 8, 9, 12, 13, 14, 15, 16, 17]$ i przynależy do klasy A. Tak zadeklarowana architektura powoduje, że w danej klasie znajduje się zbiór wzorców muzycznych podobnych do siebie pod względem budowy, tzn. wykorzystanych wysokości dźwięków muzycznych, tempa, wykorzystanych wartości rytmicznych i alokacji dźwięków w całym wzorcu.

Na rys. 4.3 przedstawiono dwa wykresy pianolowe przykładowych wzorców muzycznych wygenerowanych w zdefiniowanej klasie A wg schematu z rys. 4.2. Podobieństwo obu wzorców wyraża się poprzez występowanie podobnych struktur rytmicznych, melodycznych i harmonicznych występujących w takich samych kanałach $MIDI$ (np. przebiegi dźwięków w kanałach $ch7$ oraz $ch4$ na rys. 4.3).

Brzmienie wymienionych wzorców można usłyszeć odtwarzając pliki $w_411_1_A.flac$ oraz $w_411_2_A.flac$ zamieszczone na dołączonej do pracy płycie CD. Wymienione pliki muzyczne zawierają wzorce, w których

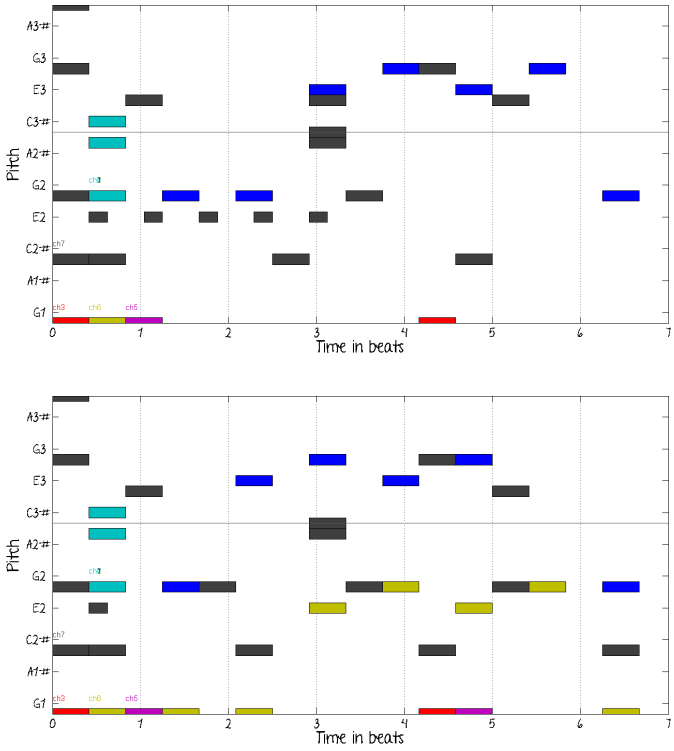


Rys. 4.2. Schemat generowania przykładowej architektury klas wzorców przez zdefiniowane mini-generatory Mg. Opracowanie własne.

do syntezy wykorzystane zostały domyślne ustawienia brzmień dla syntezatora. Dla kanałów $MIDI$ o numerze innym niż 10 wykorzystany został instrument *AcousticGrandPiano*. Dla kanału 11-ego wykorzystany został domyślny zestaw perkusyjny *Standard*. Do syntezy wykorzystany został bank brzmień *Evanssence*2 w wersji 2.1.

Przykładową kompozycję wygenerowaną na podstawie ustawień przedstawionych na stronie 97 usłyszeć można odtwarzając plik $k_raw_411.flac$. Podobnie jak wzorce muzyczne, wygenerowana kompozycja wykorzystuje domyślnie ustawione opcje instrumentów. Powtarzalność fragmentów kompozycji uzależniona jest od zmiennej rt, która jest macierzą zawierającą możliwe kombinacje powtórzeń wylosowanego wzorca muzycznego wykorzystanego w procesie generowania kompozycji.

W dalszej części tego rozdziału przedstawione zostały eksperymenty bazujące na zmianach wartości parametrów opisanych na stronie 97. Autor dokonał analizy wpływu zmian wymienionych czynników na formę generowanej kompozycji, zachowując przy tym odrębność każdego z parametrów i jego indywidualne znaczenie w procesie komponowania. Podczas badań zdefiniowana została konfiguracja instrumentów wykorzystanych przy syntezie kompozycji wg tab. 4.2.



Rys. 4.3. Wykresy pianolowe dla dwóch przykładowych wzorców klasy A. Opracowanie własne.

4.1.1. Modyfikacja parametru długości dźwięków

Pierwszym z badanych parametrów jest długość dźwięków generowanych przez mini-generatory, która w opisywanej analizie została zdefiniowana globalnie. Kolejne przyjęte wartości długości dźwięków zostały przedstawione w tab. 4.3. Dla pozostałych parametrów przyjęto wartości wg ustawień przedstawionych na stronie 97.

Zwiększenie wartości długości dźwięków dla wszystkich wzorców muzycznych przy zachowaniu stałej wartości tempa, powoduje wydłużenie wzorca muzycznego, a w konsekwencji całego utworu. Słuchając podanych w tab. 4.3 kompozycji można zauważyć bardzo spokojny charakter kompozycji $dur_1.flac$ przy gwałtownym przebiegu kompozycji $dur_0-125.flac$.

Zmiana charakteru utworu wiąże się z przyjętą koncepcją budowania wzorca muzycznego. Dla przypomnienia wzorzec muzyczny w przeprowadzanej analizie posiada długość 16 dźwięków, co oznacza, że przy zmianie długości dźwięków, zmieniają się również odległości (pauzy) pomiędzy poszczególnymi dźwiękami w ramach wzorca muzycznego. Jest to związane z jego definicją przedstawioną w rozdziale 2.2 na stronie 53.

Mając na uwadze przyjęte w tab. 4.3 wartości długości dźwięków autor postanowił wprowadzić zmiany tych wartości nie w sposób globalny, a lokalny - dla pewnej grupy mini-generatorów bądź pojedynczego mini-generatora. Wprowadzone modyfikacje długości dźwięków przedstawiono w tab. 4.4, natomiast w pliku $dur-mix.flac$ znajduje się przykładowa kompozycja wygenerowana na podstawie zmienionych ustawień parametru dur.

Kanał $MIDI$	Barwa instrumentu
1	Percusive Organ
2	Choral Aahs
3	Synth Bass 2
4	Synth Strings2
5	Brightness
6	Atmosphere
10	Dance 2(Drums Set)

Tab. 4.2. Ustawienia barw instrumentów dla generowanych kompozycji z wykorzystaniem banku brzmień $Evanessence2.sf2$. Opracowanie własne.

Długość dźwięku	Długość utworu	Nazwa pliku
0.125 sek.	1 min. 7 sek.	$dur_0-125.flac$
0.25 sek.	2 min. 13 sek.	$dur_0-25.flac$
0.5 sek.	4 min. 25 sek. 2	$dur_0-5.flac$
1 sek.	8 min. 49 sek.	$dur_1.flac$

Tab. 4.3. Przyjęte długości dźwięków oraz długości wygenerowanych kompozycji. Opracowanie własne.

Na rys. 4.4 przedstawiono wykres pianolowy kompozycji zapisanej w pliku $dur-mix.flac$. Widać wyraźne rozróżnienie w wielkości bloków symbolizujących dźwięki wygenerowane przez poszczególne mini-generatory. Kanał *MIDI* o numerze 4 (na wykresie oznaczony przez $ch4$) zawiera wzorce muzyczne wygenerowane przez mini-generator Mg_{15} natomiast kanał 5 (na wykresie oznaczony jako $ch5$) przedstawia wzorce muzyczne wygenerowane na podstawie mini-generatora Mg_{16}. W kanale *MIDI* przeznaczonym na instrumenty perkusyjne zawarty został wzorzec muzyczny wygenerowany przez Mg_3 i inne mini-generatory generujące partię perkusyjną.

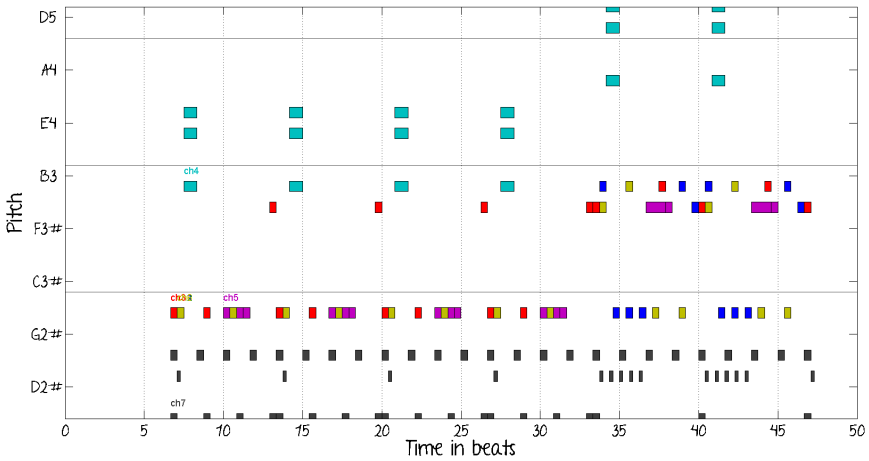
Rozróżnienie wprowadzone za pomocą modyfikacji wartości długości dźwięków dla poszczególnych mini-generatorów wprowadza możliwość kształtowania kompozycji o zmiennej dynamice rytmicznej ścieżek ją tworzących. Autor chciał w ten sposób przedstawić funkcjonalność zmiany charakteru rytmicznego utworu przy zmianie jedynie parametru dur. W dalszej części tego rozdziału zaprezentowane zostały kolejne możliwości wpływania na kształt wynikowej kompozycji muzycznej.

4.1.2. Modyfikacja parametru długości wzorca

Poprzednio przedstawiono wpływ zmian długości dźwięków na kształt generowanej kompozycji, a w szczególności rytmicznego charakteru poszczególnych wzorców muzycznych, które zostały wykorzystane do jej skomponowania. Kolejnym parametrem mającym wpływ na kształt kompozycji wyjściowej w prezentowanej koncepcji komponowania muzyki jest długość wzorca muzycznego oznaczona jako $pattern_l$. W tab. **??** znajdują się przyjęte długości wzorców muzycznych oraz długości wynikowych kompozycji dostępnych pod wskazanymi nazwami plików na płycie CD dołączonej do pracy. Dla pozostałych parametrów przyjęto wartości wg ustawień przedstawionych na stronie 97.

Długość dźwięku	Mini-generatory(y)
0.125 sek.	Mg_3
0.25 sek.	Mg_1,Mg_2,Mg_4-Mg_{14},Mg_{17}
0.5 sek.	Mg_{15}
1 sek.	8 Mg_{16}

Tab. 4.4. Zmodyfikowane długości dźwięków dla poszczególnych mini-generatorów. Opracowanie własne.



Rys. 4.4. Wykres pianolowy fragmentu kompozycji wygenerowanej na podstawie ustawień z tab. 4.4. Opracowanie własne.

Zmiana długości wzorca muzycznego wpływa na zmianę długości kompozycji wynikowej oraz liczbę dźwięków muzycznych wchodzących w skład danego wzorca. Przy założeniu wykorzystania tego samego motywu muzycznego p (np. tego przedstawionego na stronie 97), dłuższy wzorzec będzie bardziej złożony pod względem muzycznym niż wzorzec od niego krótszy, a co za tym idzie - cała kompozycja muzyczna będzie miała bardziej złożony charakter.

Aspekt zmiany złożoności jest związany również z budową wzorca muzycznego. Wzorce o tych samych długościach, lecz różnych wartościach czasu trwania dźwięków, różnią się pod względem struktury, co przedstawiono na przykładzie z rys. 2.3 na stronie 55. Przy założeniu tych samych długości wyrażonych w jednostkach czasu lub bpm, wzorzec o mniejszej wartości długości dźwięków będzie zawierać ich większą liczbę niż wzorzec o tej samej długości, zwierający dłuższe dźwięki. Relacja ta wynika bezpośrednio z przyjętej koncepcji budowy wzorca muzycznego i ma znaczący wpływ na kształt wynikowej kompozycji.

Na rys. 4.5 przedstawiono wykresy pianolowe dwóch kompozycji różniących się pod względem przyjętej długości wzorca muzycznego, przy czym długości wzorców przyjęto na podstawie tab. 4.5. W prezentowanym wyniku umieszczono dwie skrajne pod względem budowy kompozycje: $pattern_l-4.flac$ i $pattern_l-32.flac$.

Pierwsza z nich została wygenerowana na podstawie wzorców o długości 4 dźwięków. Dokonując analizy słuchowej i strukturalnej można usłyszeć powtarzalność utworzonych wzorców oraz ich wyraźnie zaznaczone repetycje w toku całej kompozycji. Usłyszane fragmenty są względnie prymitywne w aspekcie melodyki,

Długość wzorca	Długość utworu	Nazwa pliku
4 dźwięków	34 sek.	$pattern_l - 4.flac$
8 dźwięków	1 min. 7 sek.	$pattern_l - 8.flac$
16 dźwięków	2 min. 13 sek. 2	$pattern_l - 16.flac$
32 dźwięków	4 min. 25 sek.	$pattern_l - 32.flac$

Tab. 4.5. Przyjęte długości wzorców muzycznych oraz wygenerowanych kompozycji. Opracowanie własne.

harmoniki i rytmiki oraz posiadają znamiona muzyki automatycznej wygenerowanej przez maszynę.

Subiektywnie odmienne odczucie pojawia się po odsłuchaniu kompozycji $pattern_l - 32.flac$ zbudowanej ze wzorców o długości 32 dźwięków. Powtarzalność wzorców została zamaskowana poprzez ich względnie złożoną strukturę, a ich wydłużenie spowodowało wykorzystanie w procesie komponowania większej liczby dźwięków bazowego motywu muzycznego p (przedstawionego na stronie 97).

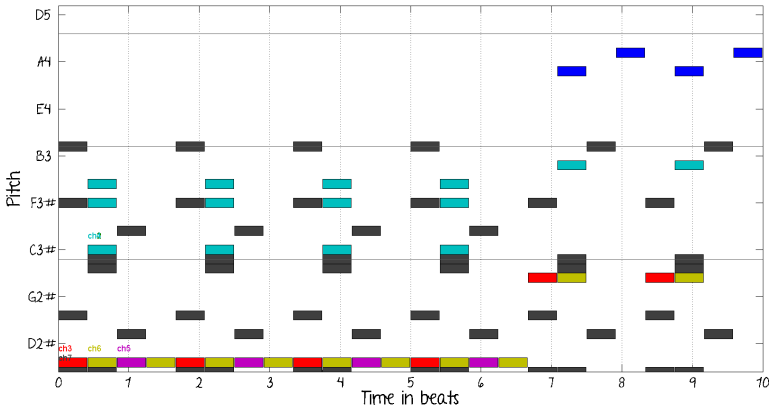
Przesłuchując całą grupę wygenerowanych w ten sposób utworów ($pattern_l - 4.flac$, $pattern_l - 8.flac$, $pattern_l - 16.flac$ oraz $pattern_l - 32.flac$) można prześledzić stopniowe przechodzenie od utworu nacechowanego powtarzalnością wygenerowanych motywów po utwór posiadający znamiona kompozycji rozwiniętej do poziomu kompozycji z gatunku muzyki tła. Równie ciekawe efekty można uzyskać łącząc zmiany parametrów długości wzorca i długości czasu trwania dźwięków go tworzących. Przykładowe kompozycje wygenerowane przez autora opisano w części "Wnioski z eksperymentów" w rozdziale 4.1 na stronie 108, natomiast w kolejnej części pracy zaprezentowany został wpływ zmian zbioru wzorców muzycznych wykorzystywanych do tworzenia kompozycji wynikowej.

4.1.3. Modyfikacja parametrów zbioru wygenerowanych i wykorzystanych wzorców muzycznych

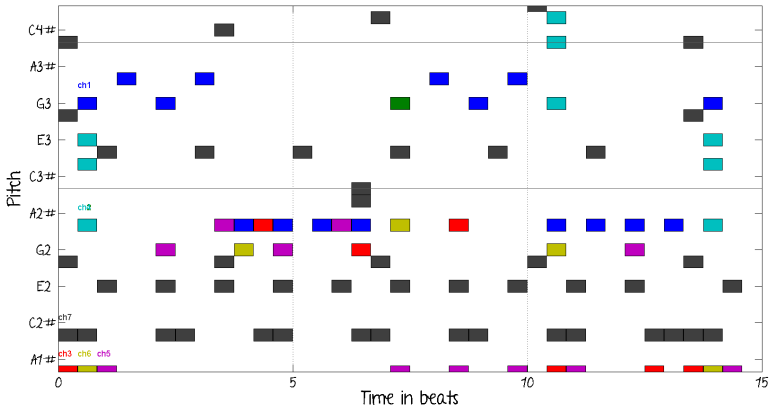
Zbiór wzorców muzycznych stanowi źródło danych dla procesu tworzenia kompozycji a jego przykładowa implementacja została przedstawiona na stronie 168 w części "Dodatek A. Listingi kodów". W przytoczonym listingu zbiór wygenerowanych wzorców został oznaczony jako $pattern_struct$, natomiast liczbę generowanych oraz wykorzystywanych w kompozycji wzorców muzycznych zdefiniowano odpowiednio przez zmienne n i $seq_patterns$, oznaczające parametry $n_pattern$ i $u_pattern$ zaprezentowane w tab. 4.6. Dla pozostałych parametrów przyjęto wartości wg ustawień przedstawionych na stronie 97.

Dla przykładowych wartości $n_pattern = 16$ oraz $u_pattern = 8$ zbiór wzorców posiadać będzie 16 elementów, a kompozycja wynikowa składać się będzie z 8 wzorców muzycznych. Przy przyjętej stałej strukturze tablicy powtórzeń $rt = [4]$ w przytoczonym przykładzie wystąpią jedynie 2 losowania wzorców muzycznych, które zostaną powielone czterokrotnie. Tak więc w pokazanej w tab. 4.6 strukturze przebiegu eksperymentu maksymalna liczba losowań wzorca występuje w kompozycji $pattern_n32 - u64.flac$ i wynosi 16.

W omawianej koncepcji systemu komponowania bez opisu formy utworu liczności zbiorów wzorców wygenerowanych oraz wykorzystanych do generowania kompozycji mają wpływ na powtarzalność fragmentów wynikowego utworu. Szczególnie



A



B

Rys. 4.5. Wykresy pianolowe fragmentów kompozycji $pattern_l-4.flac$ (A) oraz $pattern_l-32.flac$ (B) wygenerowanych na podstawie ustawień z tab. 4.5. Opracowanie własne.

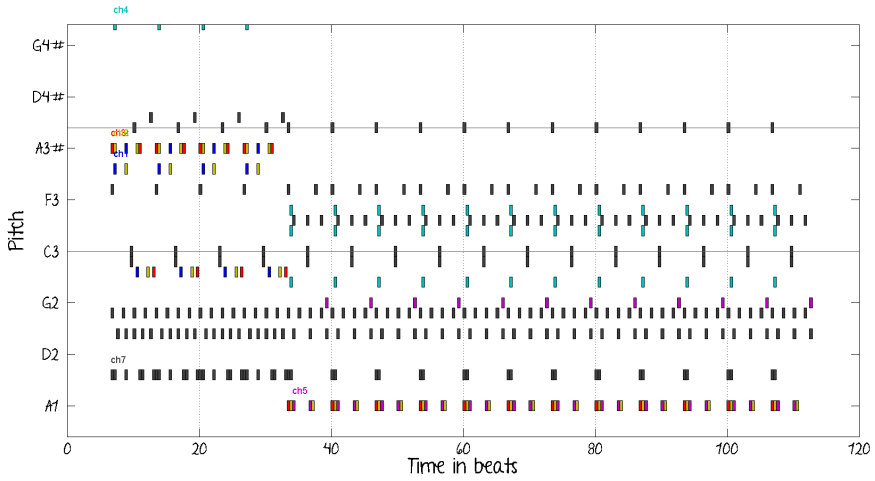
ważna jest tutaj relacja pomiędzy parametrami $n_pattern$ a $u_pattern$. W przypadku "małych" wartości $n_pattern$ w stosunku do względnie "dużych" $u_pattern$ może wystąpić zjawisko polegające na zapętleniu repetycji (ponownym wylosowaniu) wzorca muzycznego w procesie komponowania utworu. Taką sytuację przedstawiono na rys. 4.6 przedstawiającym wykres pianolowy kompozycji $pattern_n16-u16.flac$.

Na wykresie pianolowym widać odznaczający się początkowy fragment kompozycji, składający się z 4-ech wzorców oraz zapętlony fragment 12 identycznych wzorców muzycznych. Proces komponowania "utknął" na wybranym wzorcu i powielił go czterokrotnie w 3-ech kolejnych losowaniach. Cecha ta stanowi przypadłość omawianej koncepcji i pojawia się przy względnie małych zbiorach wzorców muzycznych, z których losuje się dużą liczbę wzorców wykorzystanych w dalszym etapie komponowania. Prawdopodobieństwo wylosowania tego samego wzorca w sposób oczywisty maleje wraz ze wzrostem liczby wygenerowanych wzorców i/lub zmniejszeniem wzorców losowanych do kompozycji wynikowej.

Opisany problem nie występuje w koncepcji procesu generowania przedstawionej w rozdziale 4.2. Zanim jednak autor przeanalizuje odmienne podejście do problematyki algorytmicznego komponowania muzyki, w dalszej części pracy zostanie omówiony jeszcze jeden aspekt mający wpływ na finalny kształt kompozycji - zmiana struktury tablicy zawierającej dozwolone liczby powtórzeń wzorca.

Liczba wygenerowanych wzorców (n)	Liczba użytych wzorców (u)	Nazwa pliku
16	8	$pattern_n16 - u8.flac$
16	16	$pattern_n16 - u16.flac$
16	32	$pattern_n16 - u32.flac$
32	16	$pattern_n32 - u8.flac$
32	32	$pattern_n32 - u32.flac$
32	64	$pattern_n32 - u64.flac$

Tab. 4.6. Przyjęte wartości liczności zbioru wygenerowanych (n) i wykorzystanych (u) wzorców muzycznych. Opracowanie własne.



Rys. 4.6. Wykres pianolowy kompozycji $pattern_n16 - u16.flac$ - przykład zapętlenia repetycji wzorców. Opracowanie własne.

4.1.4. Modyfikacja struktury tablicy możliwych powtórzeń wzorca

Ostatnim opisywanym parametrem mającym znaczący wpływ na strukturę kompozycji jest budowa tablicy rt, której domyślną wartość w przeprowadzonych eksperymentach przedstawiono na stronie 97. Tablica liczby powtórzeń jest wykorzystywana w etapie losowania wzorca, który ma znaleźć się w danym miejscu generowanej kompozycji.

W tab. 4.7 zaprezentowane zostały przyjęte struktury tablicy rt oraz odpowiadające im nazwy plików wygenerowanych kompozycji zawartych na dołączonej do pracy płycie CD. Autor dokonał analizy struktury utworzonych przez system utworów z punktu widzenia ulokowania repetycji wzorców muzycznych wewnątrz kompozycji.

Na wykresach z rys. 4.7 oznaczone zostały obszary występowania wzorców muzycznych występujących pojedynczo (linia przerywana) oraz podwójnie (linia ciągła). Obszary nieoznaczone zawierają wzorce czterokrotnie powtórzone w wyniku losowania. Losowość powtórzeń połączona z procesem wyboru wzorca z jednej strony wprowadza pewną chaotyczność do struktury kompozycji, z drugiej zaś narusza stagnację repetycji wzorców poprzez utworzone wtrącenia w postaci obszarów

przedstawionych na rys. 4.7. Wzorce powtórzone jednokrotnie mogą wykazywać wzajemne podobieństwo, zarówno w aspekcie lokalnym (występujące jeden po drugim) jak i globalnym (rozmieszczone w różnych miejscach kompozycji). Przykład takiego zjawiska przedstawia górny wykres na rys. 4.7. Jednocześnie kolejne wzorce mogą różnić się od siebie w znacznym stopniu, niezależnie od liczby powtórzeń, co zaprezentowano na dolnym wykresie z rys. 4.7, gdzie pierwsze wystąpienie podwójnego powtórzenia wzorca (linia ciągła) leży w sąsiedztwie kolejnego, rozbudowanego o dodatkowe partie instrumentalne lecz o zbliżonej strukturze pozostałych.

Warto zwrócić również uwagę na wpływ struktury tablicy rt na długość i "spójność" budowy kompozycji wynikowej. W przypadku, gdy przyjęte są stałe wartości pozostałych parametrów modelu systemu (strona 97), kompozycja może być wydłużana poprzez definiowanie dużych wartości powtórzeń wzorców. Jednocześnie w przypadku zdefiniowania tablicy powtórzeń za pomocą względnie małych wartości, kompozycja ulega nie tylko skróceniu ale i również jej przebieg wydaje się być (w odczuciu autora) bardziej zmienny i chaotyczny.

Omówiona w tym rozdziale istota tablicy powtórzeń wzorca jest ostatnim analizowanym przez autora parametrem systemu algorytmicznego komponowania muzyki, nieuwzględniającego istoty formy utworu wynikowego. W ostatniej części tego rozdziału opisane zostały wnioski wypływające z przeprowadzonych nad systemem badań oraz zaprezentowana została przykładowa kompozycja wygenerowana na podstawie autorskiej konfiguracji systemu i doborem instrumentów w procesie syntezy utworu.

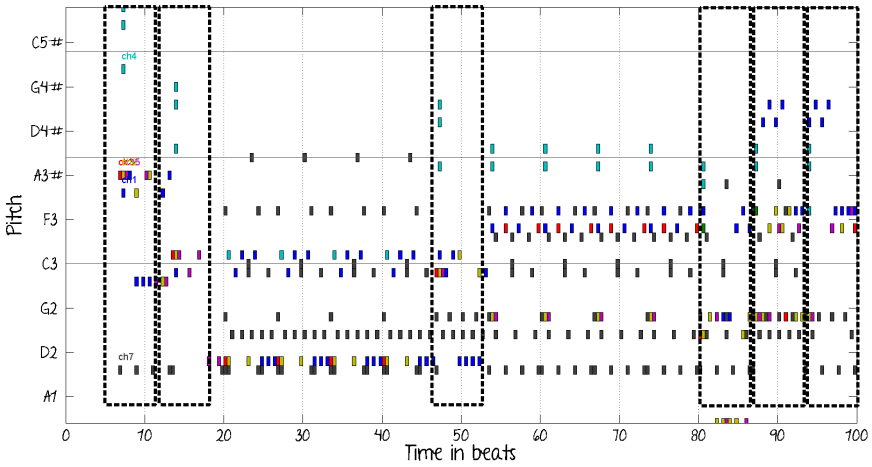
4.1.5. Wnioski z eksperymentów

W koncepcji algorytmicznego komponowania muzyki na podstawie wzorców z pominięciem formalnego opisu formy generowanej kompozycji autor przeanalizował cztery parametry przyczyniające się do zmiany jej struktury, tj. długość dźwięków muzycznych, długość wzorca, strukturę zbioru wzorców oraz strukturę tablicy powtórzeń wzorców.

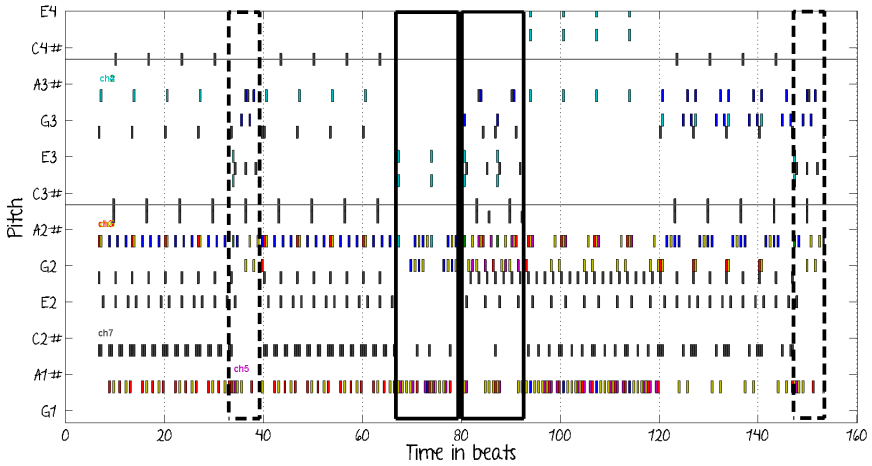
Przedstawione wyniki zmian długości dźwięków muzycznych jak i całych wzorców sugerują ich wpływ na powtarzalność fragmentów kompozycji oraz jej złożoność struktury. Przykłady wygenerowanych kompozycji wymienionych w rozdziałach 4.1.1 i 4.1.2 ukazują możliwości systemu w zakresie tworzenia muzyki o zmiennej dynamice i złożoności struktury. Przy przyjętym stałym tempie utworu oraz stałej długości wzorca muzycznego, zmiana parametru długości tworzącego go dźwięków powoduje zmianę w dynamice całej kompozycji: im krótsze dźwięki, tym bardziej dynamiczny wydaje się być efekt końcowy pracy systemu (kompozycje

Tablica rt	Nazwa pliku
$[1]$	$pattern_rt-1.flac$
$[2]$	$pattern_rt-2.flac$
$[1,2]$	$pattern_rt-1-2.flac$
$[1,4]$	$pattern_rt-1-4.flac$
$[2,4]$	$pattern_rt-2-4.flac$
$[1,2,4]$	$pattern_rt-1-2-4.flac$

Tab. 4.7. Przyjęte struktury tablicy rt. Opracowanie własne.



A



B

Rys. 4.7. Wykresy pianolowe kompozycji $pattern_rt-1-4.flac$ (A) i $pattern_rt-1-2-4.flac$ (B) z zaznaczonymi obszarami występowania pojedynczych (linią przerywaną) i podwójnych (linią ciągłą) wzorców. Opracowanie własne.

$dur_0-125.flac$ i $dur_1.flac$ opisane na stronie 102). Natomiast w sytuacji gdy przyjęta została stała długość dźwięków muzycznych i tempa kompozycji, zwiększenie długości wzorca powoduje zwiększenie ogólnej złożoności kompozycji - wzorce muzyczne zbudowane są z większej liczby dźwięków, które mogą tworzyć bardziej rozbudowane tematy muzyczne (kompozycje $pattern_l-4.flac$ i $pattern_l-32.flac$ ze strony 106).

Autor w ramach doświadczeń przeprowadził również próby zmian proporcji pomiędzy zbiorem wygenerowanych a wykorzystanych wzorców kompozycji muzycznej oraz zmian budowy tablicy powtórzeń wzorca rt. W wyniku przeprowadzenia kilku procesów komponowania otrzymano wynik nacechowany wielokrotnymi powtórzeniami tego samego wzorca (kompozycja $pattern_n16-u16.flac$ na stronie 107), co odsłania pewną słabość opisywanej koncepcji systemu w przypadku słabo urozmaiconego zbioru wzorców muzycznych. Jednocześnie kompozycje $pattern_rt-1-4$ i $pattern_rt-1-2-4$ ukazują różne aspekty powtórzeń wzorców muzycznych w ujęciu kompletnej kompozycji oraz jej chaotyczności struktury w przypadku rozbudowy tablicy powtórzeń rt.

Zdaniem autora najciekawsze efekty pracy omawianego systemu algorytmicznego komponowania muzyki uzyskuje się poprzez zmiany, a właściwie dopasowanie wszystkich wartości parametrów systemu. Aby potwierdzić swoją tezę autor zaprezentował kilka przykładowych wyników wygenerowanych na podstawie różnych konfiguracji ustawień. Wszystkie kompozycje znajdują się na płycie CD dołączonej do niniejszej pracy.

Pierwszym omawianym przykładem jest kompozycja $s1 - 415.flac$, do syntezy której wykorzystano 6 kanałów *MIDI* z jedynie 2 instrumentami banku $Evanessence2$, tj. $Ukulele$ oraz $Latin(Loops)$. W procesie komponowania autor zdefiniował następujące wartości parametrów systemu:
— $dur = 0.25$ - długość dźwięku w macierzy NM wyrażona w sekundach,
— $pattern_l = 32$ - liczba wierszy (dźwięków) macierzy NM,
— $n_pattern = 128$ - liczba wzorców muzycznych wygenerowanych przez system,
— $u_pattern = 64$ - liczba wzorców muzycznych wykorzystanych w kompozycji,
— $rt = [4]$ - tablica rt powtórzeń.
Bazowym motywem dla wzorców muzycznych jest wektor wysokości dźwięków zdefiniowany w postaci: $p =< 50, 48, 50, 53, 55, 43, 41, 43, 36, 38 >$, który posłużył do budowy macierzy KMM zdefiniowanej wg. listingu 5.24 na stronie 190. Macierz KMM składa się z trzech "części" zawierających kombinacje wygenerowane na podstawie trzech wersji wektora p: podstawowej, zwiększonej o wartość 4 (linia 10) oraz zwiększonej o wartość 7 (linia 16). Tak więc górna "część" macierzy KMM zbudowana jest na podstawie kombinacji wektora p, środkowa stanowi kombinacje wektora $p_1 =< 54, 52, 54, 57, 59, 47, 45, 47, 40, 42 >$ a dolna - kombinacje wektora $p_2 =< 57, 55, 57, 60, 62, 50, 48, 50, 43, 45 >$.

Konfiguracja mini-generatorów wykorzystanych w procesie komponowania została przedstawiona w tab. 4.8. Mini-generatory od Mg_1 do Mg_{11} należą do klasy generującej partie instrumentów perkusyjnych, natomiast mini-generatory od Mg_{12} do Mg_{18} generują partie melodii, przy czym w tym wypadku są to partie odtwarzane przez jeden instrument muzyczny. Dla urozmaicenia części rytmicznej kompozycji, autor zdefiniował dynamiczną strukturę motywów PM dla wzorców Mg_9, Mg_{10} i Mg_{11}.

Dla mini-generatora Mg_9 motyw PM przyjmuje postać wektora pm, zbudowanego na podstawie implementacji:

```
step = fix(pattern_l/ fix(rand(1)*rhytmic_step(7)+1));
vec = rhytmic_value{7}(1:step:end);
temppattern(vec,4) = [38:1:38+length(vec)-1];
```

Podobnie dla mini-generatora Mg_{10} motyw PM przyjmuje postać wektora $\widetilde{pm}$, zbudowanego na podstawie implementacji:

```
step = fix(pattern_l/ fix(rand(1)*rhytmic_step(7)+1));
vec = rhytmic_value{7}(1:step:end);
temppattern(vec,4) = [35:1:35+length(vec)-1];
```

Mini-generator Mg_{11} generuje dźwięki na podstawie wektora $\widehat{pm}$, zdefiniowanego wg. implementacji:

```
step = fix(pattern_l/fix(rand(1)*rhytmic_step(7)+1));
vec = rhytmic_value{1}(1:step:end);
temppattern(vec,4) = ones(1,length(vec))*38;
```

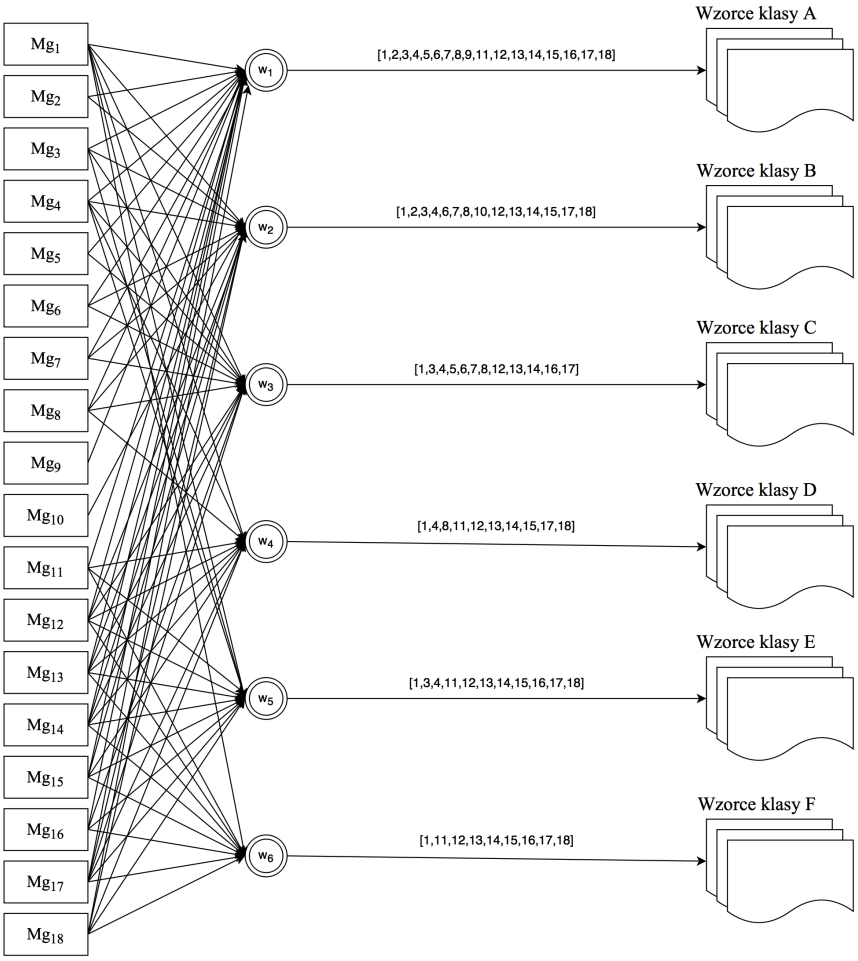
Mini-generator	Klasa	*rhytmic_step*	*rhytmic_value*	PM	Kanał *MIDI*
Mg_1	D	32	[1,5,9,...,29]	36	10
Mg_2	D	32	[2,5,8,...,32]	36	10
Mg_3	D	16	[2,5,8,...,32]	40	10
Mg_4	D	32	[1,2,...,32]	42	10
Mg_5	D	32	[2,5,8,...,32]	51	10
Mg_6	D	16	[16,20,24,28,32]	47	10
Mg_7	D	16	[16,19,22,...,31]	48	10
Mg_8	D	16	[1,5,9,...,29]	54	10
Mg_9	D	8	[17,21,25,29]	pm	10
Mg_{10}	D	8	[17,21,25,29]	$\widetilde{pm}$	10
Mg_{11}	D	8	[1,5,9,...,29]	$\widehat{pm}$	10
Mg_{12}	M	32	[2,6,10,...,30]	z KMM	1
Mg_{13}	M	8	[2,6,10,...,30]	z Mg_{12}	2
Mg_{14}	M	32	[1,6,11,...,31]	z Mg_{12}	3
Mg_{15}	M	16	[2,4,6,...,32]	z Mg_{12}	4
Mg_{16}	M	32	[3,6,9,...,30]	z Mg_{12}	5
Mg_{17}	M	32	[1,2,3,...,32]	z Mg_{12}	6
Mg_{18}	M	16	[2,5,8,...,32]	z Mg_{12}	7

Tab. 4.8. Parametry zdefiniowanych mini-generatorów dla kompozycji $s1-415.flac$. Opracowanie własne.

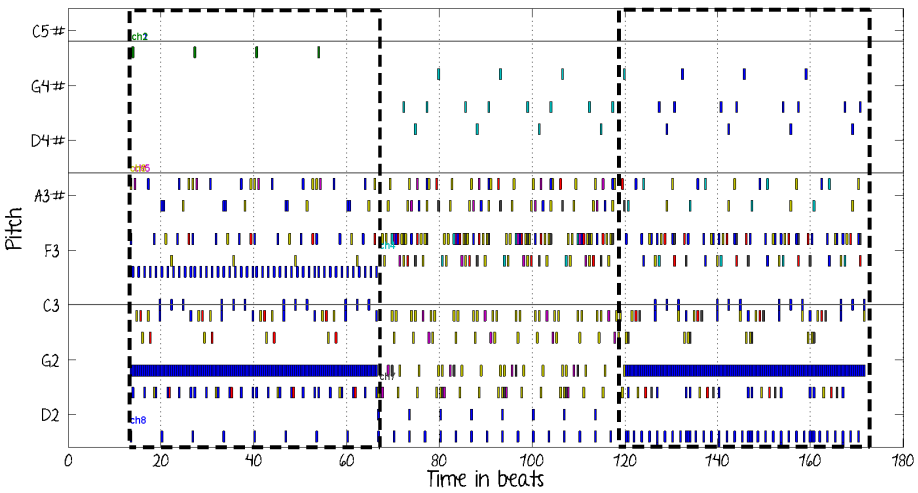
Dla tak zdefiniowanych mini-generatorów utworzona została architektura generowania kompozycji wg schematu przedstawionego na rys. 4.8. Podobnie jak na rys. 4.2 ze strony 101, bloki oznaczone przez Mg_x symbolizują zdefiniowane mini-generatory biorące udział w procesie generowania wzorca muzycznego. Od każdego bloku odchodzi pewna liczba odcinków łączących się w jednym z sześciu węzłów w symbolizujących proces konkatenacji wzorca muzycznego.

Wygenerowane wzorce muzyczne posiadają strukturę złożoną z maksymalnie 32 dźwięków muzycznych, natomiast w kompozycji końcowej występują w postaci grup zawierających 4-ry powtórzenia wzorca. Właściwość tę widać szczególnie na rys. 4.9 przedstawiającym fragment kompozycji $s1-415.flac$ zbudowany z 3-ech wzorców. Podobieństwo w budowie jest zauważalne przede wszystkim pomiędzy pierwszą a ostatnią częścią przedstawianego fragmentu, w których mini-generatory partii perkusyjnych utworzyły zbliżone struktury rytmiczne (oznaczone na rys. 4.9).

Podobieństwa w budowie poszczególnych grup wzorców są słyszalne również w kolejnych częściach przedstawianej kompozycji. Dodatkowo występujące po sobie kolejne fragmenty oscylują wokół podobnego motywu muzycznego (zbudowanego na podstawie wektora p, p_1 lub p_2). Utworzona kompozycja spełnia definicję muzyki tła (podanej we wcześniejszej części pracy) jednak, w opinii autora, jej "powtarzalność" i swego rodzaju "chaotyczność" mogą stanowić o słabych cechach prezentowanej



Rys. 4.8. Schemat generowania architektury klas wzorców przez zdefiniowane mini-generatory Mg dla kompozycji $s1 - 415.flac$. Opracowanie własne.



Rys. 4.9. Wykres pianolowy fragmentu kompozycji $s1 - 415.flac$ z zaznaczonymi grupami wzorców wykazujących podobieństwo budowy. Opracowanie własne.

koncepcji systemu. Podobne cechy komponowanej muzyki można usłyszeć w innych kompozycjach, umieszczonych na dołączonej do pracy płycie CD (strona 191). Biorąc pod uwagę powyższe stwierdzenia oraz analizując wyniki przeprowadzonych eksperymentów, autor zaproponował inną, odmienną koncepcję algorytmicznego komponowania muzyki, uwzględniającą definicję formy generowanej kompozycji.

4.2. Generowanie kompozycji bazujące na opisie formy

W poprzednim rozdziale poświęconym przeprowadzonym badaniom nad użytecznością algorytmicznego systemu komponowania muzyki na podstawie wzorców, autor wyeksponował rolę mini-generatora, jako podstawowej funkcjonalnej części koncepcji tworzenia muzyki bez określania jednoznacznej formy wynikowej kompozycji. Tak wykreowana sztuczna twórczość posiada znamiona muzyki tła, w której dopatrzyć się można teoretycznie nieskończonej struktury, ograniczonej wartościami parametrów systemu i jego możliwości sprzętowych. To nieograniczenie formy lokuje omawiany system w dziedzinie systemów komponowania raczej prymitywnych, w których zaszyta muzyczna "wiedza teoretyczna" jest minimalna. Z tego względu autor podjął próbę poszerzenia opisywanej wcześnie koncepcji komponowania o możliwość definiowania finalnej struktury formy kompozycji i jej elementów. Wysiłki i starania w kontekście wspomnianego rozwinięcia systemu zostały opisane jako analiza wpływu zmiany głównych parametrów systemu kontrolujących generowanie kompozycji na jej finalną postać.

Zgodnie z opisem koncepcji przedstawionym na stronie 85, na potrzeby eksperymentów zdefiniowane zostały 4-ry części kompozycji oznaczone kolejnymi liczbami całkowitymi. Podobnie jak w poprzednim rozdziale na stronie 97, autor zdefiniował poniżej następujące wartości początkowe parametrów systemu:

— $form = [1, 1, 3, 3, 1, 1, 4, 3, 3, 1]$ - wektor definiujący strukturę kompozycji (kolejność poszczególnych części kompozycji),
— $tempo = 160$ - tempo kompozycji wyrażone w jednostce bpm,
— $scale = [0, 2, 3, 5, 7, 8, 10, 12]$ - wektor opisujący skalę muzyczną, w której wygenerowana zostanie kompozycja (w tym przypadku jest to skala naturalna molowa),
— $possible_location = [0, 8]$ - wektor możliwych lokalizacji dźwięków we wzorcach muzycznych,
— $possible_location_snare = [0, 16]$ - j.w. z tym, że zdefiniowany dla konkretnego instrumentu (werbla),
— $possible_offset = [0]$ - wektor możliwych przesunięć dźwięków w czasie,
— $possible_offset_snare = [8]$ - j.w. z tym, że zdefiniowany dla werbla,
— $offset_mel = [0, 0.5]$ - wektor przesunięć dźwięków zdefiniowany dla melodii,
— $possible_duration_mel1 = [0.25]$ - wektor możliwych długości dźwięku dla melodii,
— $possible_duration_mel2 = [0.5]$ - alternatywny wektor możliwych długości dźwięku dla melodii,
— $possible_duration = [0.25, 0.5]$ - kolejny wektor możliwych długości dźwięku,
— $lp = 4$ - długość akordu wyrażona w jednostce bpm,
— $step_1 = 4$ - liczba akordów w ramach wzorca muzycznego,
— $step_2 = 8$ - alternatywna liczba akordów,
— $fill = [36, 50, 48, 47, 45, 43, 40, 49, 57, 66, 65, 40, 39]$ - pomocniczy wektor wykorzystany przez mini-generator do symulacji przejścia perkusyjnego.

W przedstawionej konfiguracji system generuje cztery rodzaje "fragmentów" kompozycji, przy czym nie wszystkie z nich muszą być wykorzystane w wynikowym utworze. Dodatkowo niektóre wartości parametrów, takie jak wektor długości dźwięku czy możliwych przesunięć, zostały zdefiniowane wielokrotnie w celu wprowadzenia urozmaiconych zbiorów wartości wykorzystanych przez różne mini-generatory. Domyślnym akordem dla generowania progresji akordów jest akord molowy $G_3\#$ (składający się z dźwięków: $G_3\#, B_3, D_4\#$), na podstawie którego budowane są kadencje w poszczególnych częściach kompozycji.

W przyjętej koncepcji model mini-generatora odgrywa kluczową rolę podczas budowania wzorca muzycznego dla instrumentów perkusyjnych. W tab. 4.9 przedstawiona została konfiguracja mini-generatorów zdefiniowanych w badanym systemie komponowania, a dla potrzeb przeprowadzonych eksperymentów autor zdefiniował następujące zmienne przedstawione na listingu poniżej (kod Matlab):

```
lpos = length(possible_location);
lpos_snare = length(possible_location_snare);
lof = length(possible_offset);
lof_snare = length(possible_offset_snare);
...
pl = possible_location(floor(rand(1)*lpos+1));
pls = possible_location_snare(floor(rand(1)*lpos_snare +1));
po = 0;
pos = possible_offset_snare(floor(rand(1)*lof_snare+1));

onset = [1+po:pl:size(notematrix,1)];
onset_s = [1+pos:pls:size(notematrix,1)];
onset_h1 = [1+64:pl:size(notematrix,1)];
onset_h2 = [1+po:pl/2:size(notematrix,1)];

fill = [36,50,48,47,45,43,40,49,57,66,65,40,39];
lfill = length(fill);
pm_h = fill(fix(rand(1,8)*lfill)+1);
```

Zmienne *pl* i *pls* określają wartości parametru *rhytmic_step* losowaną z wektorów lokalizacji dźwięków. Zmienne *onset*, *onset_s*, *onset_h*1 i *onseet_h*2 reprezentują wektory parametru $rhytmic_value$ - lokalizacji dźwięków w obrębie wzorca muzycznego. Mini-generatory Mg_5, Mg_7 i Mg_8 zostały zdefiniowane do budowania ciągów uderzeń w różne instrumenty perkusyjne (podane w konwencji $MIDI$ w wektorze $fill$), których strukturę opisuje zmienna *pm_h*.

Wraz z definicją mini-generatorów z tab. 4.9 zdefiniowana została architektura tworzenia klas wzorców dla poszczególnych części kompozycji muzycznej. Każdą część z osobna można traktować jako odrębną klasę, dla której generowane są wzorce muzyczne wg połączeń mini-generatorów przedstawionych na rys. 4.10.

Fragment przykładowej kompozycji $k_raw_421.flac$ zbudowanej na podstawie przedłożonej konfiguracji początkowej zaprezentowano na rys. 4.11. Wskazany wykres pianolowy został podzielony na obszary, w których występują fragmenty zdefiniowane wg parametru $form$. I tak, w pierwszym od lewej strony obszarze znajdują się dwa powtórzenia fragmentu 1-ego, w kolejnym dwa 2-ego, w trzecim następuje ponowne dwukrotne wystąpienie fragmentu 1-ego, natomiast w ostatnim obszarze znajduje się fragment 4-ty. Dodatkowo rys. 4.11 ukazuje różne długości wygenerowanych wzorców muzycznych: fragment 3-ci składa się ze wzorców zbudowanych na

Mini-generator	Klasa	*rhytmic_step*	*rhytmic_value*	PM	Kanał *MIDI*
Mg_1	D	pl	onset	36	10
Mg_2	D	pls	onset_s	40	10
Mg_3	D	pl	onset	42	10
Mg_4	D	pl	onset	51	10
Mg_5	D	pl	onset_h1	pm_h	10
Mg_6	D	pl	onset	49	10
Mg_7	D	pl/2	onset_h1	pm_h	10
Mg_8	D	pl/2	onset_h2	pm_h	10
Mg_9	D	pl	onset	54	10

Tab. 4.9. Parametry zdefiniowanych mini-generatorów. Opracowanie własne.

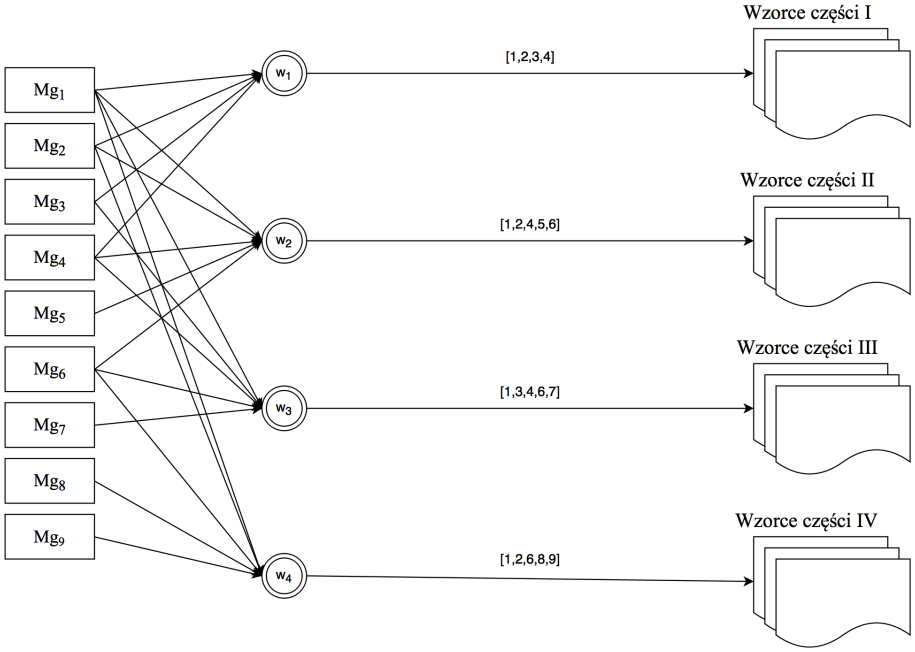
Kanał *MIDI*	Przydzielony instrument
1	Reed Organ
2	Ukulele
3	Ukulele
4	Ukulele
5	Rhodes Piano
6	Timpani
7	Nylon Guitar
8	Reed Organ
9	Fretless Bs.
10	Room 2 (zestaw perkusyjny)
11	Rhodes Piano
12	Room 2 (zestaw perkusyjny)

Tab. 4.10. Instrumenty wykorzystane w kompozycji $k_raw_421.flac$. Opracowanie własne.

podstawie 8-miu wystąpień akordów (parametr $step_2$), podczas gdy fragment 1-szy składa się ze wzorców zawierających 4-ry akordy (parametr $step_1$). W różnych odcieniach szarości przedstawione zostały również partie instrumentalne wykorzystane w kompozycji.

W tab. 4.10 wymienione zostały instrumenty przyporządkowane poszczególnym kanałom *MIDI* w kompozycji $k_raw_421.flac$. Aby wzbogacić rytmicznie muzykę generowaną przez system, dla kanału 10-ego i 12-ego autor przydzielił zestaw perkusyjny *Room*2. Synteza kompozycji została wykonana w aplikacji *SynthFont* z wykorzystaniem banku brzmień $Evanessence2.sf$, a wyniki tej syntezy można usłyszeć na płycie *CD* dołączonej do niniejszej pracy.

Kolejne eksperymenty mają na celu ukazać możliwości kreacji muzyki w odniesieniu do poszczególnych parametrów proponowanego systemu algorytmicznego komponowania muzyki. W celu zachowania podobieństwa do utworu referencyjnego (plik $k_raw_421.flac$), dalsze wyniki generowania będą syntezowane na podstawie konfiguracji z tab. 4.10.



Rys. 4.10. Schemat generowania architektury klas wzorców przez mini-generatory zdefiniowane w tab. 4.9. Opracowanie własne.

4.2.1. Modyfikacja wektora opisującego strukturę kompozycji

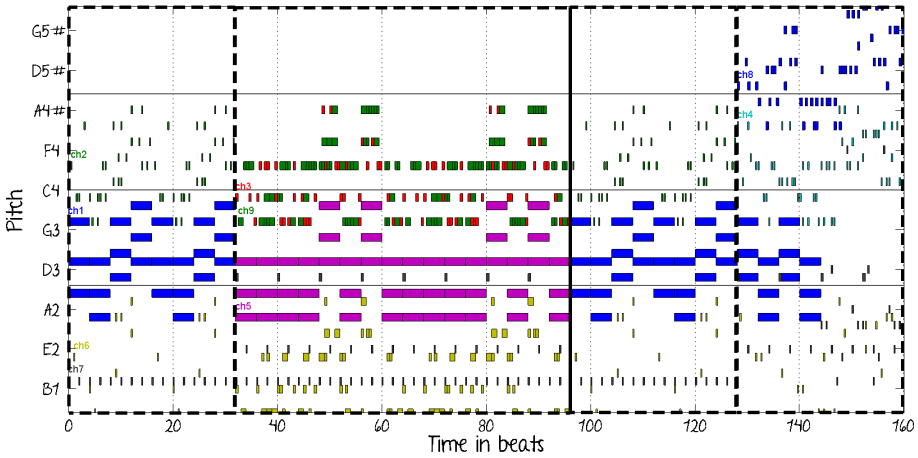
Wektor $form$	Nazwa pliku
$[1,1,1,1]$	$f_1111.flac$
$[4,4,4,4]$	$f_4444.flac$
$[1,2,3,4]$	$f_1234.flac$
$[1,2,3,4,4,3,2,1]$	$f_12344321.flac$

Tab. 4.11. Przyjęte struktury wektora $form$. Opracowanie własne.

Ogólną formę muzyczną kompozycji utworzonej za pomocą prezentowanego systemu opisuje wektor $form$ zawierający identyfikatory części kompozycji. W tab. tab : 421 przedstawiono różne jego struktury z przyporządkowanymi nazwami plików kompozycji. Wygenerowane wyniki zawierają zarówno sekwencje składające się wyłącznie z jednego rodzaju części ($f_1111.flac$ i $f_4444.flac$) jak i pojedynczych lub podwójnych ich powtórzeń ($f_1234.flac$ i $f_12344321.flac$). Autor zaleca zapoznanie się z utworzonymi kompozycjami, w celu lepszego zrozumienia wniosków płynących z dalszej analizy systemu.

Struktura wynikowych kompozycji z tab. 4.11 została umieszczona na wykresach pianolowych na rys. 4.12. Podobnie jak to miało miejsce wcześniej, różne partie instrumentalne ulokowane w poszczególnych kanałach *MIDI* zostały oznaczone różnymi odcieniami kolorystycznymi.

Faktem jest, że kompozycje te różnią się długością co wynika bezpośrednio ze struktury wektora $form$ i ustawień długości dźwięków muzycznych oraz akordów generowanych podczas tworzenia poszczególnych części. Wykresy 4.12A i 4.12B przedstawiają kompozycje, w których powtarzana jest 4-rokrotnie ta sama część kompozycji, natomiast 4.12C i 4.12D reprezentują struktury złożone ze wszystkich



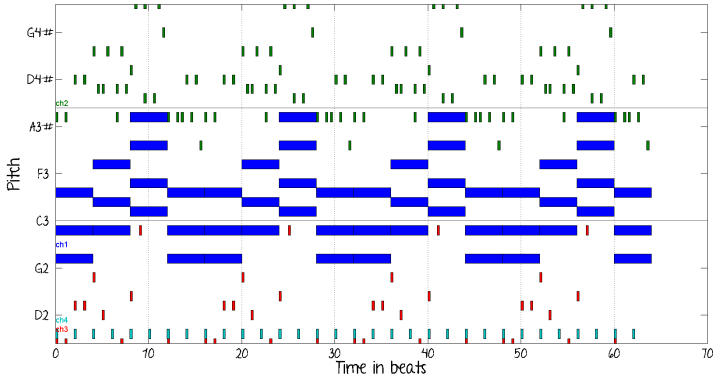
Rys. 4.11. Fragment kompozycji wygenerowanej na podstawie ustawień ze strony 113. Opracowanie własne.

4-ech zdefiniowanych w przykładzie części. Budową symetryczną odznacza się kompozycja $f_12344321.flac$ (rys. 4.12D), w której wyraźnie widać odbicie lustrzane wystąpień poszczególnych części kompozycji, względem części "4".

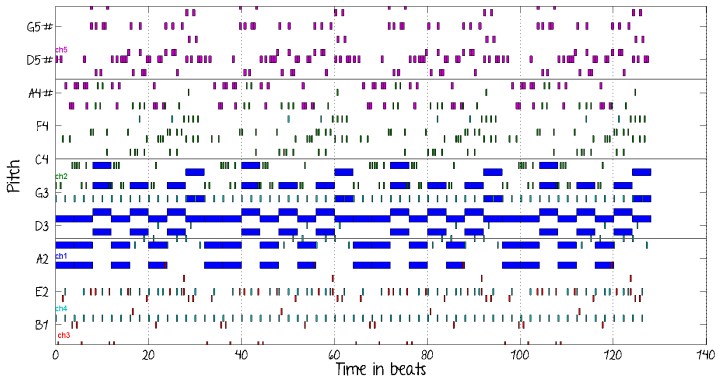
Zmiana struktury wektora $form$ niesie za sobą zmianę wystąpień poszczególnych definiowanych części kompozycji, jednak (jak można usłyszeć w przytoczonych utworach na płycie CD) wszystkie części bazują na jednym domyślnym akordzie. W ramach przedstawionych kompozycji progresje akordów przyjmują postać przedstawioną w tab. 4.12, w której struktura dźwięków poszczególnych akordów została podana wg. konwencji angielskiej (dźwięk H nie istnieje, natomiast diatonicznie po A występuje B).

Dla czytelności granice pomiędzy kolejnymi częściami kompozycji zostały oznaczone jako "$---x$", gdzie x oznacza numer części. Z analizy struktury kadencji występujących w przykładowych kompozycjach wynika, że w większości przypadków kadencja rozpoczyna się od zdefiniowanego bazowego akordu $G_3\#, B_3, D_4\#$. Sytuacja ta ma miejsce w przypadku części 1-ej, 2-ej oraz 4-ej natomiast w części 3-ciej kadencje rozpoczyna akord $C_3\#, E_3, G_3\#$. Ponadto, widać także, że przyjęty mechanizm tworzenia progresji zezwala na powtórzenia akordów, a także dopuszcza możliwość rozszerzenia danego akordu do postaci czterodźwięku (rys. 4.12 na str. 118 oraz tab. 4.12 na str. 119).

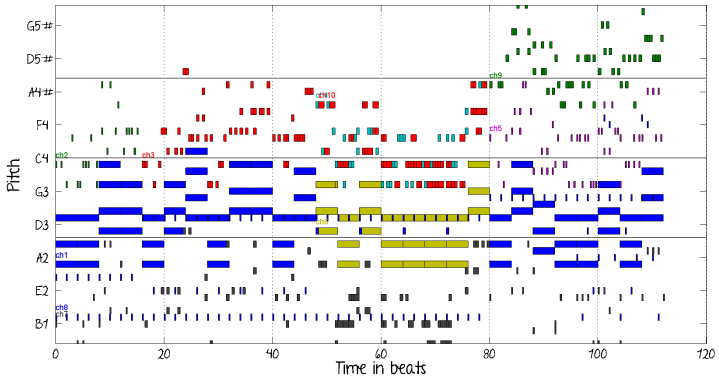
Zmiana wektora $form$ jest jedynie czynnikiem wpływającym na rozmieszczenie przygotowanych przez system części kompozycji. W dalszej części tego rozdziału autor przedstawił wyniki zmian parametrów systemu, które bezpośrednio wpływają na strukturę poszczególnych części kompozycji oraz wzorce muzyczne, z których jest tworzona.



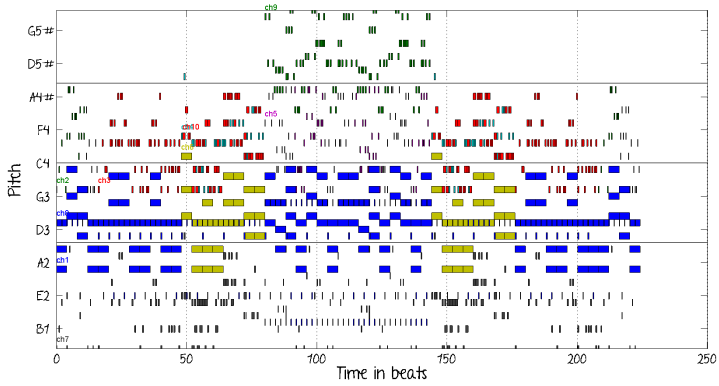
A



B



C



D

Rys. 4.12. Wykresy pianolowe kompozycji z tab. 4.11. A - $f_1111.flac$, B - $f_4444.flac$, C - $f_1234.flac$ i D - $f_12344321.flac$. Opracowanie własne.

Progresja (akordy)	Nazwa pliku
$[G2\#, B_2, D_3\# \rightarrow B_2, D_3, F_3\# \rightarrow$ $\rightarrow C_3\#, E_3, G_3\#, B_3 \rightarrow$ $\rightarrow G_2\#, B_2, D_3\#]$	$f_1111.flac$
$[G_2\#, B_2, D_3\# \rightarrow G_2\#, B_2, D_3\# \rightarrow$ $\rightarrow C_3\#, E_3, G_3\#, B_3 \rightarrow$ $\rightarrow G_2\#, B_2, D_3\# \rightarrow$ $\rightarrow C_3\#, E_3, G_3\# \rightarrow$ $\rightarrow G_2\#, B_2, D_3\# \rightarrow$ $\rightarrow C_3\#, E_3, G_3\# \rightarrow$ $\rightarrow D_3\#, F_3\#, A_3\#]$	$f_4444.flac$
$[G_2\#, B2, D_3\# \rightarrow G_2\#, B2, D_3\# \rightarrow$ $\rightarrow C_3\#, E_3, G_3\#, B_3 \rightarrow$ $\rightarrow C_3\#, E_3, G_3\# \rightarrow ---1$ $\rightarrow G_2\#, B_2, D_3\# \rightarrow$ $\rightarrow C_3\#, E_3, G_3\# \rightarrow$ $\rightarrow D_3\#, F_3\#, A_3\#, C_4\# \rightarrow$ $\rightarrow G_2\#, B_2, D_3\# \rightarrow$ $\rightarrow E_3, G_3, B_3 \rightarrow$ $\rightarrow E_3, G_3, B_3 \rightarrow$ $\rightarrow G_2\#, B_2, D_3\# \rightarrow$ $\rightarrow D_3\#, F_3\#, A_3\# \rightarrow ---2$ $\rightarrow C_3\#, E_3, G_3\# \rightarrow$ $\rightarrow G_2\#, B_2, D_3\# \rightarrow$ $\rightarrow C_3\#, E_3, G_3\# \rightarrow$ $\rightarrow G_2\#, B_2, D_3\# \rightarrow$ $\rightarrow G_2\#, B_2 D_3\# \rightarrow$ $\rightarrow G_2\#, B_2, D_3\# \rightarrow$ $\rightarrow G_2\#, B_2, D_3\# \rightarrow$ $\rightarrow E_3, G_3, B_3 \rightarrow ---3$ $\rightarrow G_2\#, B_2, D_3\# \rightarrow$ $\rightarrow E_3, G_3, B_3 \rightarrow$ $\rightarrow A_2\#, C_3\#, F_3 \rightarrow$ $\rightarrow G_2\#, B_2, D_3\# \rightarrow$ $\rightarrow G_2\#, B_2, D_3\# \rightarrow$ $\rightarrow C_3\#, E_3, G_3\# \rightarrow$ $\rightarrow G_2\#, B_2, D_3\# \rightarrow$ $\rightarrow D_3\#, F_3\#, A_3\# ---4]$	$f_1234.flac$
$[G_2\#, B2, D_3\# \rightarrow G_2\#, B2, D_3\# \rightarrow$ $\rightarrow C_3\#, E_3, G_3\#, B_3 \rightarrow$ $\rightarrow C_3\#, E_3, G_3\# \rightarrow ---1$ $\rightarrow G_2\#, B_2, D_3\# \rightarrow$ $\rightarrow C_3\#, E_3, G_3\# \rightarrow$ $\rightarrow D_3\#, F_3\#, A_3\#, C_4\# \rightarrow$ $\rightarrow G_2\#, B_2, D_3\# \rightarrow$ $\rightarrow E_3, G_3, B_3 \rightarrow$ $\rightarrow E_3, G_3, B_3 \rightarrow$ $\rightarrow G_2\#, B_2, D_3\# \rightarrow$ $\rightarrow D_3\#, F_3\#, A_3\# \rightarrow ---2$ $\rightarrow C_3\#, E_3, G_3\# \rightarrow$ $\rightarrow G_2\#, B_2, D_3\# \rightarrow$ $\rightarrow C_3\#, E_3, G_3\# \rightarrow$ $\rightarrow G_2\#, B_2, D_3\# \rightarrow$ $\rightarrow G_2\#, B_2 D_3\# \rightarrow$ $\rightarrow G_2\#, B_2, D_3\# \rightarrow$ $\rightarrow G_2\#, B_2, D_3\# \rightarrow$ $\rightarrow E_3, G_3, B_3 \rightarrow ---3$ $\rightarrow G_2\#, B_2, D_3\# \rightarrow$ $\rightarrow E_3, G_3, B_3 \rightarrow$ $\rightarrow A_2\#, C_3\#, F_3 \rightarrow$ $\rightarrow G_2\#, B_2, D_3\# \rightarrow$ $\rightarrow G_2\#, B_2, D_3\# \rightarrow$ $\rightarrow C_3\#, E_3, G_3\# \rightarrow$ $\rightarrow G_2\#, B_2, D_3\# \rightarrow$ $\rightarrow D_3\#, F_3\#, A_3\# \rightarrow ---4$ $\rightarrow G_2\#, B_2, D_3\# \rightarrow$ $\rightarrow E_3, G_3, B_3 \rightarrow$ $\rightarrow A_2\#, C_3\#, F_3 \rightarrow$ $\rightarrow G_2\#, B_2, D_3\# \rightarrow$ $\rightarrow G_2\#, B_2, D_3\# \rightarrow$ $\rightarrow C_3\#, E_3, G_3\# \rightarrow$ $\rightarrow G_2\#, B_2, D_3\# \rightarrow$ $\rightarrow D_3\#, F_3\#, A_3\# \rightarrow ---4$ $\rightarrow C_3\#, E_3, G_3\# \rightarrow$ $\rightarrow G_2\#, B_2, D_3\# \rightarrow$ $\rightarrow C_3\#, E_3, G_3\# \rightarrow$ $\rightarrow G_2\#, B_2, D_3\# \rightarrow$ $\rightarrow G_2\#, B_2 D_3\# \rightarrow$ $\rightarrow G_2\#, B_2, D_3\# \rightarrow$ $\rightarrow G_2\#, B_2, D_3\# \rightarrow$ $\rightarrow E_3, G_3, B_3 \rightarrow ---3$ $\rightarrow G_2\#, B_2, D_3\# \rightarrow$ $\rightarrow C_3\#, E_3, G_3\# \rightarrow$ $\rightarrow D_3\#, F_3\#, A_3\#, C_4\# \rightarrow$ $\rightarrow G_2\#, B_2, D_3\# \rightarrow$ $\rightarrow E_3, G_3, B_3 \rightarrow$ $\rightarrow E_3, G_3, B_3 \rightarrow$ $\rightarrow G_2\#, B_2, D_3\# \rightarrow$ $\rightarrow D_3\#, F_3\#, A_3\# \rightarrow ---2$ $\rightarrow G_2\#, B2, D_3\# \rightarrow$ $\rightarrow G_2\#, B2, D_3\# \rightarrow$ $\rightarrow C_3\#, E_3, G_3\#, B_3 \rightarrow$ $\rightarrow C_3\#, E_3, G_3\# \rightarrow ---1]$	$f_12344321.flac$

Tab. 4.12. Budowa progresji akordów dla poszczególnych kompozycji. Opracowanie własne.

4.2.2. Modyfikacja wektora skali muzycznej

Na akord bazowy, wokół którego budowane są progresje akordów występujące w kompozycji, bezpośrednio wpływa skala muzyczna przyjęta w postaci wektora wartości interwałów muzycznych pomiędzy kolejnymi stopniami skali muzycznej. Dla tak zdefiniowanego wektora wystarczy podać bezwzględną wartość (w konwencji systemu $MIDI$) wysokości dźwięku muzycznego, który ma być podstawowym dźwiękiem służącym budowie skali. W przyjętej konfiguracji testowej ze strony 113 jest to dźwięk $G_3\#$, na którym zbudowany został bazowy akord molowy zbudowany z dźwięków: $G_3\#$, B_3 (czyli interwał tercji małej) oraz $D_4\#$ (interwał kwinty). Na skali molowej zostały oparte wszystkie progresje akordów przedstawione w rozdziale 4.2.1 znajdujące się na dołączonej płycie CD. Autor w dalszej części pracy postara się przybliżyć różnice pomiędzy kompozycjami muzycznymi wygenerowanymi przez system komponowania na podstawie różnych wariantów wektora skali *scale*.

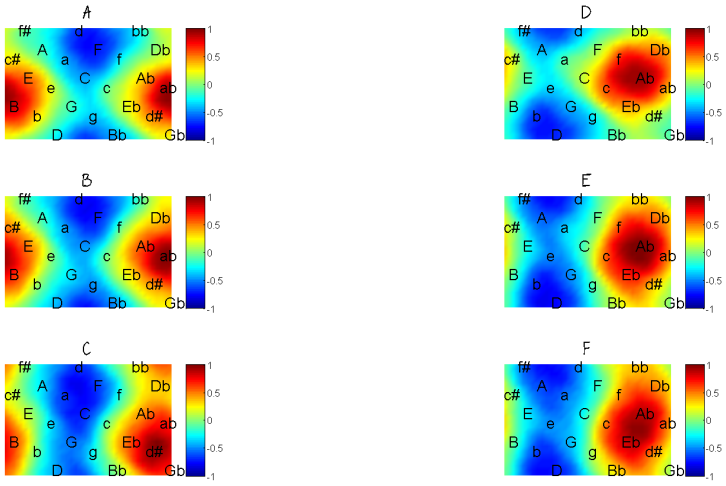
W eksperymencie zdefiniowano wektory 6-ciu popularnych skal muzycznych a ich budowę wraz z odpowiadającymi im nazwami wygenerowanych kompozycji, przedstawiono w tab. 4.13. Trzy pierwsze kompozycje zostały wygenerowane na podstawie skal molowych w następującej kolejności: molowa naturalna ($s_mol_nat.flac$), *molowa harmoniczna* ($s_mol_har.flac$) oraz *molowa dorycka* ($s_mol_dor.flac$). Kolejne trzy wygenerowano na podstawie skal durowych: *naturalnej* ($s_dur_nat.flac$) , *harmonicznej* ($s_dur_har.flac$) i *miękkiej* ($s_dur_soft.flac$). Dla wszystkich 6-ciu kompozycji dźwiękiem podstawowym (tonalnym) skali jest dźwięk $G_3\#$, a pozostałe parametry przyjmują wartości wg ustawień ze strony 113.

Analiza brzmienia przedstawionych w tab. 4.13 kompozycji pozwala ulokować wyniki utworzone przez system w oddzielnych grupach molowych i durowych. Można zauważyć, że w obrębie grupy odczucie wyraźnej różnicy w brzmieniu pomiędzy kompozycjami jest mniejsze, niż w przypadku kompozycji należących do różnych grup. Brzmienie progresji akordów w tonacjach durowych potocznie nazywane jest wesołym lub radosnym. Skale molowe natomiast odpowiadają za uwypuklanie bardziej smutnych emocji. Różnice w odczuwanych brzmieniach kompozycji molowych i durowych wywodzą się ze strukturalnych różnic pomiędzy histogramami klas wysokości dźwięków muzycznych, użytych w tych kompozycjach. Wizualizację tych różnic przedstawiono na rys. 4.13 w postaci samoorganizujących się map (z ang. *SOM - Self Organizing Map*) klas wysokości dźwięków ([95], [66]), dla których w procesie trenowania wykorzystano profile Krumhansl'a - Kessler'a [31].

Po lewej stronie rys. 4.13 zostały ulokowane mapy SOM dla kompozycji molowych, natomiast z prawej strony znajdują się mapy SOM kolejnych kompozycji durowych. W ramach map 4.13A, 4.13B i 4.13C tonalność kompozycji została

Wektor $form$	Nazwa pliku
$[0, 2, 3, 5, 7, 8, 10, 12]$	$s_mol_nat.flac$
$[0, 2, 3, 5, 7, 8, 11, 12]$	$s_mol_har.flac$
$[0, 2, 3, 5, 7, 9, 11, 12]$	$s_mol_dor.flac$
$[0, 2, 4, 5, 7, 9, 11, 12]$	$s_dur_nat.flac$
$[0, 2, 4, 5, 7, 8, 11, 12]$	$s_dur_har.flac$
$[0, 2, 4, 5, 7, 8, 10, 12]$	$s_dur_soft.flac$

Tab. 4.13. Przyjęte struktury wektora *scale*. Opracowanie własne.

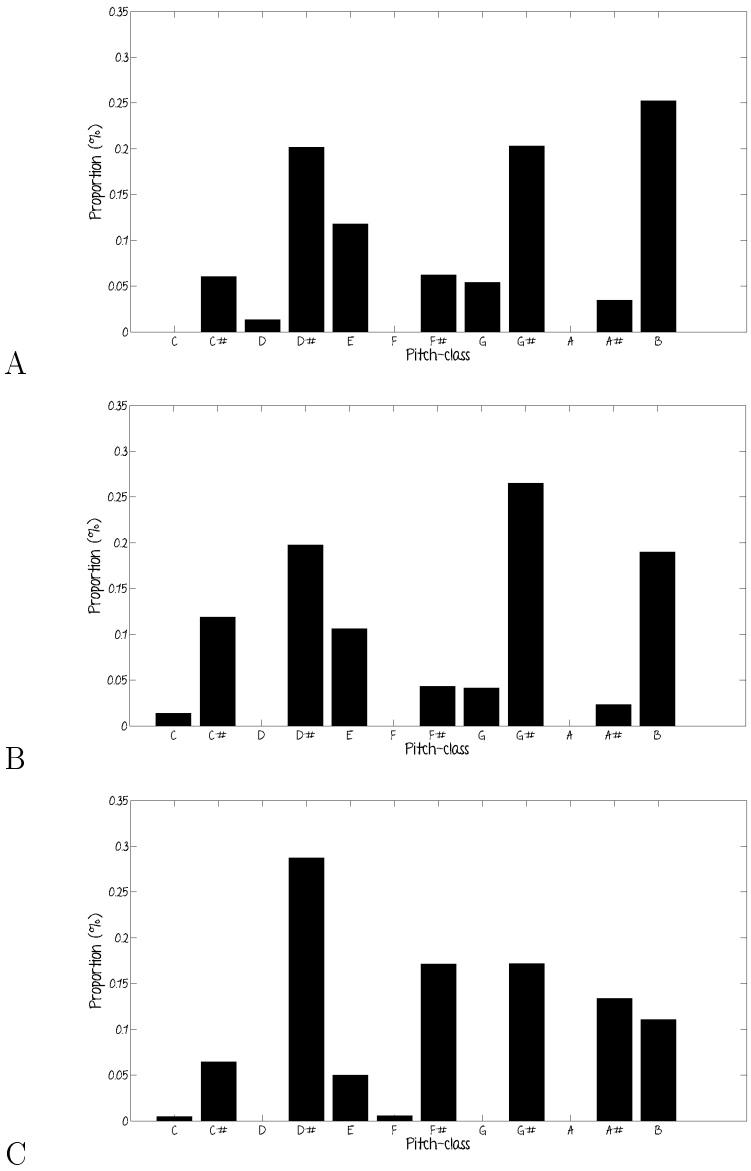


Rys. 4.13. Samoorganizujące się mapy klas wysokości dźwięków ([95], [66]) dla kompozycji *s_mol_nat.flac* (A), *s_mol_har.flac* (B), *s_mol_dor.flac* (C), *s_dur_nat.flac* (D), *s_dur_har.flac* (E) i *s_dur_soft.flac* (F). Opracowanie własne.

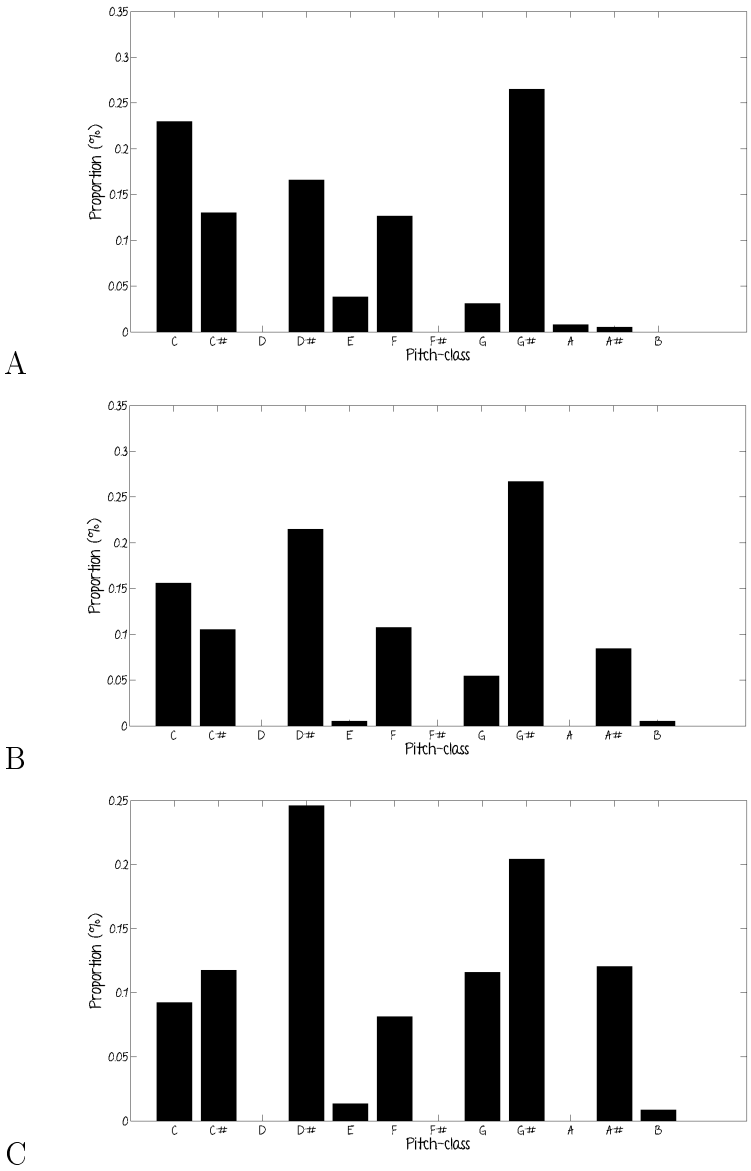
oszacowana wokół tonacji molowej $a\flat$ (czyli de-facto $g\#$), natomiast epicentra map 4.13D, 4.13E i 4.13F oscylują wokół tonacji durowej $A\flat$ czyli $G\#$. Prezentowane mapy potwierdzają możliwości wpływania na zmianę tonacji generowanej przez system kompozycji w postaci definiowania struktury wektora *scale*. Wizualizacje z rys. 4.13 ukazują również wyraźne rozgraniczenie pomiędzy kompozycjami molowymi i durowymi oraz względnie małe różnice pomiędzy kompozycjami należącymi do tej samej grupy. Przeprowadzone dalej bardziej szczegółowe analizy kompozycji wykazały konkretne różnice rozkładu klas wysokości dźwięków dla poszczególnych kompozycji.

Wizualizacje modelu Parncutt'a [86] dla kompozycji molowych (rys. 4.14) i durowych (rys. 4.15) wykazują pewne szczególne cechy rozkładu klas wysokości dźwięków. Zgodnie z oczekiwaniami, model wykazuje dużą wartość rozkładu centrum tonalnego, tj. dźwięku $G\#$, który stanowi podstawę dla progresji akordów generowanych kompozycji ($G_3\#$). Dla kompozycji molowych model wyznacza również wysokie wartości rozkładu dla dźwięków $D\#$ i B, wchodzących w skład przyjętego bazowego akordu molowy zbudowanego na dźwiękach $G_3\#$, B_3 oraz $D_4\#$.

Model Parncutt'a klas wysokości dźwięków dla kompozycji durowych (rys. 4.15) przedstawia względnie duże wartości proporcjonalnego rozkładu dla klas $G\#$, $D\#$ oraz C. Jest to spowodowane wykorzystaniem w procesie komponowania podstawowego akordu bazującego na dźwiękach $G_3\#$, $C4$ i $D_4\#$. Pewnym odstępstwem wykazuje się dystrybucja klas wysokości dla kompozycji *s_dur_soft.flac*, w której wartość proporcji klasy C nie wchodzi w skład trójki najwyższych wartości (tj. $G\#$, $D\#$ i C). W tym wypadku wyższe wartości w stosunku do klasy C wykazują $C\#$, G oraz $A\#$.



Rys. 4.14. Histogramy klas wysokości dźwięku wg modelu Parncutta'a [86] dla kompozycji *s_mol_nat.flac* (A), *s_mol_har.flac* (B) i *s_mol_dor.flac* (C). Opracowanie własne.



Rys. 4.15. Histogramy klas wysokości dźwięku wg modelu Parncutta'a [86] dla kompozycji *s_dur_nat.flac* (A), *s_dur_har.flac* (B) i *s_dur_soft.flac* (C). Opracowanie własne.

Przeprowadzone w tym rozdziale badania nad możliwością wpływania na tonację kompozycji potwierdziły zdolność systemu do generowania muzyki wg skali narzuconej przez użytkownika. Zastosowanie w badaniach jedynie 6-ciu popularnych skal muzycznych nie ogranicza możliwości algorytmicznego komponowania muzyki tonicznej, a jedynie ma na celu ukazanie prostoty sposobu definiowania tej funkcjonalności. W dalszej części pracy autor skupił się nad opisanie wpływu zmiany lokalizacji, przesunięcia i długości dźwięku muzycznego na kształt rytmicznej struktury kompozycji.

4.2.3. Modyfikacja wektorów lokalizacji dźwięków

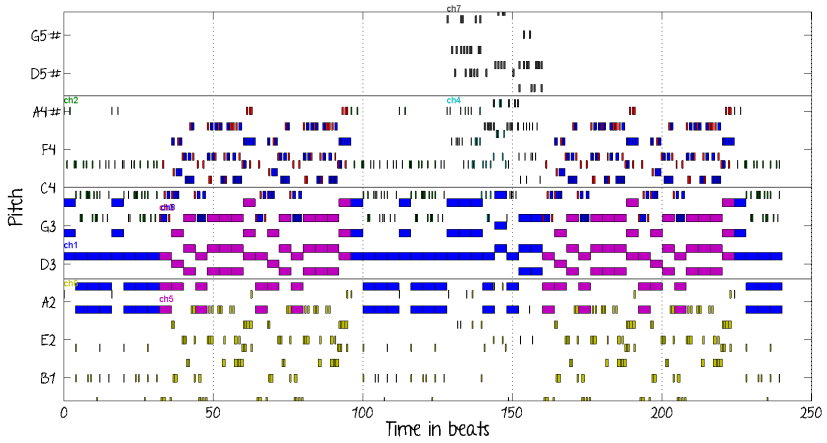
Wektor lokalizacji dźwięków jest jednym z parametrów wpływających na strukturę rytmiczną generowanej przez system kompozycji. Przechowuje on wartości dopuszczalnych odległości pomiędzy kolejnymi dźwiękami w ramach wzorca muzycznego utworzonego przez mini-generator. Jak opisano na stronie 113 w przyjętej przez autora koncepcji systemu algorytmicznego komponowania istnieje możliwość zdefiniowania większej liczby wektorów lokalizacji dźwięków. W tej części rozdziału dokonano analizy wyników funkcjonowania system algorytmicznego komponowania muzyki pod kątem możliwości zmiany położenia poszczególnych dźwięków muzycznych poprzez modyfikację struktury wektorów lokalizacji.

W trakcie badań autor przeprowadził modyfikacje wektora *possible_location*, którego struktury przedstawiono w tab. 4.14. Wektor *possible_location_snare* oraz inne zmienne zdefiniowane na stronie 114 pozostały w postaciach zdefiniowanych na stronie 113. Należy również dodać, że jedna ze ścieżek melodycznych generowanych przez system została opisana (niezależnie od mini-generatorów) jako partia instrumentów perkusyjnych (strona 115: dla 12-stego kanału $MIDI$ przypisano zestaw perkusyjny $Room2$ z banku brzmień $Evanessence2.sf$). W tym sensie kompozycje z tab. 4.14 zawierają partie perkusyjne utworzone zarówno za pomocą mini-generatorów, jak i funkcji generującej linię melodyczną.

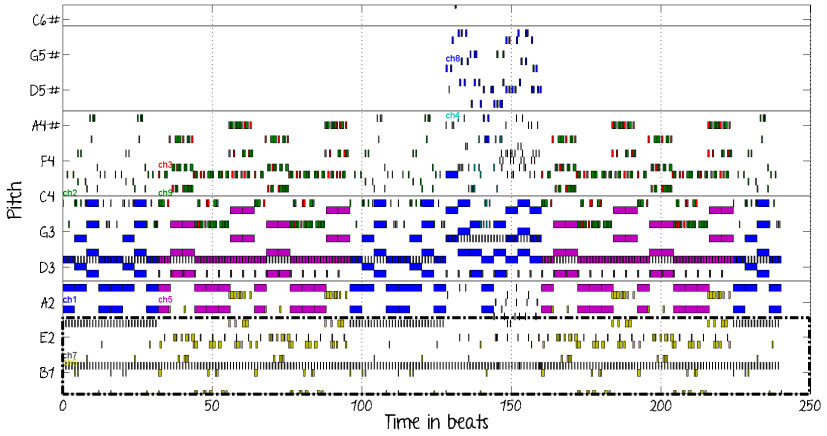
Struktury kompozycji z tab. 4.14 przedstawione zostały na rys. 4.16 w postaci wykresów pianolowych z zaznaczonymi obszarami widocznego występowania partii instrumentalnych wygenerowanych przez mini-generatory (rys. 4.16B i 4.16C). Po odsłuchaniu badanych kompozycji oraz analizie wykresów pianolowych, można zauważyć pewne "braki" występowania partii perkusyjnych w kompozycji $pl_0.flac$. Wiąże się to z przyjętą dla tej kompozycji strukturą parametru *possible_location*, który w tym wypadku jest skalarem o wartości 0. W związku z tym rozlokowanie dźwięków w ramach wzorca budowanego przez mini-generator jest niemożliwe, ponieważ fragment funkcji tworzący wektor rozmieszczeń dźwięków przyjmuje za krok tworzenia wektora wartość równą 0. W kompozycji $pl_0.flac$ nie słychać również werbla, dla którego zdefiniowano

Wektor *possible_location*	Nazwa pliku
[0]	$pl_0.flac$
[4]	$pl_4.flac$
$[0, 2, 4, 8]$	$pl_0248.flac$

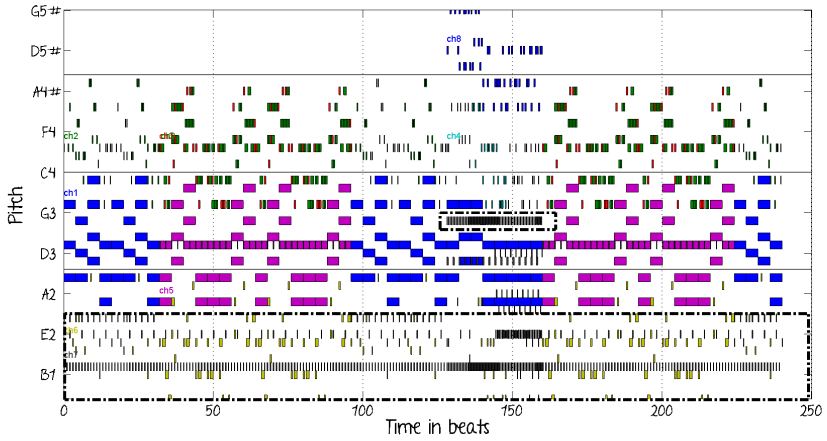
Tab. 4.14. Przyjęte struktury wektora *possible_location*. Opracowanie własne.



A



B



C

Rys. 4.16. Wykresy pianolowe kompozycji *pl_0.flac* (A), *pl_4.flac* (B) i *pl_0248.flac* (C) z zaznaczonymi widocznymi przebiegami partii wygenerowanych przez mini-generatory z tab. 4.9 na stronie 115. Opracowanie własne.

mini-generator przyjmujący jako parametr *rhytmic_step* wartość losowaną z wektora *possible_location_snare*, a który zgodnie z konfiguracją podaną na stronie 113 przyjmuje postać $possible_location_snare = [0, 16]$. Słychać więc, że w tym wypadku podczas losowania wartości dla mini-generatora tworzącego partię werbla wylosowana została wartość 0.

Inaczej wygląda struktura kompozycji *pl_4.flac* i *pl_0248.flac*. Dla pierwszej z nich zmienna *possible_location* przyjmuje wartość 4, tak więc mini-generatory bazujące na tej zmiennej produkują wzorce muzyczne, w których dźwięki wstawiane są co 4-ry elementy. Właściwość tę słychać w równym uderzenie w różne instrumenty perkusyjne, w tym samym momencie. Werbel natomiast bazuje na rozkładzie *possible_location_snare*, w którym słychać wyraźne uderzenie w na "3".

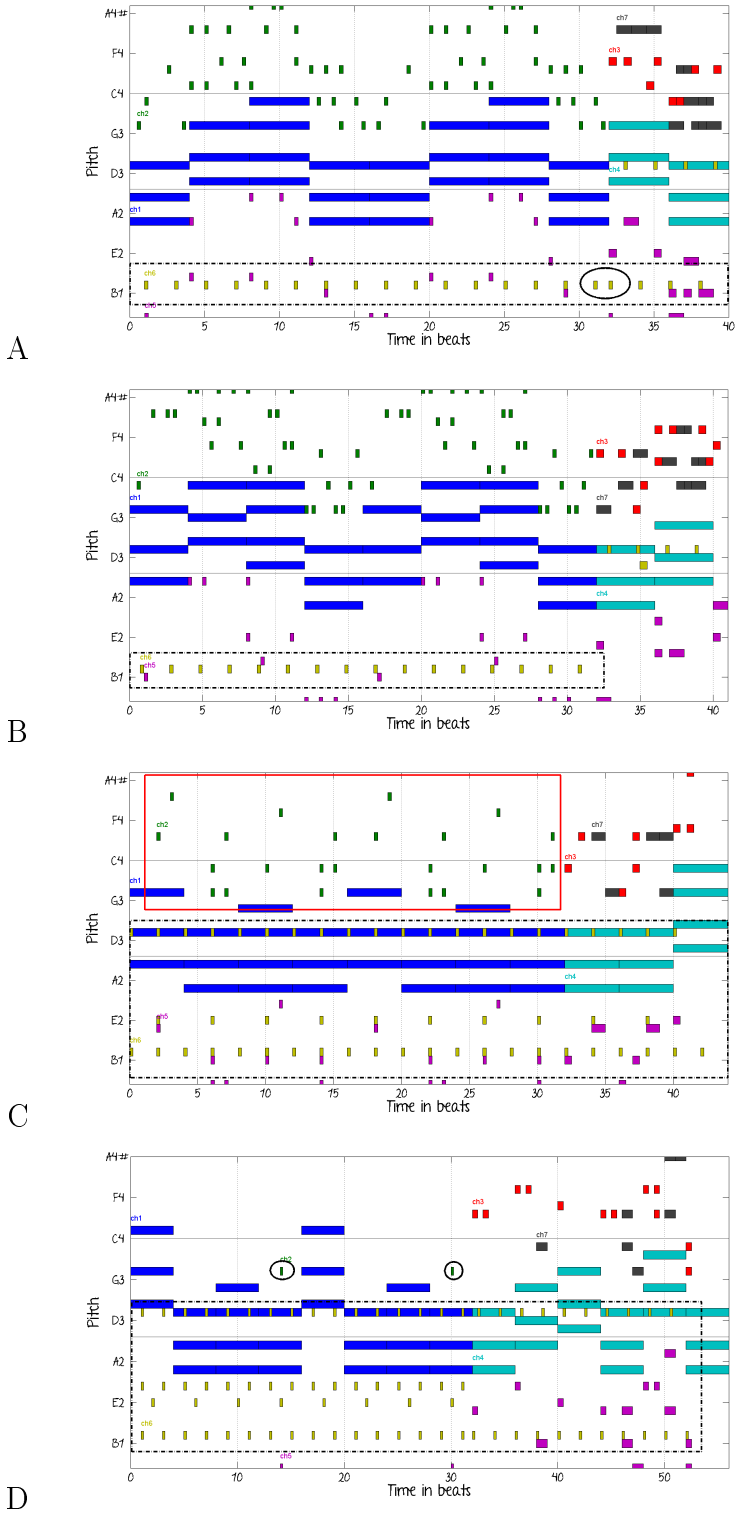
W kompozycji *pl_0248.flac* (rys. 4.16C) autor wykorzystał natomiast zmienną *possible_location* w postaci wektora, z którego mogą być wylosowane pozycje 0, 2, 4 lub 8. Zróżnicowanie losowanych dla mini-generatora wartości kroku wstawiania dźwięku prowadzi do tworzenia bardziej urozmaiconej struktury rytmicznej. Wyraźną kulminację partii perkusyjnej wygenerowanej przez mini-generatory widać (i słychać) w czasie ok. 150 bpm na rys. 4.16C, w którym następuje długie "przejście" (z ang. *fill*) do kolejnej części kompozycji.

Jak można było zaobserwować w przytoczonych kompozycjach, modyfikacja parametru odpowiedzialnego za rytmiczne rozmieszczenie dźwięków we wzorcu muzycznym pozwala kontrolować strukturę rytmiczną kompozycji (jeżeli tylko zdefiniowane zostały mini-generatory budujące odpowiednie partie instrumentalne). W prezentowanych przykładach mini-generatory generują partie instrumentów perkusyjnych, jednak nic nie stoi na przeszkodzie aby wykorzystać ten koncept do tworzenia pobocznych linii instrumentalnych, w których parametr PM stanowi motyw muzyczny nie wchodzący w dysonans z generowaną melodią oraz kadencją kompozycji. Przyjęta koncepcja systemu algorytmicznego komponowania muzyki zakłada również modyfikację struktury rytmicznej utworu za pomocą innego parametru jakim jest wektor przesunięć dźwięków wzorca muzycznego.

4.2.4. Modyfikacja wektorów przesunięć dźwięków

Drugim parametrem (obok wektora lokalizacji dźwięków) wpływającym na kształt struktury rytmicznej generowanej kompozycji jest wektor przesunięć dźwięków, składający się z wartości możliwych przesunięć (offsetów) w ramach wzorca muzycznego, progresji akordów czy linii melodycznej. W tab. 4.15 zaprezentowane zostały struktury wektorów $possible_offset$ i $offset_mel$ odpowiedzialne kolejno za możliwe przesunięcia w obrębie wybranych wzorców muzycznych (generowanych przez mini-generatory) oraz przesunięcia dźwięków podczas generowania linii melodycznej. W ostatniej kolumnie znajdują się nazwy plików muzycznych załączonych do niniejszej pracy na płycie CD. Pozostałe parametry systemu, podobnie jak w poprzednio przeprowadzonych eksperymentach, pozostały w niezmienionej postaci opisanej na stronie 113.

Odzwierciedleniem zmian kompozycji dokonanych przy definiowaniu wektorów $possible_offset$ i $offset_mel$ są wykresy pianolowe początkowych ich fragmentów przedstawione na rys. 4.17. Na poszczególnych wykresach zaznaczono dodatkowo obszary leżące w zainteresowaniu autora, a pozostające cechami charakterystycznymi, odróżniającymi daną kompozycję od innych.



Rys. 4.17. Wykresy pianolowe początkowych fragmentów kompozycji *po_4_om_005.flac* (A), *po_3_om_005.flac* (B), *po_0_om_2.flac* (C) i *po_024_om_24.flac* (D) z zaznaczeniem charakterystycznych cech struktur rytmicznych. Opracowanie własne.

Wektor $possible_offset$	Wektor $offset_mel$	Nazwa pliku
[4]	$[0, 0.5]$	$po_4_om_005.flac$
[3]	$[0, 0.5]$	$po_3_om_005.flac$
[0]	[2]	$po_0_om_2.flac$
$[0, 2, 4]$	$[2, 4]$	$po_024_om_24.flac$

Tab. 4.15. Przyjęte struktury wektorów $possible_offset$ i $offset_mel$. Opracowanie własne.

Analizę muzyczną rozpoczęto od kompozycji $po_4_om_005.flac$, której początkowe fragmenty zaprezentowano na wykresie 4.17A. Przerywaną linią zaznaczony został obszar występowania charakterystycznego motywu wygenerowanego przez mini-generator Mg_1 (zdefiniowanego w tab. 4.9 na stronie 115) i odgrywanego za pomocą bębna centralnego (instrument o numerze 36 w próbce brzmień zestawu Room 2 - tab. 4.10 na stronie 115). W badanym obszarze słychać wyraźne przesunięcie względem linii melodycznej (górny fragment wykresu), które następnie zostało niwelowane w przejściu z 1-ej do 3-ciej części kompozycji. Zmiana rytmiczna zaznaczona została na rys. 4.17A za pomocą elipsy, w której ostatni dźwięk generowany przez bęben centralny w 1-ej części poprzedza nowy przebieg wygenerowany za pomocą innej reguły dla części 3-ciej kompozycji. Pomimo wyraźnie słyszalnego offsetu pomiędzy wymienionymi fragmentami, struktura rytmiczna pozostaje w zgodzie z ogólnym przebiegiem kompozycji.

Sytuacja zgoła odmienna występuje w kompozycji $po_3_om_005.flac$, w której wartość przesunięcia wektora $possible_offset$ wynosi 3 elementy, a dotyczy rozmieszczenia dźwięków we wzorcu muzycznym za pomocą wektorów $onset$ i $onset_h2$ przedstawionych na stronie 114. Autor celowo dokonał przesunięcia o wskazaną wartość, aby pokazać w jaki sposób za pomocą względnie mało różniących się wartości rytmicznych przesunięć, można otrzymać wyraźnie odbiegające od siebie pod względem poprawności rytmicznej kompozycje. Ta nierytmiczność objawia się nie tylko w partii generowanej przez mini-generator Mg_1 (oznaczonej na wykresie 4.17B linią przerywaną), ale również w dalszych przebiegach generowanych na podstawie wektorów $onset$ oraz $onset_h2$.

Kolejnym, tym razem poprawnym rytmicznie wariantem kompozycji opisanym przez autora jest struktura muzyczna zapisana w pliku $po_0_om_2.flac$, której fragment przedstawia rys. 4.17C. Szczególną cechą tej formy muzycznej jest rytmiczność linii melodycznej, zaznaczonej za pomocą linii przerywanej w górnej części wykresu 4.17C. W odróżnieniu od poprzednio opisanych kompozycji, w tym wariancie dokonano modyfikacji wektora $offset_mel$, wykorzystywanego do generowania linii melodycznych, natomiast parametr $possible_offset$ przyjął domyślną wartość ze strony 113 równą 0. Takie ustawienie spowodowało wykreowanie formy, w której linie melodyczne występujące w różnych zdefiniowanych częściach kompozycji tworzą powtarzalne struktury rytmiczne przesunięte o 2 uderzenia (jednostki bpm). Stanowi to różnicę w interpretacji wartości przez różne części systemu. W mini-generatorach wartość przesunięcia traktowana jest jako liczba elementów w wektorze, natomiast funkcja generujące linię melodyczną bazuje na względnych wartościach rytmicznych (względem tempa utworu) wyrażonych w uderzeniach na minutę.

Parametr *possible_duration_mel*1	Parametr *possible_duration_mel*2	Nazwa pliku
[0.0125]	[0.5]	$pdm1_00125_pdm2_05.flac$
[0.25]	[0.0125]	$pdm1_025_pdm2_00125.flac$
[2]	[0.5]	$pdm1_2_pdm2_05.flac$
[0.25, 2, 4]	[0.5]	$pdm1_02524_pdm2_05.flac$

Tab. 4.16. Przyjęte struktury parametrów *possible_duration_mel*1 i *possible_duration_mel*2. Opracowanie własne.

W kulminacyjnym punkcie opisywanych badań autor dokonał modyfikacji zarówno parametru *possible_offset* jak i *offset_mel*, czego przejawem jest kompozycja przedstawiona na rys. 4.17*D*. Na podanym wykresie pianolowym linią przerywaną zaznaczono obszar występowania partii perkusyjnych wygenerowanych przez mini-generatory na podstawie parametru $possible_offset = [0, 2, 4]$, natomiast elipsami zaznaczono dźwięku rozlokowane w linii melodycznej według parametru $offset_mel = [2, 4]$. W 1-ej części kompozycji bęben centralny odgrywa swoją partię równo z innym instrumentem (*Closed Hi-Hat*) generowanym przez mini-generator Mg_3. Rytmicznie pomiędzy wymienione partie wstawiony został przebieg werbla, generowany przez mini-generator Mg_2, co w całości utworzyło ciekawy schemat perkusyjny.

Podobnie przebieg linii melodycznych tworzonych w części 1-ej i 3-ciej utworzył ciekawą strukturę zbudowaną na podstawie przesunięć o 2 i 4 jednostki *bpm*. Rytmicznie linia melodyczna z 3-ciej części kompozycji $po_024_om_24.flac$ przypomina przebieg z 1-ej części kompozycji $po_0_om_2.flac$. Podobieństwa i różnice pomiędzy przedstawionymi w tym rozdziale kompozycjami wynikają z różnej interpretacji wartości parametrów przesunięć dźwięków. W kolejnym rozdziale autor postara się opisać możliwości zmiany kolejnego aspektu rytmiki kompozycji jakim jest długość dźwięków muzycznych.

4.2.5. Modyfikacja wektorów długości dźwięków

Lokalizacja dźwięków, formowana zarówno poprzez zmianę parametru lokalizacji jak i wprowadzanie przesunięć w postaci przedstawionej w poprzednim rozdziale wpływa istotnie na przebieg rytmiczny kompozycji muzycznej. Innym istotnym aspektem kształtującym charakter rytmiczny utworu jest długość dźwięków muzycznych tworzących poszczególne elementy dzieła. Utwór obdarzony dźwiękami przewlekłymi w czasie, choćby rozlokowanymi dość gęsto, będzie miał charakter bardziej spokojny niż kompozycja naszpikowana względnie krótkimi dźwiękami odgrywanymi w sposób "szarpany" (tzn. z artykulacją np. typu *staccato*). To właśnie na charakter artykulacji utworu ma wpływ modyfikacja parametru długości dźwięków muzycznych.

W tej części pracy omówiony został aspekt zmiany artykulacji dźwięków poprzez modyfikowanie parametrów systemu odpowiedzialnych za długość dźwięków. W tab. 4.16 przedstawiono 4-ry różne konfiguracje parametrów *possible_duration_mel*1 i *possible_duration_mel*2, wpływających na artykulację linii melodycznych generowanych przez opisywany system komponowania muzyki. Podobnie jak w poprzednich

eksperymentach, ich wyniki zostały zapisane na dołączonej do pracy płycie CD w postaci plików audio, których nazwy podano w ostatniej kolumnie tab. 4.16.

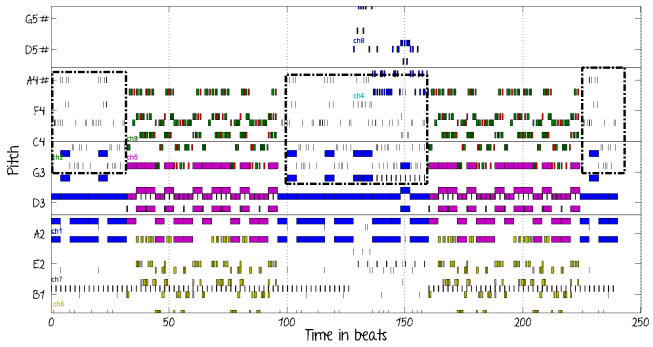
Efekty modyfikowania wymienionych parametrów zostały przedstawione na rys. 4.18 jako wykresy pianolowe wygenerowanych kompozycji. Dodatkowo obszary, w których usłyszeć można wyraźny wpływ parametrów zmieniających długości dźwięków, oznaczone zostały za pomocą linii przerywanej.

Analizę zmiany przebiegu linii melodycznych rozpoczęto od kompozycji $pdm1_00125_pdm2_05.flac$, której wykres pianolowy znajduje się na rys. 4.18A. Parametr $possible_duration_mel1$ przyjmuje w tym ustawieniu wartość 0.0125 bpm, co skutkuje generowaniem linii melodycznych o krótko brzmiących dźwiękach. Partie te są widoczne w obszarach części 1-ej i 4-ej kompozycji (zaznaczone obszary na rys. 4.18A), w których to fragmentach artykulacja dźwięków została zmieniona w stosunku do części 2-iej i 3-ciej. Ciekawy jest jednak fakt, że pod względem brzmienia obszary zaznaczone nie odbiegają znanie od reszty kompozycji. Może być to związane z doborem instrumentu odgrywającego tę partię melodii (dla przypomnienia - konfiguracja wybranych przy syntezie utworu instrumentów znajduje się w tab. 4.10 na stronie 115). Autor wnioskuje zatem, że barwa wybranego instrumentu nie pozwala zaobserwować wyraźnej różnicy w wybrzmiewaniu pomiędzy dźwiękami o długości 0.0125, a 0.5 bpm.

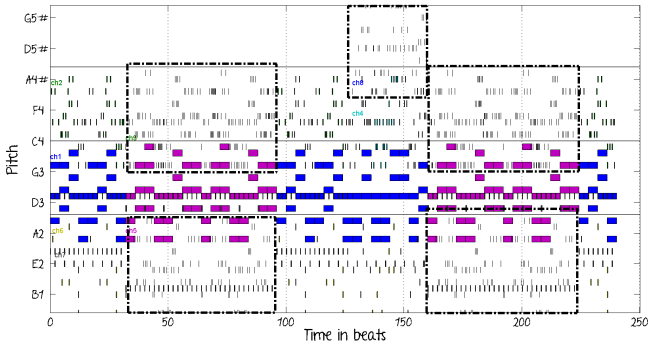
Podobnie sytuacja wygląda dla kompozycji $pdm1_025_pdm2_00125.flac$ z rys. 4.18B, w której parametr $possible_duration_mel1$ przyjmuje wartość 0.25 bpm, natomiast wartość dla $possible_duration_mel2$ wynosi 0.0125 bpm. Tym razem modyfikacjom uległy części 3-cia i 4-ta, w których charakterystyczne obszary zaznaczono za pomocą linii przerywanej. Różnicą jest jednak subiektywne odczucie zwiększonej "gęstości upakowania" dźwięków melodii w części 3-ciej w porównaniu do rozlokowania dźwięków części 1-ej. Jest to związane z dwukrotną długością wzorców muzycznych wykorzystanych w części 3, w stosunku do części 1-ej. W tej pierwszej kadencja zbudowana jest na bazie 8-miu akordów, natomiast w części 1-ej progresja zawiera 4-ry akordy. Cechy te szerzej zostały omówione w kolejnej części tego rozdziału i nie stanowią *clou* bieżąco opisywanych badań.

Na uwagę zasługuje natomiast kolejna kompozycja o nazwie $pdm1_2_pdm2_05.flac$, której wykres pianolowy przedstawiono na rys. 4.18C. W tym utworze rozróżnienie pomiędzy długością dźwięków i ich artykulacją wydaje się być bardziej wyraźnym. Różnice pomiędzy długościami dźwięków oznaczono w przykładowym fragmencie za pomocą linii przerywanej (rys. 4.18C). Dźwięki odgrywane w części 1-ej są dłuższe niż w poprzednich kompozycjach za sprawą wartości parametru $possible_duration_mel1$ równej 2. Linie melodyczne generowane za pomocą tego ustawienia wydają się być mniej "ruchliwe" w stosunku do krótkich dźwięków z przebiegów kompozycji $pdm1_00125_pdm2_05.flac$ czy $pdm1_025_pdm2_00125.flac$.

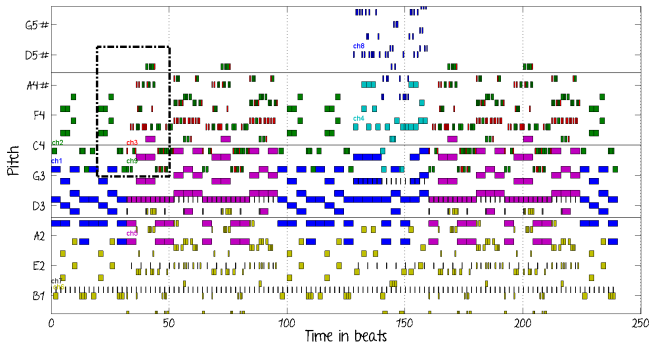
Podobny charakter wykazuje kompozycja $pdm1_02524_pdm2_05.flac$, której wykres pianolowy widnieje na rys. 4.18D. Dźwięki rozmieszczone w części 1-ej tym razem posiadają różne długości (w porównaniu z poprzednio badanymi kompozycjami) ze względu na parametr $possible_duration_mel1$, który przyjął postać wektora wartości $[0.25, 2, 4]$. Te charakterystyczne rozróżnienia długości są obserwowane nie tylko w głównej linii melodycznej odtwarzanej przez instrument *Ukulele* (tab. 4.10 na stronie 115), ale także w linii basowej powstałej przez fragmentację linii melodycznej, a syntezowanej za pomocą instrumentu *Fretless Bs*.



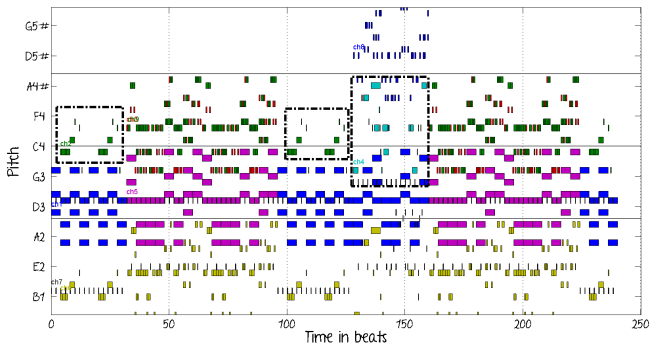
A



B



C



D

Rys. 4.18. Wykresy pianolowe kompozycji *pdm*1_00125_*pdm*2_05.*flac* (A), *pdm*1_025_*pdm*2_00125.*flac* (B), *pdm*1_2_*pdm*2_05.*flac* (C) i *pdm*1_02524_*pdm*2_05.*flac* (D) z zaznaczeniem charakterystycznych cech struktur rytmicznych. Opracowanie własne.

Autor opisując wyniki modyfikacji przykładowych parametrów długości dźwięków muzycznych miał na myśli uwypuklenie funkcjonalności systemu algorytmicznego komponowania muzyki w aspekcie możliwości wpływania na artykulację dźwięków muzycznych za pomocą zmian ich wybrzmiewania. Dobór wartości oraz instrumentów ma w dalszym ciągu charakter subiektywny, obiektywnie natomiast odpowiednio dobrane wartości mogą przynieść zamierzony i w istocie przewidywalny efekt końcowy. W dalszych rozważaniach ukazany został ostatni wymiar generowania kompozycji muzycznej, w której użytkownik ma wpływ na budowę kadencji muzycznej w obrębie zdefiniowanej części utworu.

4.2.6. Modyfikacja liczby i czasu trwania akordów

W ujęciu algorytmicznego komponowania muzyki przedstawionym przez autora w rozdziale poświęconym koncepcji tworzenia progresji akordów (rys. 2.7 na stronie 69) opisana została funkcjonalność systemu odpowiedzialna za tworzenie trzonu kompozycji, do którego następnie, obok przebiegów partii generowanych przez mini-generatory, dokładane są linie melodyczne i basowe. Struktura progresji akordów bazująca na utworzonej wcześnie progresji odgrywa kluczową rolę w procesie tworzenia całej kompozycji. W tym rozdziale autor postara się przedstawić sposób wpływania na różnorodność formy generowanej muzyki za pośrednictwem modyfikacji liczby i czasu trwania akordów budujących zdefiniowaną część kompozycji.

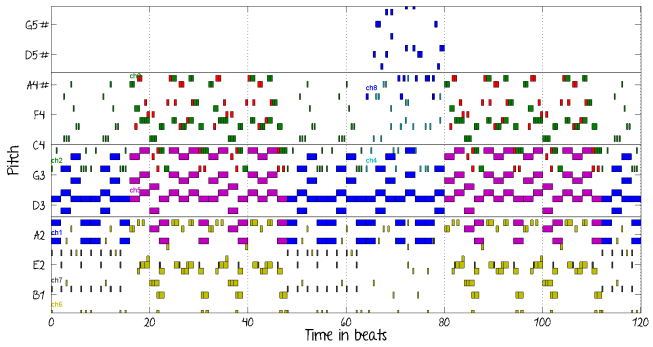
W tab. 4.17 znajdują się przyjęte przykładowe wartości parametrów odpowiedzialnych za czas trwania i liczbę akordów w poszczególnych częściach kompozycji. Parametry: $lp1$, $lp2$, $lp3$ oraz $lp4$ odpowiadają za czas trwania pojedynczego akordu odpowiednio w częściach kompozycji 1, 2, 3 i 4. Parametry: $step1$, $step2$, $step3$ oraz $step4$ odpowiedzialne są natomiast za liczbę akordów w kolejnych częściach kompozycji. W ostatniej kolumnie tab. 4.17 znajdują się nazwy plików muzycznych utworzonych przez system na podstawie zmodyfikowanych parametrów.

Ze względu na różne konfiguracje parametrów, wynikowe kompozycje muzyczne znacząco różnią się pod względem struktury (i długości), a ich kształt został przedstawiony na rys. 4.19. Podobne długości posiadają kompozycje $lp_2222_s_4888.flac$ i $lp_4444_s_2444.flac$ zaprezentowane na rys. 4.19A i 4.19C. Zdecydowanie dłuższe kompozycje $lp_8888_s_4888.flac$ (rys. 4.19B) i $lp_4444_s_8121212.flac$ (rys. 4.19D) charakteryzują się bardziej złożoną strukturą muzyczną. Warto jednak nieco bliżej przyjrzeć się cechą szczególnym każdego utworu z osobna.

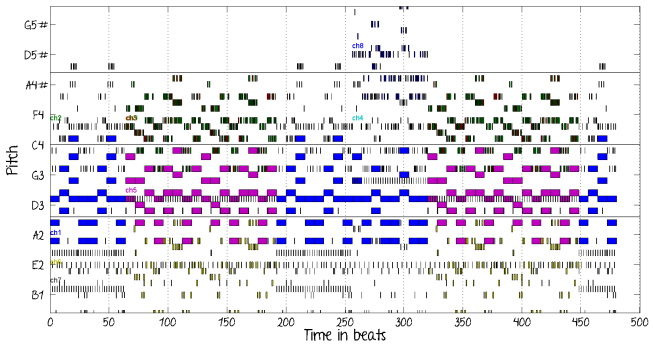
W kompozycji $lp_2222_s_4888.flac$ (rys. 4.19A) parametry liczby akordów wchodzących w skład zdefiniowanych części kompozycji pozostały przy wartościach

Par. $lp1$	Par. $lp2$	Par. $lp3$	Par. $lp4$	Par. $step1$	Par. $step2$	Par. $step3$	Par. $step4$	Nazwa pliku
2	2	2	2	4	8	8	8	$lp_2222_s_4888.flac$
8	8	8	8	4	8	8	8	$lp_8888_s_4888.flac$
4	4	4	4	2	4	4	4	$lp_4444_s_2444.flac$
4	4	4	4	8	12	12	12	$lp_4444_s_8121212.flac$

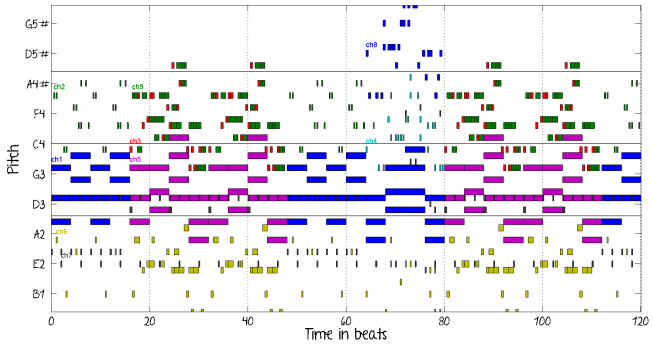
Tab. 4.17. Przyjęte wartości parametrów $lp1$, $lp2$, $lp3$, $lp4$, $step1$, $step2$, $step3$ oraz $step4$. Opracowanie własne



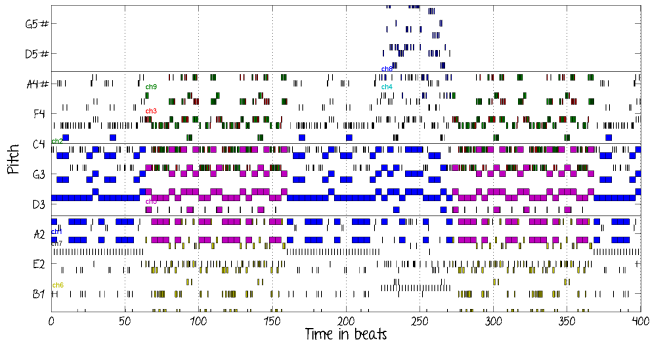
A



B



C



D

Rys. 4.19. Wykresy pianolowe kompozycji *lp_2222_s_4888.flac* (A), *lp_8888_s_4888.flac* (B), *lp_4444_s_2444.flac* (C) i *lp_4444_s_8121212.flac* (D). Opracowanie własne.

domyślnych (podanych na stronie 113), natomiast dwukrotnie skrócono długość akordów do wartości 2 *bpm*. Podczas odsłuchiwania kompozycji można zauważyć szybką zmienność melodii oraz harmonii, szczególnie uwidocznioną w częściach zbudowanych z 8-miu akordów. Skrócenie długości akordu, ze względu na specyfikę generowania melodii (koncepcja tego procesu została opisana na stronie 69), spowodowało zdegradowanie (uproszczenie) jej struktury.

Przy takich samych ustawieniach parametrów liczby akordów, kompozycja $lp_8888_s_4888.flac$ (rys. 4.19B)charakteryzuje się akordami czterokrotnie dłuższymi. Odwrotnie jak w poprzednio przedstawionym utworze, linie melodyczne są bardziej rozbudowane i "przeciągnięte" w czasie. W częściach zbudowanych z 8-miu akordów różnica ta wydaje się być jeszcze większa, a utworzone progresje tworzą z przebiegiem linii melodycznej i basowej ciekawe współbrzmienie, z różnie akcentowanymi fragmentami taktów.

Kompozycja $lp_4444_s_2444.flac$ (rys. 4.19C) cechuje się akordami o stałej długości 4 *bpm* jednak skróconych, w stosunku do poprzednio prezentowanych utworach, progresji akordów. W 1-ej części kompozycji mamy do czynienia z progresją bazującą na dwóch akordach, natomiast pozostałe części tworzą progresje 4-ro akordowe. Podobnie jak w kompozycji $lp_2222_s_4888.flac$ można tu usłyszeć zmienność formy, szczególnie w początkowej i końcowej fazie utworu.

Zgoła odmienną strukturę posiada kompozycja $lp_4444_s_8121212.flac$ (rys. 4.19D), gdzie progresje akordów składają się z 8-miu i 12-stu elementów. W środkowej części kompozycji cechą charakterystyczną jest powtórzona progresja 12-akordowa z dodatkową partią instrumentów perkusyjnych. Zgodnie z oczekiwaniami obok rozległych następujących po sobie struktur akordów, przebiegają linie melodyczne tworzone na bazie zmian akordów podyktowanych macierzą tranzycji generowania akordów, której koncepcję przedstawiono na stronie 69.

Jak pokazują wyniki przeprowadzonych w tym rozdziale badań, modyfikacja liczby i długości akordów w poszczególnych częściach kompozycji niesie ze sobą zmiany w złożoności linii melodycznych. Zmianie ulega również charakter utworu, który przeistacza się z utworu szybkozmiennego (dla małych wartości parametrów *lp* i *step*) do rozległych form muzycznych, w których usłyszeć można rozbudowane motywy melodyczne, współgrające rytmicznie i harmonicznie z resztą struktury utworu.

4.2.7. Wnioski z eksperymentów

Przedstawione w tym rozdziale wyniki badań nad możliwościami systemu algorytmicznego komponowania muzyki na podstawie wzorców muzycznych i opisu formy utworu, pokazują odmienny charakter tworzonej muzyki, w porównaniu z systemem bazującym wyłącznie na mini-generatorach, a którego analizy dokonano w rozdziale 4.1. Wyodrębnionych 6 grup parametrów sterujących pracą systemu stanowi jedynie podstawę możliwości modyfikowania końcowej formy kompozycji.

Modyfikacja struktury wektora formy razem ze zdefiniowaniem algorytmu generującego poszczególne części kompozycji to podstawowe narzędzia kreujące finalną postać dzieła muzycznego utworzonego przez proponowany system. Zmiana kolejności elementów formy może być pomocna w eksploracji ciekawych i nieszablonowych wyników, a sposób definiowania części kompozycji wprowadza szerokie możliwości kreowania dużych fragmentów formy.

Istotnym elementem kształtującym cześć harmoniczną generowanego utworu jest parametr opisujący skalę muzyczną oraz akord bazowy, na których zbudowane są

sekwencje akordów w kompozycji wynikowej. Wynika to ze specyfiki przyjętej koncepcji generowania linii melodycznych oraz linii basowych, które niejako "śledzą" przebieg sekwencji akordu, a budulcem dla nich jest budowa poszczególnych elementów tej progresji. Zatem z pomocą odpowiednio dobranych skali oraz akordu bazowego, można zbudować utwór np. o charakterze smutnym (w tonacjach molowych) lub bardziej radosnym (w tonacjach durowych).

Aspekt rytmiczny jest obok harmonii i melodyki jedną z najważniejszych cech generowanego utworu, dlatego też w przyjętej koncepcji zdefiniowane zostały odpowiednie parametry, pozwalające zmieniać jego strukturę rytmiczną. Modyfikacjom rytmicznym mogą podlegać zarówno poszczególne dźwięki (poprzez zmiany parametrów długości i przesunięć dźwięków w obrębie wzorca muzycznego lub części kompozycji) jak i całe fragmenty muzyczne, których charakterystykę można kształtować za pomocą odpowiednio dobranych wartości liczby i czasu trwania akordów. Zmiana wymienionych parametrów prowadzić może również do zmiany artykulacji dźwięków, co w połączeniu z ciekawym brzemieniem instrumentu syntezującego daną partię, może dać ciekawe efekty końcowe.

Parametry zdefiniowane w systemie w różnym stopniu pokrywają domeny muzyki związane z procesem komponowania utworu. Autor prezentując wyniki swoich badań miał na celu jedynie przedstawienie określonych technik sterowania tym procesem w obrębie koncepcji zaproponowanego w rozdziale 2.3. systemu. W ostatniej części tej pracy zostanie podjęta próba porównania dwóch badanych koncepcji systemów komponowania, a także perspektywy rozwoju badań nad kolejnymi ich wersjami. Dalszy dyskurs odsłoni również aspekt użyteczności generowanych przez obie koncepcje rozwiązań w kontekście popularyzacji badań autorskich w różnych środowiskach i na różnych płaszczyznach upowszechniania wyników badań.

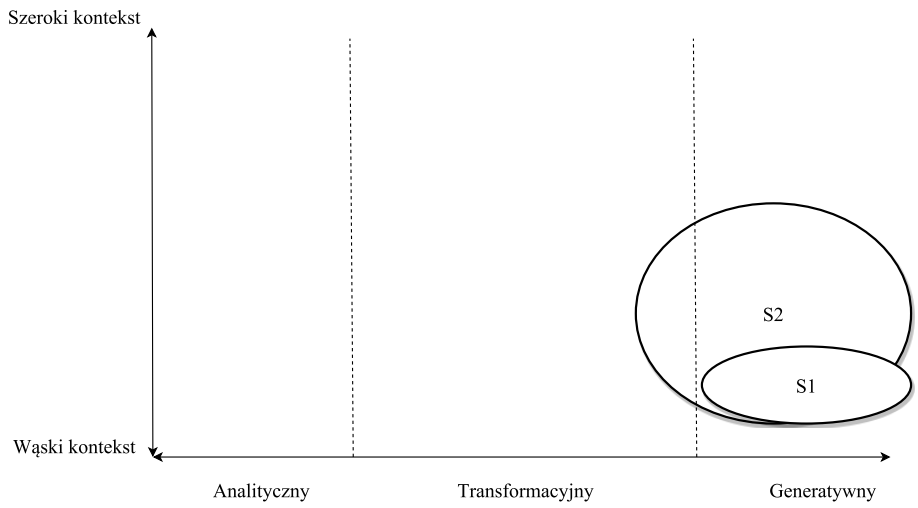
Rozdział 5

Konkluzje oraz perspektywy rozwoju i wdrożeń

Muzyka generatywna, będąca częścią sztuki generatywnej w rozumienia Bodena i Edmondsa [25], znalazła swoje miejsce w domenie muzyki powstałej na podstawie zbioru reguł, w postaci proceduralnej, z określoną dozą autonomii inwencji, która ją utworzyła. Poruszając się w dziedzinie informatyki, autor ograniczył pojęcie inwencji do programu komputerowego, a problematykę pracy ulokował w przestrzeni komputerowych systemów generowania muzyki. Przez ten pryzmat postrzegania procesu komponowania (aspekt zmechanizowany, po części zautomatyzowany) zaproponowane zostały dwie koncepcje systemów algorytmicznego komponowania muzyki.

Domeną pierwszej z nich jest tworzenie utworów z pogranicza muzyki tła, której definicję przytoczono w [45]. W założeniach koncepcji (str. 52) oraz implementacji (str. 75) tego systemu silnym uwarunkowaniem jest odejście od definicji formy muzycznej, na korzyść struktury zawierającej powiązane ze sobą wzorce muzyczne, pozostające w obrębie wspólnego motywu muzycznego oraz struktury rytmicznej. Wymienione wzorce stanowią podstawowy element budowy wynikowej kompozycji muzycznej. Sposób ich wzajemnego łączenia odnosi się do ustalenia podobieństwa tranzycji pomiędzy kolejnymi wzorcami, opisanymi w macierzy PST (przykład: rys. 2.5 na str. 65), zawierającej wartości prawdopodobieństwa tranzycji między-wzorcowej. Jako miarę podobieństwa przyjęto wartość różnicy pomiędzy funkcjami dystrybucji klas wysokości dźwięków, uwzględniającej model Parncutt'a [86]. W centrum procesu komponowania ulokowany został mechanizm funkcji generującej krótkie fragmenty kompozycji (na postawie przyjętej grupy parametrów), który z racji wykorzystywanego w tym procesie kontekstu i zakresu działania nazwano mini-generatorem (str. 56).

Odmienne podejście do procesu komponowania muzyki przedstawia system, w którego koncepcji (str. 66) i implementacji (str. 84) wyraźnie podkreślona została rola formy muzycznej opisującej powiązania pomiędzy zdefiniowanymi częściami kompozycji. Rola mini-generatora została zredukowana jedynie do tworzenia partii instrumentalnych dla instrumentów perkusyjnych, natomiast istotne znaczenie przywiązano do mechanizmu definiowania większych części kompozycji, dla których stosowane są oddzielne (choć nie jest to zasadą) reguły komponowania muzyki. Elementem centralnym w tym podejściu jest bazowy akord zbudowany w ramach określonej jako parametr skali muzycznej, a nowością, z punktu widzenia poprzedniej koncepcji, jest harmonia budowana na podstawie reguł kadencji muzycznych przedstawianych następnie w postaci macierzy tranzycji akordów (rys. 2.7, na str. 69). Generowanie linii melodycznej (str. 69) i towarzyszącej jej linii basowej (str. 71) stanowi dopełnienie dla kompleksowego systemu algorytmicznego komponowania, który umożliwia kreowanie muzyki w strukturze zamkniętej, posiadającej wstęp, rozwinięcie oraz zakończenie.



Rys. 5.1. Płaszczyzna pozycjonowania algorytmów komponowania muzyki Wollera w ujęciu autorskiego systemu algorytmicznego komponowania muzyki z uwzględnieniem ($S2$) oraz bez uwzględnienia ($S1$) formy kompozycji. Opracowanie własne.

W kontekście podsumowania obu koncepcji systemów warto przypomnieć o opisanym na początku tej pracy modelu Wollera (str. 42) - narzędzia pozwalającego na ulokowanie określonego systemu komponowania na płaszczyźnie innych rozwiązań (rys. 1.23). Przytoczony rysunek pokazuje ogólną lokalizację rozwiązań proponowanych przez autora, jednak w ujęciu ogólnym nie pozwalającym na odniesienie się do konkretnych przedstawionych implementacji. Z tego względu zasadnym wydaje się być podjęcie próby rozgraniczenia i lokalizacji omawianych rozwiązań w sposób bardziej szczegółowy, czego rezultatem jest płaszczyzna Wollera przedstawiona na rys. 5.1.

Na płaszczyźnie oznaczone zostały dwa obszary $S1$ i $S2$, które swoją powierzchnią wizualizują systemy algorytmicznego komponowania muzyki przedstawione kolejno w rozdziałach 2.2 i 2.3. Rozwiązanie w postaci systemu $S1$, generującego muzykę tła na podstawie wyników mini-generatorów, bez wykorzystania zdefiniowanej struktury kompozycji, jest według autora systemem o wysoce generatywnym charakterze i wąskim kontekście muzycznym. Ocena ta jest podyktowana przyjętym sposobem opisu procesu komponowania oraz mechanizmem generowania kompozycji wyjściowej, w którym tok postępowania przebiega dwuetapowo: początkowo mini-generatory budują wzorce muzyczne na podstawie wartości parametrów wejściowych, a następnie końcowy wynik muzyczny jest budowany poprzez "sklejanie" gotowych wzorców muzycznych, wybranych na podstawie przyjętej metryki podobieństwa między-wzorcowego. Widać więc w tym ujęciu "czysto" generatywny charakter systemu, ponieważ większe formy muzyczne powstają z początkowo względnie małych "cząstek".

Kolejnym aspektem jest kontekst muzyczny, brany pod uwagę przez system podczas procesu generowania kompozycji. Sam mechanizm powstawania wzorców muzycznych na podstawie wybranego wariantu motywu muzycznego (czyli sekwencji wysokości dźwięków znajdującej się w macierzy KMM, str. 57), bazuje na z góry wygenerowanej strukturze, dla której system nie dokonuje analizy zgodności tonacji pomiędzy kolejnymi wariantami (zakładając *a priori* ich zgodność). Podstawowym

motorem działania procesu "sklejania" (a więc dopuszczania sąsiedztwa kolejnych wzorców muzycznych) jest budowanie macierzy prawdopodobieństwa tranzycji (macierz PST, str. 65) pomiędzy danymi wzorcami, a także wybór odpowiedniej reguły komponowania dla mini-generatora (str. 58). Z tego względu kontekst muzyczny, w którym porusza się algorytm komponowania jest dosyć wąski, ponieważ nie bada on "większego" otoczenia, fragmentu kompozycji, dla której ma zostać "dobudowana" kolejna część struktury.

System $S1$ cechuje się także transformacyjnym charakterem algorytmu z uwagi na wykorzystanie reguł transformacyjnych podczas generowania wzorców muzycznych (str. 62). Cecha ta posiada jednak pewną dynamikę zmiany, ponieważ reguły są definiowane przez użytkownika i w ostatecznej formie systemu mogą zostać całkowicie pominięte - wtedy algorytm komponowania traci swój transformacyjny charakter.

Publikowanie wyników pracy

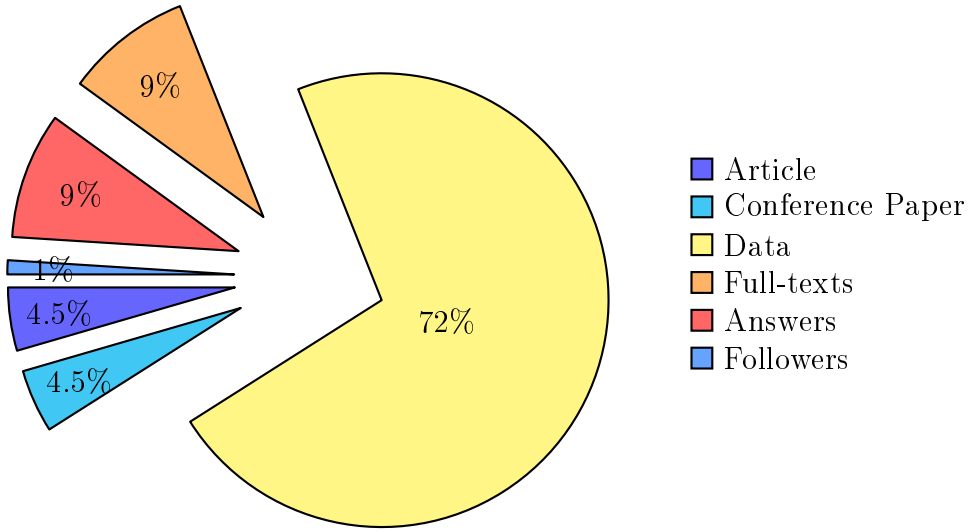
Obie koncepcje systemów algorytmicznego komponowania muzyki stanowią dorobek naukowy autora wypracowany na przestrzeni kilku lat, w którym nie brakowało zagłębiania się w różne kierunki rozwoju pracy badawczej. W tym miejscu ze względu na charakter rozdziału należałoby przybliżyć nieco ścieżkę rozwoju myśli autora, która skierowała go do momentu osiągnięcia pewnego dojrzałego poziomu efektów badań, uzyskanych w ciągu minionych lat.

Korzeni zainteresowania poruszaną w pracy tematyką należy szukać w [70] gdzie po raz pierwszy autor poruszył problematykę generowania muzyki przez maszynę za pomocą heurystyki algorytmów genetycznych. Publikacja opisuje podsumowanie eksperymentów i dokonań autora z okresu przygotowywania pracy inżynierskiej, w której została podjęta próba zaimplementowania funkcji przystosowania, szacującej stopień zgodności "osobników" - macierzy nut (opis budowy tej struktury przedstawiono na str. 13) zbudowanej z sampli ("chromosomów") - krótkich fragmentów muzycznych, których prawidłowość ułożenia w "osobniku" oceniano za pomocą różnych kryteriów akceptowalności sąsiedztwa. Owoce tej pracy zaprezentowano w ramach Festiwalu Sztuki Digitalnej digital_ia.09 na jednym z paneli dyskusyjnych [4].

Następnym etapem w rozwoju pracy naukowej autora było badanie możliwości kreowania muzyki sztucznej za pomocą odpowiednio zmodyfikowanych łańcuchów Markowa. Wyniki otrzymane w ramach pracy magisterskiej opublikowano w [7], gdzie zaprezentowano koncepcję generowania melodii monofonicznych, do budowy których wykorzystane zostały probabilistyczne dane na temat wysokości, czasu trwania i położenia danego dźwięku muzycznego w melodii.

W następnych latach autor zaczął rozwijać całkowicie odmienne niż przyjęte do tej pory podejście do tworzenia muzyki generatywnej. Rok 2010 przyniósł zalążki późniejszej koncepcji algorytmicznego komponowania muzyki za pomocą mniejszych fragmentów kompozycji - wzorców muzycznych. W pracach [71][8] czy [9] poruszone zostały aspekty tworzenia modeli generujących muzykę z wykorzystaniem reguł komponowania oraz wejściowych danych muzycznych, które następnie analizowano i segmentowano. Otrzymane w ten sposób segmenty stanowiły podstawę do tworzenia (bazując na przyjętej metryce podobieństwa otrzymanych segmentów) "nowych" wyjściowych danych muzycznych.

Autor od koncepcji systemu komponowania o charakterze analityczno-transformacyjnym doszedł do idei generowania muzyki od podstaw - z krótkich fragmentów reprezentowanych jako motywy czy figury muzyczne,



Rys. 5.2. Składowe oceny RG Score = 2.47 autora. Dane na dzień 06.03.2017. Opracowanie własne.

utworzone za pomocą dedykowanych do tego celu komponentów - mini-generatorów. Początkowo czysto generatywny system algorytmicznego komponowania został wzbogacony o system reguł i dodatkowych parametrów wejściowych (jak choćby opis struktury formy kompozycji wyjściowej), które pozwoliły na tworzenie kompozycji zamkniętych, posiadających wstęp, rozwinięcie i zakończenie. Wyniki uzyskane przez autora zaprezentowano w pracach [10] i [11].

W ramach opisu procesu publikowania wyników badań warto wspomnieć o popularyzatorskiej działalności autora, w której zaangażował się on w prezentowanie rzeczywistych kompozycji i własnych koncepcji na antenie Polskiego Radia Szczecin (audycja "Trącić Myszką" [6, 5]) czy podczas różnych lokalnych festiwali i wydarzeń z branży IT ("rabbIT" oraz "Noc Naukowców"). Podczas wymienionych wydarzeń autor z wykorzystaniem sprzętu multimedialnego oraz prototypu własnego systemu algorytmicznego komponowania muzyki oddziaływał na uczestników pokazów, a także dawał możliwość kreowania własnych konpcecji muzycznych opisując i udostępniając autorskie metody generowania kompozycji. Akcje promowania wyników badań były prowadzone również wśród najmłodszych słuchaczy czego przykładem może być Festiwal Nauki i Sztuki w Stargardzie, podczas którego autor prezentował możliwość zalgorytmizowania tworzenia muzyki za pomocą komputera [64]. Popularyzowanie autorskiej koncepcji odbywało się również przy okazji wydarzeń kulturalnych takich jak wernisaże fotografii, a efekty pracy autora docierały nawet do odległych stron świata, czego przykładem jest udział w warsztatach technologiczno-kulturowych w Pekinie i Shenzhen jako laureata pierwszej polskiej edycji programu "Telecom seeds for the future" organizowanego przez koncern Huawei. [102]

Przez wszystkie te lata autor ciągle poszukiwał wyrazów akceptowalności tworzonej muzyki generatywnej, a pośrednio - metod jej komponowania. Potwierdzeniem działań w kierunku popularyzowania wyników autorskiej pracy naukowej mogą być różne aktywności lub dane statystyczne pochodzące z profilu autora na platformie Research Gate. W tab. 5.1 zaprezentowane zostały wybrane dane profilowe, mające wpływ na ogólną rozpoznawalność naukową autora, reputację, której wartość

Nazwa statystyki	Wartość
Article	2
Conference Paper	2
Data	32
Full-texts	4
Answers	1
Followers	47
Following	65
Reads	961
RG Reach	49

Tab. 5.1. Statystyki profilu autora na portalu ReasearchGate. Dane na dzień 06.03.2017. Opracowanie własne.

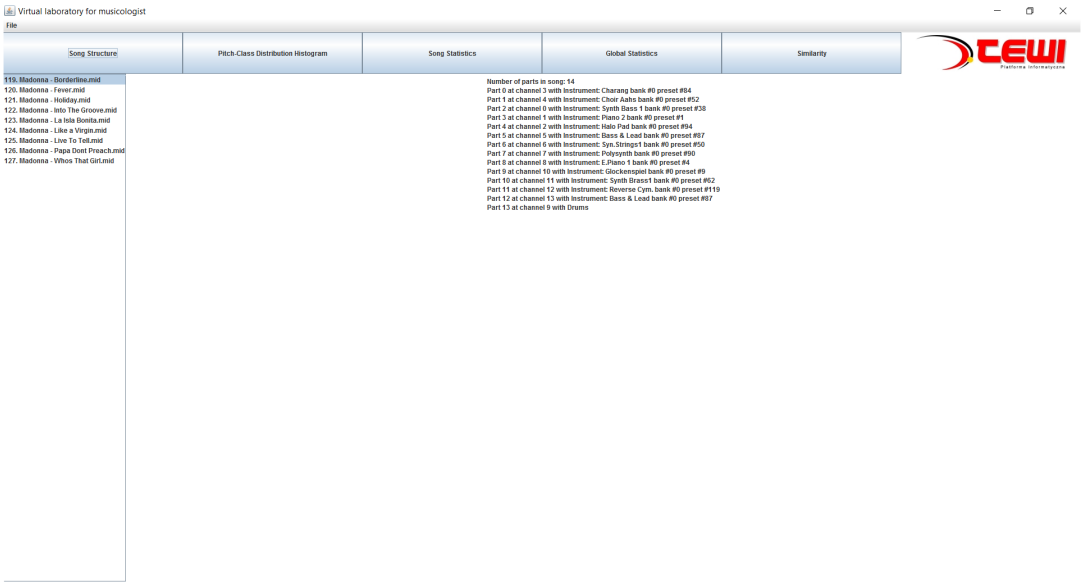
określana jest jako *RG Score*, a jej składowe w ujęciu profilu autora przedstawiono na rys. 5.2.

Przeważającą część (72%) oceny *RG Score* autora stanowią dane naukowe (*Data*), które w tym przypadku są kompozycjami muzycznymi uzyskanymi przez autorskie metody tworzenia muzyki generatywnej od roku 2014. Na liście utworów [87] znaleźć można wyniki eksperymentów nie tylko *stricte* związanych z badaniami systemów algorytmicznego komponowania muzyki przedstawionych w rozdziale 4, ale i prób tworzenia muzyki sztucznej za pomocą odpowiednio spreparowanych automatów komórkowych. Dzięki platformie autor mógł podzielić się wynikami uzyskanymi w badaniach opisanych w ramach publikacji naukowych i prezentowanych na różnych konferencjach międzynarodowych takich jak *FedCSIS [9]* czy *ICNAAM* [11]. Część uzyskanych autorskich kompozycji muzyki generatywnej znajduje się także w pliku *kompozycje.zip*, znajdującym się na załączonej do pracy płycie CD.

Perspektywy rozwoju

Działalność popularyzatorska autora jest jedną ze składowych jego dorobku naukowego ocenianego w tej pracy. Kolejnym istotnym aspektem są próby urzeczywistnienia różnych autorskich koncepcji naukowych do postaci działającego systemu informatycznego. Jednym z kroków podjętych w tym kierunku jest aplikacja "Wirtualne Laboratorium Muzykologa" wykonana w ramach projektu TEWI [93] i stanowiąca wstęp do utworzenia złożonego systemu algorytmicznego komponowania muzyki generatywnej na podstawie wzorców muzycznych pozyskanych z wejściowego zbioru gotowych utworów.

Na rys. 5.3, 5.4, 5.5, 5.6 i 5.7 przedstawione zostały kolejne zrzuty ekranów aplikacji, a jej kod źródłowy zawarty jest na dołączonej do pracy płycie CD. W założeniach prezentowana aplikacja stanowić ma podstawę do rozbudowy wirtualnej stacji roboczej, oferującej możliwość analizowania materiału muzycznego zapisanego w postaci bajtowej (formacie $MIDI$). Narzędzie zostało napisane za pomocą języka programowania $JAVA$, a metody przetwarzające dane muzyczne zostały opracowane na podstawie frameworka *MIDI Toolbox* dla środowiska Matlab autorstwa Eerola i Toiviainena [48].



Rys. 5.3. Ekran Song Structure z aplikacji "Virtual laboratory for musicologist". Opracowanie własne.

Na rys. 5.3 pokazany został ekran zawierający listę wykorzystanych w przykładowym utworze instrumentów wraz z ich przynależnością do konkretnego kanału $MIDI$. Dane te stanowią podstawę do analizowania instrumentalizacji utworu, a także doboru instrumentów do konkretnych partii kompozycji.

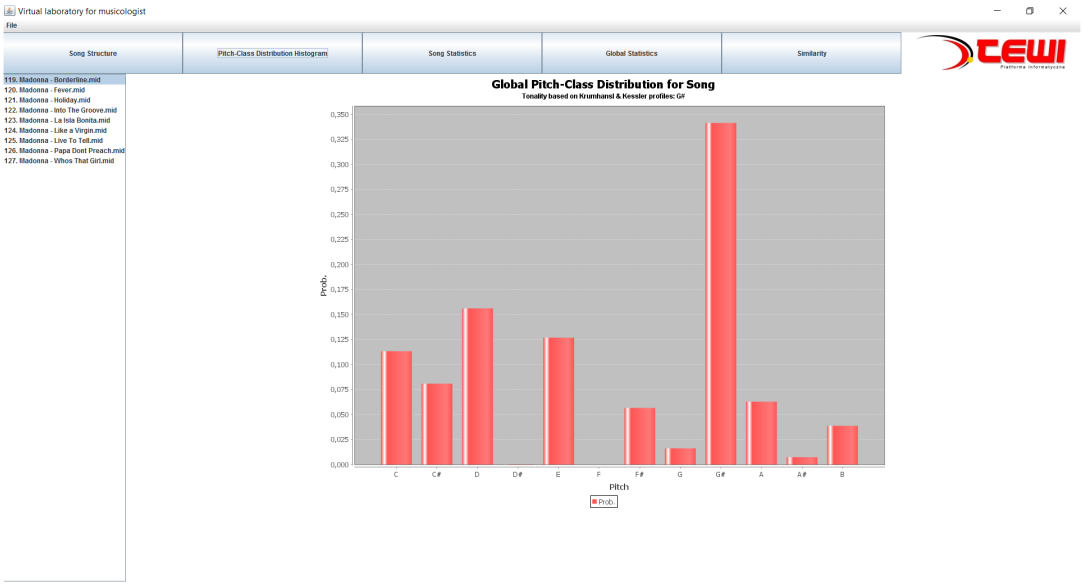
Główną funkcjonalnością aplikacji jest dostarczanie danych statystycznych opisujących zawartość kompozycji. Jednym z aspektów tej analizy jest wyznaczanie histogramu dystrybucji wysokości dźwięków muzycznych dla danego utworu, czego przykładem jest zrzut ekranu zaprezentowany na rys. 5.4.

Kolejną z funkcjonalności programu jest wyznacznie funkcji rozkładu wysokości, dynamiki oraz wartości rytmicznych dźwięków występujących w kompozycji (rys. 5.5). Dodatkowo dla użytkownika dostępne są również dane dotyczące skrajnych wartości wysokości, długości dźwięku czy tempa utworu.

Dla zbioru kompozycji dostępne są dane dotyczące globalnego rozkładu klas wysokości dźwięków w kolekcji, rozkładu wysokości i dynamiki dźwięków oraz wybranych danych statystycznych utworu (rys. 5.6).

Jedną z ważniejszych funkcji aplikacji jest wyznaczanie podobieństwa każdego z utworów kolekcji do jego średniego przedstawiciela opisanego za pomocą histogramów: dystrybucji klas wysokości (PCD), wysokości i dynamiki dźwięków muzycznych występujących w kompozycji (rys. 5.7). Informacje otrzymane w wyniku analizy przedstawionych danych mogą posłużyć do przygotowania struktur podobnych do tych przedstawionych w rozdziale 3, dotyczącym implementacji systemu algorytmicznego komponowania muzyki.

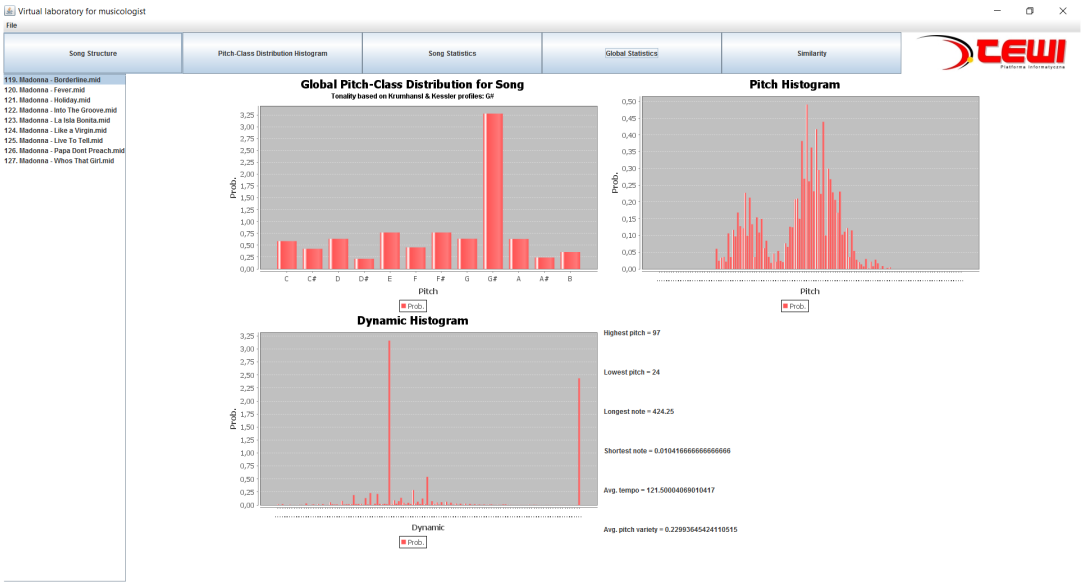
Przytoczony opis wykonanej przez autora aplikacji stanowi jedynie wstęp do przygotowania większego systemu informatycznego, który posiłkując się danymi wejściowymi w postaci zbiorów muzycznych oraz mechanizmem ich odpowiedniego analizowania pod kątem różnych charakterystyk, byłby w stanie pełnić funkcję półautomatycznego narzędzia, służącego do tworzenia muzyki generatywnej. W zamierzeniu autora pozostaje utworzenie takiego systemu o charakterze rozproszonym dostępnym w postaci on-line, który proces syntezy materiału wyjściowego przerzu-



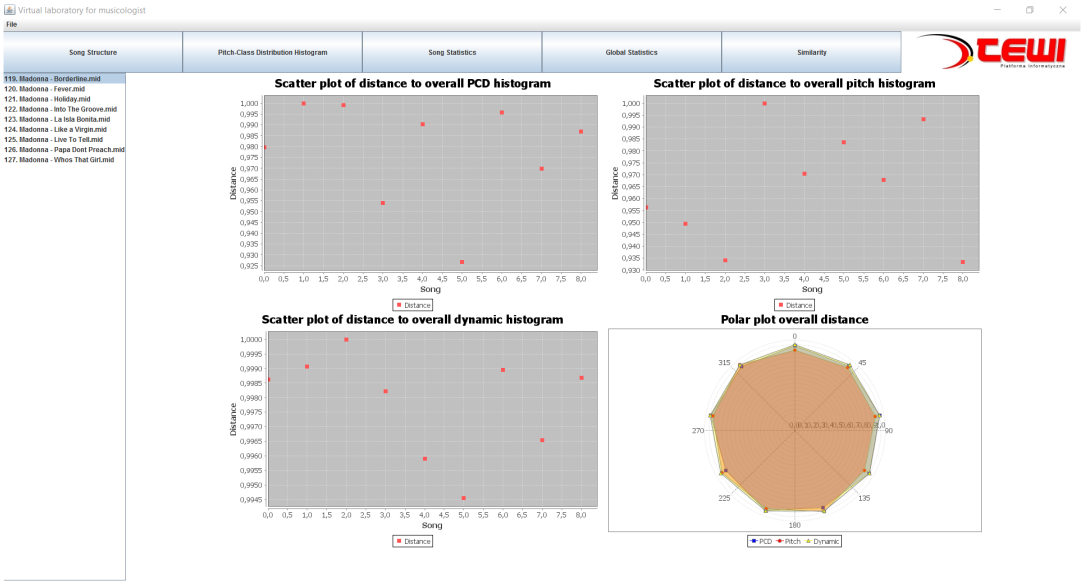
Rys. 5.4. Ekran Pitch Class Distribution Histogram z aplikacji "Virtual laboratory for musicologist". Opracowanie własne.

Rys. 5.5. Ekran Song Statistics z aplikacji "Virtual laboratory for musicologist". Opracowanie własne.

całby na barki zewnętrznych systemów software'owej syntezy muzyki. Proces ten prawdopodobnie opierałby się na mechanizmie kolejkowania zadań i powiadamiania użytkowników o gotowości uzyskania wygenerowanej kompozycji np. na drodze mailowej, gdzie użytkownik otrzymywałby odnośnik do pobrania pliku audio. Autor liczy, że w toku rozpowszechniania wyników badań naukowych, znajdzie się podmiot komercyjny, który urzeczywistni choć w niewielkim stopniu dalsze cele badawczo-rozwojowe, które stale motywują autora do podejmowania różnych inicjatyw związanych z promowaniem swojej pracy badawczej.



Rys. 5.6. Ekran Global Statistics z aplikacji "Virtual laboratory for musicologist". Opracowanie własne.



Rys. 5.7. Ekran Similarity z aplikacji "Virtual laboratory for musicologist". Opracowanie własne.

Wnioski końcowe

Nawiązując do komponentów problemu badawczego sformułowanych na str. 48 - celu i hipotezy badawczej należy ocenić, że zostały one osiągnięte. Cel opisany jako *opracowanie opracowanie modelu komputerowego systemu komponowania muzyki użytkowej (posiadającej wartość rynkową), w którym do generowania oryginalnych utworów muzycznych wykorzystuje się metody autorskie oparte na metodach sztucznej inteligencji,* został bez wątpienia osiągnięty. W rozdziale 2 zostały szczegółowo opisane owe autorskie metody podkreślając ich nowatorskie aspekty. Autor uważa je za własne, pozbawione wad prawnych i innowacyjne wg często przywo-

ływanego dziś kryterium - są nowe i mają wartość (metodyczną, techniczną i, być może, rynkową).

Hipoteza opisana jako przypuszczenie, że *bazując na metodach sztucznej inteligencji, możliwe jest opracowanie takich algorytmów komponowania utworów muzycznych, których efektem będą utwory atrakcyjne w odbiorze i posiadające strukturę poprawną w aspekcie teorii muzyki*, zdaniem autora także została dowiedziona.

W procesie komponowania starannie przestrzegano zasad poprawności tworzenia muzyki wg jej kanonów historycznie utrwalonych, lecz także doskonalonych. Spełniony został także, w przekonaniu autora postulat atrakcyjności muzyki. Dowodzą tego liczne akcje popularyzatorskie i aktywność w społeczności Research Gate.

Autor ocenia więc, że w procesie rozwiązywania postawionego problemu twórczego został on rozwiązany, jednak nie w jeden jedyny dopuszczalny sposób. Wszak mnogość rozwiązań to właśnie istota muzyki generatywnej, tak intensywnie rozwijającej się we współczesnym świecie, w świecie muzyki ciągle rozwijającej się zarówno w aspekcie kompozycji, wykonania, jak i jej wpływu na człowieka.

Muzyka i siła jej oddziaływania na człowieka fascynowała uczonych od zarania dziejów. Już Platon i Arystoteles zastanawiali się nad jej oddziaływaniem na duszę. Prof. David Huron w swoich licznych publikacjach na temat początków muzyki stwierdził, że może być ona ewolucyjnym przystosowanie się człowieka i, choć problem ten w dalszym ciągu pozostaje w sferze spekulacji, to jednak istnieje wiele przesłanek przemawiających za jej korzeniami biologicznymi. [42]

Ciekawymi spostrzeżeniami z tej dziedziny podzielił się także prof. Leonid Perlovsky, odnosząc się do różnych aspektów ludzkiej natury. W [63] stwierdził on m. in. że w każdym z rodzajów sztuki emocje, które są wyzwalane podczas odbioru dzieła, są ważne, jednakże muzyka, poprzez swoje zróżnicowanie, pobudza pierwotne ośrodki neuronowe oraz posiada nieporównywalnie silniejsze oddziaływanie na emocje niż inne dziedziny sztuki. Muzyka różnicuje estetyczną potrzebę człowieka, jaką jest instynkt wiedzy i robi to w dużo większym stopniu niż inne dziedziny sztuki. Dźwięki mogą dotrzeć do najstarszych, nieświadomych głębi ludzkiej psychiki, lecz mogą również odnosić się do najbardziej podniosłych idei ludzkiej egzystencji.

Muzyka generatywna (urzeczywistniona w kompozycja autorskich prezentowanych w ramach tej pracy) dopiero się rozwija [103]. Przewidywać należy jej dalszy ustawiczny rozwój. Wiele ośrodków badawczych (np. *MIT* [28]) wymienionych w tej pracy eksperymentuje zarówno z metodami tworzenia muzyki jak i sposobami jej wykonywania. Nieodwracalny jest rozwój metod sztucznej inteligencji i zacieranie granic pomiędzy naszą i komputerową inteligencją. Zapewne także muzyka sztuczna będzie uważana za przejaw transhumanizmu i będzie przedmiotem zaciekłych debat wśród muzyków, informatyków i filozofów nauki. Bardzo prawdopodobne będzie także jej komercyjne wykorzystanie.

Autor podejmując opisywaną w tej pracy problematykę kierował się, prócz własnych fascynacji, także a może i przede wszystkim aktualnością prowadzonych badań. Ośrodki takie jak *MIT*, *Columbia University* [97], *University College London* [68] czy *Griffith University* [98], oferując w ramach prowadzonych kursów tematykę związaną z muzyką generatywną, kształtują kierunek rozwoju tej gałęzi nauki. W odczuciu autora dorobek badawczy przedstawiony w tej pracy pozostaje w zgodzie z tym narastającym trendem, niemniej jednak pozostawiając w dalszym ciągu szerokie pole do doskonalenia, rozszerzania i poprawiania uzyskanych autorskich metod tworzenia muzyki generatywnej.

Bibliografia

[1] Gm1 sound set. [online]. [dostęp: 2013-08-26 17:04Z].

[2] MIDI File Format - The Sonic Spot. [online]. [dostęp: 2014-08-26 17:04Z].

[3] MIDI Files. [online]. [dostęp: 2014-08-26 17:04Z].

[4] Digitalia09. dyfuzja technologii i imaginacji. Panel dyskusyjny, 2009. [dostęp online 05.03.2017: `http://www.szczecin.eu/en/zycie_w_miescie/miasto_poleca/festiwale/8986-festiwal_sztuki_digitalnej_digital_ia09.html`].

[5] Trącić myszką. komputerowy kompozytor w domowym studiu nagraniowym. Audycja radiowa, 2011. [dostęp online 04.03.2017: `http://www.myszka.org/archiwum-myszki/tracic-myszka-2011/06112011-komputerowy-kompozytor-w-domowym-studiu-nagraniowym`].

[6] Trącić myszką. synteza i przetwarzanie dźwięku. Audycja radiowa, 2011. [dostęp online 04.03.2017: `http://www.myszka.org/archiwum-myszki/tracic-myszka-2011/17102011-synteza-i-przetwarzanie-dzwieku`].

[7] Mazurowski Ł. Modele inżynierii teleinformatyki, wybrane zastosowania 4. chapter Łańcuchy Markowa w systemach generowania muzyki sztucznej, pages 135–146. Politechnika Koszalińska, Koszalin, 2009.

[8] Mazurowski Ł. An algorithmic composition - the chosen aspects. *Przegląd Elektrotechniczny*, R. 88, nr 10b:243–246, 2012.

[9] Mazurowski Ł. Computer models for algorithmic music composition. In *2012 Federated Conference on Computer Science and Information Systems (FedCSIS)*, pages 733–737, Sept 2012.

[10] Mazurowski Ł. Algorithmic composition transformational-generative system for background music generation. *Przegląd Elektrotechniczny*, R. 91, nr 2:112–117, 2015.

[11] Mazurowski Ł. Generative electronic background music system. *AIP Conference Proceedings*, 1648(1):660004, 2015.

[12] Biles J. A. Genjam: a genetic algorithm for generating jazz solos. *Proceedings of ICMC'94*, pages 131–137, 1994. University of Michigan Library, MI: MPublishing.

[13] Biles J. A. Genjam: Evolutionary computation gets a gig. In *Third Conference on Information Technology Curriculum*, Rochster, New York, 2002.

[14] Biles J. A. Genjam in transition: from genetic jammer to generative jammer. In *International Conference on Generative Art*, Milan, Włochy, 2002.

[15] Gartland-Jones A. Musicblox: A real-time algorithmic composition system incorporating a distributed interactive genetic algorithm. *Applications of Evolutionary Computing: EvoWorkshops 2003. LNCS*, 2611:490–501, 2003.

[16] Shoenberg A. *Fundamentals of Music Compositioni.* Clays Ltd, St Ives plc., Londyn, 1970.

[17] Oppenheim D.V. Pazel D.P. Wright J. Abrams S., Furher R. A framework for representing and manipulating tonal music. In *International Computer Music Conference-2000*, San Francisco, 1999. IBM, International Computer Music Association.

[18] Pazel D.P. Wright J. Abrams S., Oppenheim D.V. Higher-level composition control in music sketcher: Modifiers and smart harmony. In *International Computer*

Music Conference-1999, San Francisco, 1999. IBM, International Computer Music Association.

[19] Rodrigues J. F. Assayag G., Feichtinger H. G. *Mathematics and Music. A Diderot Mathematical Forum.* Springer-Verlag, Berlin Heidelberg, 2002.

[20] Bel B. Migrating musical concepts: an overview of the bol processor. *Computer Music Journal*, 22(2).

[21] Eno B. Generative Music: Evolving metaphors, in my opinion, is what artists do. *Motion Magazine*, 1996.

[22] Jacob B. Algorithmic composition as a model of creativity. *Organised Sound*, 1.

[23] Jacob B. Composing with genetic algorithms. In *Proceedings of the 1995 International Computer Music Conferencee*, 1995.

[24] Bentley P. Blackwell T.M. Improvised music with swarms. In *Evolutionary Computation, 2002. CEC '02. Proceedings of the 2002 Congress on*, volume 2, pages 1462–1467, 2002.

[25] Edmonds E.A. Boden M.A. What is generative art? *Digital Creativity*, 20:21–46, 2009.

[26] Crawford T. Byrd D. Problems of music information retrieval in the real world. *Information Processing and Management*, 38:249–272, 2002.

[27] Ariza C. Prokaryotic groove: Rhythmic cycles as real-value encoded genetic algorithms. In *Proceedings of the 2002 International Computer Music Conferencee*, 2002.

[28] Ariza Ch. 21m.380 music and technology: Algorithmic and generative music. 2010. [dostęp online 05.03.2017: `https://ocw.mit.edu`].

[29] Jain S. Wadhwa S. Chaudhury S., Nag T.C. Role of sound stimulation in reprogramming brain connectivity. *Journal of Biosciences*, 38(3):605–614, 2013.

[30] Miikkulainen R. Chen C. C. J. Creating melodies with evolving recurrent neural network. *Proceedings of IJCNN'01*, 2001.

[31] Krumhansl C.L. *Cognitive Foundations of Musical Pitch.* Oxford University Press, New York, 1990.

[32] Dale N. Cockerton T., Moore S. Cognitive test performance and background music. *Perceptual and Motor Skills*, 85:1435–1438, 1997.

[33] E. F. Codd. *Cellular Automata.* Academic Press, Inc., Orlando, FL, USA, 1968.

[34] Hofstadter D.R. Cope D. *Virtual music: computer synthesis of musical style.* Mass.: MIT Press, Cambridge,, 2001.

[35] Conklin D. Music generation from statistical models. In *Proceedings of the AISB 2003 - Symposium on Artificial Intelligence and Creativity in the Arts and Sciences*, pages 30–35, 2003.

[36] Cope D. An expert system for computer-assisted composition. *Computer Music Journal*, 11(4):30–40, 1987.

[37] Cope D. *Computers and Musical Style.* A-R Editions, 1991.

[38] Cope D. Recombinant music using the computer to explore musical style. *IEEE Computer*, 24:22–28, 1991. IEEE Computer Society Press Los Alamitos.

[39] Cope D. Computer modeling of musical intelligence in emi. *Computer Music Journal*, 16:69–83, 1992. The MIT Press.

[40] Cope D. *Experiments in musical intelligence.* A-R Editions, 1996.

[41] Cope D. *The algorithmic composer.* A-R Editions, 2000.

[42] Huron D. Is music an evolutionary adaptation? *Annals of the New York Academy of Sciences*, 930(1):43–61, 2001.

[43] Ralley D. Genetic algorithm as a tool for melodic developmenty. In *Proceedings of the 1995 International Computer Music Conferencee*, 1995.

[44] Goldberg D.E. *Algorytmy genetyczne i ich zastosowania.* Wydawnictwa Naukowo-Techniczne, Warszawa, 2004.

[45] Collins English Dictionary. background music. Dostęp on-line: 26-07-2014.

[46] Ames Ch. Domino M. Understanding music with ai. chapter Cybernetic composer: an overview, pages 186–205. MIT Press, Cambridge, MA, USA, 1992.

[47] Baker J. E. Reducing bias and inefficiency in the selection algorithm. In *Proceedings of the Second International Conference on Genetic Algorithms and their Application*, pages 14–21. USA: Lawrence Erlbaum Associates, 1987.

[48] Toiviainen P. Eerola T. Mir in matlab: The midi toolbox. *Proceedings of 5th International Conference on Music Information Retrieval*, pages 22–27, 2004.

[49] Correa D.C. et al. Neural network based systems for computer-aided musical composition:supervised x unsupervised learning. *SAC*, pages 1738–1742, 2008. ACM.

[50] Dubnov S. et al. Using machine-learning methods for musical style modeling. *IEEE Computer*, 36(10), 2003.

[51] Lartillot O. et al. Automatic modeling of musical style. *Proceedings of ICMC'02*, 2002.

[52] Pachet F. Playing with Virtual Musicians: the Continuator in Practice. *IEEE Multimedia*, 9.

[53] Pachet F. The continuator: Musical interaction with style. In *ICMC Conference Proceedings 2002*, pages 211–218, Göteborg, Sweden, 2002.

[54] Rosenblatt F. The perceptron: A probabilistic model for information storage and organisation of the brain. *Psychological Review*, (65):368–408, 1958.

[55] Wesołowski F. *Zasady muzyki.* Polskie Wydawnictwo Muzyczne, Kraków, 1998.

[56] Schoner B. Farbood M. Analysis and synthesis of Palestrina-style counterpoint using Markov chains. In *International ComputerMusic Conference*, 2001.

[57] Copley P. Gartland-Jones A. The suitability of genetic algorithms for music composition. *Contemporary Music Review*, 22:43–55, 2003.

[58] Tuson A. Geraint Wiggins G. Papadopoulos G., Phon-amnuaisuk S. Evolutionary methods for musical composition. In *In International Journal of Computing Anticipatory Systems*, 1998.

[59] Sametti A. Goffredo H. Scoresynth: A system for the synthesis of music scores based on petri nets and music algebra. *IEEE Computer*, 24(7):56–60, 1991.

[60] Slimane M. Guéret C., Monmarché N. Ants can play music. In Blum C.-Gambardella L.M. Mondada F. Stützle T. Dorigo M., Birattari M., editor, *Ant Colony Optimization and Swarm Intelligence*, volume 3172 of *Lecture Notes in Computer Science*, pages 310–317. Springer Berlin Heidelberg, 2004.

[61] Isaacson L. Hiller L. Musical Composition with a High-Speed Digital Computer. *Journal of the Audio-Engineering Society*, 6:154–160, 1958.

[62] Ayres L. Horner A. Harmonisation of musical progression with genetic algorithms. In *Proceedings of the 1995 International Computer Music Conferencee*, 1995.

[63] Perlovsky L. I. Music – the first principle. role of music in evolution of consciousness. *Musical Theater*, 2005.

[64] Info-wiadomości. Festiwal nauki i sztuki oraz dzień otwarty w i liceum ogólnokształcącym im. adama mickiewicza w stargardzie, 2014. [dostęp online 08.03.2017: `http://www.wiadomosci.rii.pl/index.php?H=11&HH=9970`].

[65] Renkewitz F. Kampfe J., Sedlmeier P. The impact of background music on adult listeners: A meta-analysis. *Psychology of Music*, 39:424–448, 2011.

[66] Toiviainen P. Krumhansl C. L. *Tonal cognition.* Annals of the New York Academy of Sciences. NY: New York Academy of Science, 2004.

[67] Jackendoff R. Lerdhal F. *A Generative Theory of Tonal Music.* Harmonologia Series, No 6. MIT Press, 1983.

[68] University College London. Compgc20 - computer music, 2016. [dostęp

online 07.03.2017: `http://www.cs.ucl.ac.uk/students/syllabus/msccs/gc20_computer_music/`].

[69] Langsman M. *Algorithmic Composition. MSc. Thesis.* University of Edinburgh, The School of Arts, Culture and Environment, 2004.

[70] Wiliński A. Mazurowski Ł. Koncepcja generowania utworu muzycznego przez maszynę – akt sztucznej twórczości opartej na algorytmach genetycznych. *Metody Informatyki Stosowanej*, nr 3 (Tom 16):203–212, 2008.

[71] Wiliński A. Mazurowski Ł. Computer models of similarity in musical pieces based on the multi-dimensional patterns. *Journal of Applied Computer Science*, Vol. 20, nr 2:79–94, 2012.

[72] Mozer M.C. Neural Network Music Composition by Prediction: Exploring the Benefits of Psychoacoustic Constraints and Multi-scale Processing. *Connection Science*, 6:247–280, 1994.

[73] Hoggar S. McAlpine K., Miranda E.R. Making music with algorithms: A case-study system. *Computer Music Journal*, 23(2).

[74] Jung-Ju C. Min-Joon Y., In-Kwon L. *Background Music Generation Using Music Texture Synthesis*, volume 3166 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2004.

[75] Biles J. A. Miranda E. R. *Evolutionary Computer Music.* Springer-Verlag, Londyn, 2007.

[76] Tominaga K. Miura T. An approach to algorithmic music composition with an artificial chemistry. In *Proc. of the German Workshop on Artificial Life*, 2006.

[77] Cook N. *A guide to musical analysis.* OXFORD University Press, 1994.

[78] Galanter P. What is generative art? complexity theory as a context for art theory. In *6th Generative Art Conference*, 2003.

[79] Janata P. Erp measures assay the degree of expectancy violation of harmonic contexts in music. *Cognitive Neuroscience*, (2):152–154, 1995.

[80] Reagan P. *Computer music generation via decision tree learning. Master Thesis.* Department of Computer Science, Carnegie Mellon University, 1999.

[81] Toiviainen P. Symbolic ai versus connectionism in music research. In Miranda E.R., editor, *Readings in Music and Artificial Intelligence.* Harwood Academic Publishers, 2000.

[82] Wiggins G. Papadopoulos G. A genetic algorithm for generation of jazz melodie solos. *Proceedings of STEP'98*, pages 7–9, 1998.

[83] Wiggins G. Phon-Amnuasiuk S. The four-part harmonisation problem: A comparison between genetic algorithms and a rule-based system. In *Proceedings of AISB 99*, 1999.

[84] Brown A. R. Opportunities for evolutionary music composition. *Proceedings of the ACMC'02*, pages 27–34, 2001.

[85] Miranda E. R. *Composing Music with Computers (Music Technology),.* Focal Press, 2001.

[86] Parncutt R. A perceptual model of pulse salience and metrical accent in musical rhytms. *Music Percepction*, 11:409–464, 1994.

[87] ResearchGate. Łukasz mazurowski - publications, 2017. [dostęp online 07.03.2017: `https://www.researchgate.net/profile/ukasz_Mazurowski/publications?category=data`].

[88] Inokushi S. Saiwaki N., Kato K. An approach to analysis of eegs recorded during music listening. *Journal of New Music Research*, (26):227–243, 1997.

[89] Chiu S.C. Shan M.K. Algorithmic compositions based on discovered musical patterns. *Multimedia Tools and Applications*, 46.

[90] Buttram T. Beyond games: Bringing directmusic into the living room. In *DirectX*

9 Audio Exposed: Interactive Audio Development, Texas, 2004. USA: Wordware Publishing Inc.

[91] Holmes T. *Electronic and Experimental Music. Third ed.* Routledge, New York, 2008.

[92] Unemi T. Sbeat3: A tool for multi-part music composition by simulated breeding. In *Proceedings of the Eighth International Conference on Artificial Life (A-Life VIII)*, 2002.

[93] TEWI. Wirtualne laboratorium muzykologa, 2013. [dostęp online 07.03.2017: `http://tewi.p.lodz.pl`].

[94] Bharucha J.J. Tillmann B., Janata P. Activation of the inferior frontal cortex in musical priming. *Cognitive Brain Research*, (16):145–161, 2003.

[95] Krumhansl C. L. Toiviainen P. Measuring and modeling real-time responses to music: the dynamics of tonality induction. *Perception*, 32.

[96] Setomoto M. Tominaga K. An artificial-chemistry approach to generating polyphonic musical phrases. In *Applications of Evolutionary Computing*, volume 4976 of *Lecture Notes in Computer Science*, pages 463–472. Springer Berlin Heidelberg, 2008.

[97] Columbia University. Computer music center, 2017. [dostęp online 07.03.2017: `http://cmc.music.columbia.edu/cmc_courses/`].

[98] Griffith University. Generative music (3710qcm), 2016. [dostęp online 07.03.2017: `https://degrees.griffith.edu.au/Course/3710QCM`].

[99] Fayer M. Verbeurgt K., Dinolfo M. Extracting patterns in music for composition via markov chains. In *Proceedings of the 17th international conference on Innovations in applied artificial intelligence*, IEA/AIE'2004, pages 1123–1132. Springer Springer Verlag Inc, 2004.

[100] Born J. Voigt H.M., Santibanez-Koref I. Hierarchically structured distributed genetic algorithm. In *Parallel Problem Solving from Nature, 2: Proceedings of the Second Conference on Parallel Problem Solving from Nature*, pages 145–154. Elsevier Science Inc. New York, NY, USA, 1992.

[101] Zabierowski W. Accord sequences generation in agreement with harmony tonal rules using cascade artificial neural networks. *Pomiary Automatyka Kontrola*, (11):1394–1396, 2011.

[102] Spider's Web. Po co huawei zabrał polskich studentów do chin? - relacja spider's web, 2014. [dostęp online 08.03.2017: `http://www.spidersweb.pl/2014/08/polscy-studenci-chiny-huawei.html`].

[103] Wikipedia. Generative music — wikipedia, the free encyclopedia, 2017. [dostęp online 07.03.2017: `https://en.wikipedia.org/wiki/Generative_music`].

[104] Miranda E.R. Diederich J. Wooller R., Brown S.R. A framework for comparison of process in algorithmic music systems. In *Generative Arts Practice-2005 A Creativity and Cognition Symposium*, Sydney, 2005. Excellence in Research Australia (ERA), University of Technology.

[105] Wyrzykowski K. Wrycza S., Marcinkowski B. *Język UML 2.0 w modelowaniu systemów informatycznych.* Helion, Gliwice, 2005.

[106] I. Xenakis. *Formalized Music: Thought and Mathematics in Composition.* Harmonologia Series, No 6. Pendragon Press, 1992.

[107] Napieralski A. Zabierowski W. The problem of grammar choice for verification of harmonisation process in tonal harmony. In *Proceedings of the International Conference Modern Problems of Radio Engineering, Telecommunications and Computer Science*, pages 86–88. House of Lviv Polytechnic National University, 2008.

[108] Napieralski A. Zabierowski W. Generation of harmonic sequences in accordance to tonal harmony rules with artificial neural networks. *Elektronika : konstrukcje, technologie, zastosowania*, 50(11):22–24, 2009.

Glosariusz

agogika - (z gr. agoge = prowadzenie, kierowanie), zajmuje się wszelkimi zjawiskami dotyczącymi tempa w utworze muzycznym, strona 8

akcenty metryczne - podkreślają dźwięki o pewnym okreslonym położeniu w stosunku do dźwięków sąsiednich i są punktem oparcia dla rytmu , strona 9

artykulacja - (z łac. articulation) element dzieła muzycznego wskazujący na sposób wydobywania kolejnych dźwięków (lub współbrzmień) i ich łączenia, nadając przez to wykonywanemu utworowi właściwy wyraz i charakter , strona 8

dominanta - akord zbudowany na V stopniu skali; dąży do rozwiązania na tonikę, dzięki dźwiękowi prowadzącemu (VII stopień skali) , strona 68

dynamika - (z gr. dynamis = siła) zajmuje się określeniem wszelkich zjawisk związanych z natężeniem siły dźwięków i jego zmianami w utworze muzycznym , strona 8

EEG Elektroencefalograf, urządzenie rejestrujące zespół słabych sygnałów elektrycznych zapisanych jednocześniej w różnych punktach głowy, wyrażających aktywność fal muzgowych , strona 40

frazowanie - (z gr. phrasis = zwrot) wyznaczanie odcinków dzieła muzycznego tworzących pewne całości wyrazowe , strona 8

frazy - elementy formy muzycznej. Fraza zawiera dwa, lub więcej motywów., strona 28

fuga - rodzaj techniki polifonicznej, o bardzo złożonej strukturze, wymagającej co najmniej dwóch głosów, strona 9

funkcja harmoniczna - znaczenie akordu zbudowanego na określonym stopniu skali durowej lub molowej w systemie dur-moll; wynika ze związku z innymi akordami, ze stopnia napięcia harmonicznego., strona 68

Glissando - oznaczenie płynnego przejścia od jednego dźwięku do drugiego po kolei poprzez wszystkie możliwe do zagrania na danym instrumencie dźwięki leżące między nimi., strona 20

harmonia - zgodne współbrzmienie dźwięków, strona 8

homofonia - rodzaj środka użytego w technice kompozytorskiej, w którym wszystkie głosy są całkowicie podporządkowane jednej melodii , strona 9

imitacja - rodzaj techniki polifonicznej, polegający na powtórzeniu melodii jednego głosu w innym , strona 9

interwał - odległość pomiędzy dwoma dźwiękami muzycznymi , strona 7

inwencja - rodzaj techniki polifonicznej, bazujący na imitacji i często zbliżony do uproszczonej fugi, pozbawionej niektórych elementów; nie posiada ściśle określonej konstrukcji, strona 9

kadencja - jest to formuła melodyczno-harmoniczna, również rytmiczna i dynamiczna, służąca do zakończenia utworu lub jego części, strona 68

kanon - rodzaj techniki polifonicznej, polegający ścisłej imitacji, w której melodię

jednego głosu powtarzają kolejno z jednakowym opóźnieniem pozostałe głosy, strona 9

kolorystyka dźwiękowa - element dzieła muzycznego związany ze środkami wykonawczymi utworu i pozwalający na dostrzeganie różnic pomiędzy dźwiękami o tej samej wysokości zaśpiewanymi bądź zagranymi przez różne głosy lub instrumenty, strona 8

medianta - poboczna funkcja harmoniczna zbudowana na III lub VI stopniu skali, strona 68

melodia - (z gr. meloidia = śpiew) element dzieła muzycznego definiujący szereg postępujących po sobie dźwięków o określonej długości i odległości między nimi oraz logicznie powiązanych wysokościach, strona 8

melodia chromatycznie rozszerzona - melodia utrzymana w tonacji majorowej lub minorowej, bazująca na dźwiękach gamowłaściwych oraz ich chromatycznych odmianach, strona 9

melodia diatoniczna - melodia utrzymana w tonacji majorowej lub minorowej, bazująca na dźwiękach gamowłaściwych, strona 9

metrum - (z gr. metron) jest to czynnik porządkoujący ugrupowania rytmiczne za pomocą regularnie powtarzających się akcentów metrycznych , strona 8

motywy - najmniejsze cząstki formy muzycznej. W muzyce dwa, lub więcej dźwięków muzycznych stanowiące pewną całość. Motywy mogą być przekształcane i ulegać wariacjom. Zwykle motyw opisany jest zespołem interwałów i rytmem., strona 28

okresy - elementy formy muzycznej. Okres składa się się z dwóch zdań muzycznych. Pierwsze to poprzednik, drugie następnik. Okres jest najmniejszym, samodzielnym utworem muzycznym., strona 28

polifonia - rodzaj środka użytego w technice kompozytorskiej, w którym równocześnie dwa lub więcej dźwięków prowadzonych jest niezależnie od siebie , strona 9

przewrót - przeniesienie najniższego dźwięku akordu o oktawę w górę nad pozostałe dźwięki; w opisywanym systemie autor dokonuje przeniesienia prymy - jest to tzw. pierwszy przewrót, strona 87

rytm - rytm jest czynnikiem regulującym następstwo dźwięków w utworze muzycznym w czasie i organizującym je w pewne określone ugrupowania, strona 8

skala durowa harmoniczna - skala durowa naturalna z obniżonym VI stopniem, strona 120

skala durowa miękka - skala durowa naturalna z obniżonymi stopniami VI i VII, strona 120

skala durowa naturalna - siedmiostopniowa skala muzyczna, w której sekundy małe (półtony) zawarte są tylko pomiędzy stopniami III-IV i VII-VIII. Pomiędzy pozostałymi stopniami skali występują sekundy wielkie (całe tony), strona 120

skala molowa dorycka - skala molowa naturalna z podwyższonym stopniem VI i VII, strona 120

skala molowa harmoniczna - skala molowa naturalna z podwyższonym VII stopniem, strona 120

skala molowa naturalna (eolska) - siedmiostopniowa skala muzyczna, w której sekundy małe zawarte są tylko pomiędzy stopniami II-III i V-VI. Pomiędzy pozostałymi stopniami skali występują sekundy wielkie, strona 120

staccato - skracanie dźwięków, mniej więcej o połowę ich wartości, poprzez ostre ich wydobywanie i oddzielanie od siebie, strona 129

struktura formalna - powstała przez współdziałanie wszystkich elementów dzieła muzycznego forma struktry tego dziełam, w której można wyodrębnić pewne części podobne i kontrastowe , strona 8

subdominanta - akord zbudowany na IV stopniu skali; jeden z trzech głównych stopni gamy obok toniki i dominanty, strona 68

submedianta - poboczna funkcja harmoniczna zbudowana na VI stopniu skali; inna nazwa tej funkcji to medianta dolna lub medianta VI stopnia , strona 68

symbol startowy - wyróżniony symbol nieterminalny, od którego, poprzez zastosowanie reguł produkcji, zaczyna się generowanie wszystkich wyrazów języka formalnego., strona 28

takt odcinek tekstu muzycznego zawierający ściśle określoną sumę wartości nut i pauz, strona 7

tempo - jest to czynnik wpływający na długość wartości rytmicznnych występujących w dziele muzycznym , strona 8

tonika - akord toniczny zbudowany na I stopniu skali, stosowany na ogół w postaci pierwotnej, tj. trójdzwięku; rozładowuje napięcie tonalne; tonika jako stopień skali najczęściej rozpoczyna i kończy melodię , strona 68

zbiór reguł produkcji słowa - zbiór reguł w postaci ciągów symboli terminalnch i nieterminalnych, określających możliwe podstawienia symboli w wyprowadzanym słowie., strona 28

zbiór symboli nieterminalnych - zbiór symboli, które można definiować. Symbole nieformalne umożliwiają tworzenie ciągów zawierających kombinacje symboli terminalnych i nieterminalnych., strona 28

zbiór symboli terminalnych - zbiór elementarnych symboli tworzących wyrazy języka formalnego. Symbole terminalne to znaki, które mogą pojawić się na wejściu lub wyjściu z reguły produkcji gramatyki formalnej. Symbole terminalne nie mogą być zmieniane przez reguły gramatyki formalnej., strona 28

zdania muzyczne - elementy formy muzycznej. Zdanie zawiera dwie frazy i obejmuje zazwyczaj cztery takty, strona 28

Spis rysunków

1.1 Regularny podział wartości rytmicznych. Źródło [55]. 6
1.2 Skale systemu modalnego. Opracowanie własne. 8
1.3 Podstawowe reprezentacje muzyki. Źródło: [26]. 11
1.4 Pianolowy wykres (góra) oraz zapis nutowy (dół) macierzy z tab. 1.4. Opracowanie własne. 15
1.5 Wykres pianolowy przykładowej serii nut (kolor czerwony) oraz serii przetransponowanej o 3 półtony (kolor niebieski). Opracowanie własne. 17
1.6 Wykres pianolowy przykładowej serii nut (kolor czerwony) oraz serii z odwróconą wysokością dźwięków (kolor niebieski). Opracowanie własne. 18
1.7 Wykres pianolowy przykładowej serii nut (kolor czerwony) oraz serii odwrotnej (kolor niebieski). Opracowanie własne. 18
1.8 Wykres pianolowy przykładowej serii nut (kolor czerwony) oraz serii przetransformowanej z wykorzystaniem metody backwards upside-down (kolor niebieski). Opracowanie własne. 19
1.9 Wykres pianolowy przykładowej serii nut (kolor czerwony) oraz serii przetransformowanej z wykorzystaniem metody backwards upside-down (kolor niebieski). Opracowanie własne. 20
1.10 Przykładowy wektor v_m. Opracowanie własne. 21
1.11 Wykres pianolowy wygenerowanej melodii dla $v_0 = \{60, 1, 64\}$ z wykorzystaniem przykładowej implementacji w środowisku Matlab. Opracowanie własne. 23
1.12 Wykres pianolowy fragmentu utworu “Pretty woman” Roya Orbisona. Opracowanie własne. 25
1.13 Wykres pianolowy przykładowej melodii wygenerowanej na podstawie macierzy z tab. 1.8. Opracowanie własne. 27
1.14 Pianolowy wykres kompozycji wyprodukowanej wg reguły ZS_1. Opracowanie własne. 30
1.15 Wykres pianolowy kompozycji wygenerowanej wg przykładowej reguły transformacji. Opracowanie własne. 31
1.16 Prosty automat skończony przedstawiony za pomocą diagramu UML maszyny stanów w programie Enterprise Architect. Opracowanie własne. 32
1.17 Przykład automatu skończonego generującego sekwencje rytmiczne. Opracowanie własne. 32
1.18 Wykresy pianolowe sekwencji rytmicznych definiujących operatory krawędzi a, b, c i d. Opracowanie własne. 33
1.19 Wykres pianolowy przykładowej sekwencji rytmicznej wygenerowanej przez zaimplementowany automat skończony. Opracowanie własne. 34
1.20 Typowy przebieg algorytmu genetycznego przedstawiony za pomocą diagramu maszyny stanów UML. Opracowanie własne. 35
1.21 Przykład 12-sto bitowego łańcucha kodującego trzy parametry dźwięku muzycznego: wysokość, dynamikę i czas trwania. Opracowanie własne. 36

1.22 Przykładowe siatki automatów komórkowych generujących składowe partie instrumentalne na podstawie tablicy *lut2* (A) i *lut* (B). Opracowanie własne. . 39
1.23 Płaszczyzna pozycjonowania algorytmów komponowania muzyki zaproponowana przez Woollera z ulokowanym autorskim rozwiązaniem (GEBMS). Opracowanie własne. 42

2.1 Ogólna architektura generatywnego systemu algorytmicznego komponowania muzyki na podstawie wzorców. Opracowanie własne. 53
2.2 Ogólna architektura generatywnego systemu algorytmicznego komponowania muzyki tła - podejście bez definiowania formy kompozycji. Opracowanie własne. 54
2.3 Wykresy pianolowe przykładowego wzorca o długości 4 bpm i wartości długości nuty: 1 bpm (A), 0.5 bpm (B), 0.25 bpm (C) oraz 0.125 bpm (D). Opracowanie własne. 55
2.4 Przykładowa architektura generowania wzorców. Opracowanie własne. 64
2.5 Przykładowe macierze PST dla 64-ech (A) i 256-ciu (B) wygenerowanych wzorców muzycznych. Opracowanie własne 65
2.6 Ogólna architektura generatywnego systemu algorytmicznego komponowania muzyki tła - podejście z uwzględnieniem opisu formy generowanego utworu. Opracowanie własne. 67
2.7 Generowanie zmian akordów. Opracowanie własne. 68

3.1 Ekran ogólnych ustawień aplikacji SynthFont 96
3.2 Wykres pianolowy w aplikacji SynthFont 96

4.1 Wizualizacja macierzy KMM. Opracowanie własne. 98
4.2 Schemat generowania przykładowej architektury klas wzorców przez zdefiniowane mini-generatory Mg. Opracowanie własne. 101
4.3 Wykresy pianolowe dla dwóch przykładowych wzorców klasy A. Opracowanie własne. 102
4.4 Wykres pianolowy fragmentu kompozycji wygenerowanej na podstawie ustawień z tab. 4.4. Opracowanie własne. 104
4.5 Wykresy pianolowe fragmentów kompozycji $pattern_l - 4.flac$ (A) oraz $pattern_l - 32.flac$ (B) wygenerowanych na podstawie ustawień z tab. 4.5. Opracowanie własne. 106
4.6 Wykres pianolowy kompozycji $pattern_n16 - u16.flac$ - przykład zapętlenia repetycji wzorców. Opracowanie własne. 107
4.7 Wykresy pianolowe kompozycji $pattern_rt - 1 - 4.flac$ (A) i $pattern_rt - 1 - 2 - 4.flac$ (B) z zaznaczonymi obszarami występowania pojedynczych (linią przerywaną) i podwójnych (linią ciągłą) wzorców. Opracowanie własne. . 109
4.8 Schemat generowania architektury klas wzorców przez zdefiniowane mini-generatory Mg dla kompozycji $s1 - 415.flac$. Opracowanie własne. 112
4.9 Wykres pianolowy fragmentu kompozycji $s1 - 415.flac$ z zaznaczonymi grupami wzorców wykazujących podobieństwo budowy. Opracowanie własne. . 112
4.10 Schemat generowania architektury klas wzorców przez mini-generatory zdefiniowane w tab. 4.9. Opracowanie własne. 116
4.11 Fragment kompozycji wygenerowanej na podstawie ustawień ze strony 113. Opracowanie własne. 117
4.12 Wykresy pianolowe kompozycji z tab. 4.11. A - $f_1111.flac$, B - $f_4444.flac$, C - $f_1234.flac$ i D - $f_12344321.flac$. Opracowanie własne. 118

4.13 Samoorganizujące się mapy klas wysokości dźwięków ([95], [66]) dla kompozycji *s_mol_nat.flac* (A), *s_mol_har.flac* (B), *s_mol_dor.flac* (C), *s_dur_nat.flac* (D), *s_dur_har.flac* (E) i *s_dur_soft.flac* (F). Opracowanie własne. 121
4.14 Histogramy klas wysokości dźwięku wg modelu Parncutta'a [86] dla kompozycji *s_mol_nat.flac* (A), *s_mol_har.flac* (B) i *s_mol_dor.flac* (C). Opracowanie własne. 122
4.15 Histogramy klas wysokości dźwięku wg modelu Parncutta'a [86] dla kompozycji *s_dur_nat.flac* (A), *s_dur_har.flac* (B) i *s_dur_soft.flac* (C). Opracowanie własne. 123
4.16 Wykresy pianolowe kompozycji *pl_0.flac* (A), *pl_4.flac* (B) i *pl_0248.flac* (C) z zaznaczonymi widocznymi przebiegami partii wygenerowanych przez mini-generatory z tab. 4.9 na stronie 115. Opracowanie własne. 125
4.17 Wykresy pianolowe początkowych fragmentów kompozycji *po_4_om_005.flac* (A), *po_3_om_005.flac* (B), *po_0_om_2.flac* (C) i *po_024_om_24.flac* (D) z zaznaczeniem charakterystycznych cech struktur rytmicznych. Opracowanie własne. 127
4.18 Wykresy pianolowe kompozycji *pdm1_00125_pdm2_05.flac* (A), *pdm1_025_pdm2_00125.flac* (B), *pdm1_2_pdm2_05.flac* (C) i *pdm1_02524_pdm2_05.flac* (D) z zaznaczeniem charakterystycznych cech struktur rytmicznych. Opracowanie własne. 131
4.19 Wykresy pianolowe kompozycji *lp_2222_s_4888.flac* (A), *lp_8888_s_4888.flac* (B), *lp_4444_s_2444.flac* (C) i *lp_4444_s_8121212.flac* (D). Opracowanie własne. 133

5.1 Płaszczyzna pozycjonowania algorytmów komponowania muzyki Wollera w ujęciu autorskiego systemu algorytmicznego komponowania muzyki z uwzględnieniem ($S2$) oraz bez uwzględnienia ($S1$) formy kompozycji. Opracowanie własne. 137
5.2 Składowe oceny RG Score = 2.47 autora. Dane na dzień 06.03.2017. Opracowanie własne. 139
5.3 Ekran Song Structure z aplikacji "Virtual laboratory for musicologist". Opracowanie własne. 141
5.4 Ekran Pitch Class Distribution Histogram z aplikacji "Virtual laboratory for musicologist". Opracowanie własne. 142
5.5 Ekran Song Statistics z aplikacji "Virtual laboratory for musicologist". Opracowanie własne. 142
5.6 Ekran Global Statistics z aplikacji "Virtual laboratory for musicologist". Opracowanie własne. 143
5.7 Ekran Similarity z aplikacji "Virtual laboratory for musicologist". Opracowanie własne. 143

Spis tablic

1.1 Podstawowe nazwy interwałów wraz z przypadającymi dla nich: liczbą stopni i półtonów. Opracowanie własne. 7
1.2 Charakterystyka podstawowowych reprezentacji muzyki (na podstawie [26]). Opracowanie własne. 12
1.3 Reprezentacje tekstu w porównaniu z reprezentacją muzyki (na podstawie [26]). Opracowanie własne. 12
1.4 Przykładowa macierz nut w środowisku Matlab. Opracowanie własne. 13
1.5 Heksadecymalna postać pliku $MIDI$ wygenerowanego na podstawie macierzy nut z rys. 1.4. Opracowanie własne. 13
1.6 Format kanałowego komunikatu $MIDI$ (na podstawie [2]). Opracowanie własne. 14
1.7 Przykładowa tabela prawdopodobieństwa zajścia zdarzenia w systemie komponowania muzyki (na podstawie [85]). Opracowanie własne. 24
1.8 Macierz prawdopodobieństwa tranzycji łańcucha Markowa pierwszego rzędu dla przykładowego fragmentu utworu z rys 1.12. Opracowanie własne. 26
1.9 Podział systemów algorytmicznego komponowania muzyki ze względu na jawność i niejawność wykorzystanych reguł komponowania. Opracowanie własne. 46

2.1 Podstawowe zadania procesu komponowania (na podstawie [75]). Opracowanie własne. 50
2.2 Związek pomiędzy klasą K mini-generatora, rodzajem MM oraz zdefiniowaną regułą generowania wzorca. Opracowanie własne. 58
2.3 Macierz hierarchii tonalnych Krumhansl-Kesslera. Opracowanie własne. 61
2.4 Pozycja współczynnika korelacji R w kontekście tonacji muzycznej. Opracowanie własne. 62
2.5 Tabela TRPT z przykładowymi zdefiniowanymi regułami transformacyjnymi. Opracowanie własne. 62
2.6 Przykładowe kadencje oparte na tonacji C-dur. Opracowanie własne. 69
2.7 Macierz tranzycji generowania sekwencji akordów. Opracowanie własne. 69

4.1 Parametry zdefiniowanych mini-generatorów. Opracowanie własne. 100
4.2 Ustawienia barw instrumentów dla generowanych kompozycji z wykorzystaniem banku brzmień $Evanessence2.sf2$. Opracowanie własne. 103
4.3 Przyjęte długości dźwięków oraz długości wygenerowanych kompozycji. Opracowanie własne. 103
4.4 Zmodyfikowane długości dźwięków dla poszczególnych mini-generatorów. Opracowanie własne. 104
4.5 Przyjęte długości wzorców muzycznych oraz wygenerowanych kompozycji. Opracowanie własne. 105
4.6 Przyjęte wartości liczności zbioru wygenerowanych (n) i wykorzystanych (u) wzorców muzycznych. Opracowanie własne. 107
4.7 Przyjęte struktury tablicy rt. Opracowanie własne. 108

4.8 Parametry zdefiniowanych mini-generatorów dla kompozycji $s1 - 415.flac$. Opracowanie własne. 111
4.9 Parametry zdefiniowanych mini-generatorów. Opracowanie własne. 115
4.10 Instrumenty wykorzystane w kompozycji $k_raw_421.flac$. Opracowanie własne. 115
4.11 Przyjęte struktury wektora $form$. Opracowanie własne. 116
4.12 Budowa progresji akordów dla poszczególnych kompozycji. Opracowanie własne. 119
4.13 Przyjęte struktury wektora *scale*. Opracowanie własne. 120
4.14 Przyjęte struktury wektora *possible_location*. Opracowanie własne. 124
4.15 Przyjęte struktury wektorów $possible_offset$ i $offset_mel$. Opracowanie własne. 128
4.16 Przyjęte struktury parametrów $possible_duration_mel1$ i $possible_duration_mel2$. Opracowanie własne. 129
4.17 Przyjęte wartości parametrów $lp1$, $lp2$, $lp3$, $lp4$, $step1$, $step2$, $step3$ oraz $step4$. Opracowanie własne 132

5.1 Statystyki profilu autora na portalu ReasearchGate. Dane na dzień 06.03.2017. Opracowanie własne. 140

Dodatek A. Listingi kodów

List. 5.1. Transpozycja macierzy nut w środowisku Matlab. Opracowanie własne.

```
function [outNoteMatrix] = transp(inNoteMatrix, amount)
outNoteMatrix = inNoteMatrix;
outNoteMatrix(:,4) = outNoteMatrix(:,4) + amount;
```

List. 5.2. Inwersja wysokości nut macierzy nut w środowisku Matlab. Opracowanie własne.

```
function [outNoteMatrix] = backwards(inNoteMatrix)
outNoteMatrix = inNoteMatrix;
outNoteMatrix(:,4) = inNoteMatrix(end:-1:1,4);
```

List. 5.3. Pełna inwersja macierzy nut w środowisku Matlab. Opracowanie własne.

```
function [outNoteMatrix] = fullBackwards(inNoteMatrix)
outNoteMatrix = inNoteMatrix;
outNoteMatrix(:,4) = inNoteMatrix(end:-1:1,4);
outNoteMatrix(:,[2,7]) = inNoteMatrix(end:-1:1,[2,7]);
```

List. 5.4. Transformacja upside-down macierzy nut w środowisku Matlab. Opracowanie własne.

```
function [outNoteMatrix] = upsideDown(inNoteMatrix)
outNoteMatrix = inNoteMatrix;
diffNote = diff(inNoteMatrix(:,4));
for  i=2:size(outNoteMatrix,1)
        outNoteMatrix(i,4) = outNoteMatrix(i-1,4) -
           diffNote(i-1);
end
```

List. 5.5. Transformacja backwards upside-down macierzy nut w środowisku Matlab. Opracowanie własne.

```
function [outNoteMatrix] = backwardsUpsideDown(
   inNoteMatrix)
buffNoteMatrix = fullBackwards(inNoteMatrix);
outNoteMatrix = upsideDown(buffNoteMatrix);
```

List. 5.6. Przykładowa implementacja generatorów wektorów oraz zdefiniowanych wyrażeń F_1, F_2 i F_3 w środowisku Matlab. Opracowanie własne.

```
function [melody] = musicVectors(v0)
x=-18; y=0.5; z=1;
outF1 = F1(v0); outF2 = F2(x,y,z,v0); outF3 = F3(x,y,z,v0);
melodyMatrix = cat(1,outF1,outF2,outF3);
melody = createnmat(melodyMatrix(:,1)',v0(2));
        for i=1:length(melody)
                melody(i,:) = setvalues(melody(i,:), 'dur', melodyMatrix(i,2));
                melody(i,:) = setvalues(melody(i,:), 'vel', melodyMatrix(i,3));
                if (i>1)
                melody(i,:) = setvalues(melody(i,:), 'onset', melodyMatrix(i-1,2) + melody(i-1,1) );
                end
        end

function [outF1] = F1(v0)
%F1 = [v1]
outF1 = [14+v0(1), 4*v0(2), v0(3)]; return

function [outF2] = F2(x,y,z,v0)
%F2 = [v2, v2, 2*v2]
v2 = [(x+12)+v0(1), y*v0(2), z*v0(3)];
outF2 = [v2; v2; 2*v2]; return

function [outF3] = F3(x,y,z,v0)
%F3 = [v3, v1 + v3, v3]
v3 = [(x-4)+v0(1), y*v0(2), z*v0(3)];
v1 = [14+v0(1), 4*v0(2), v0(3)];
outF3 = [v3; v1 + v3; v3];  return
```

List. 5.7. Mechanizm kumulatywnego wyboru procedury transformującej wejściową macierz nut w środowisku Matlab. Opracowanie własne.

```
function [outNoteMatrix] = cumChoice(inNoteMatrix, p)
choice = rand(1);
outNoteMatrix=[];
        if choice < p(1)
          outNoteMatrix = transp(inNoteMatrix, 8);
        end
        if choice < p(1) + p(2)
          outNoteMatrix = backwards(inNoteMatrix);
        end
        if choice < p(3) + p(2) + p(1)
          outNoteMatrix = fullBackwards(inNoteMatrix);
        end
        if choice < p(4) + p(3) + p(2) + p(1)
          outNoteMatrix = upsideDown(inNoteMatrix);
        else
          outNoteMatrix = backwardsUpsideDown(
             inNoteMatrix);
        end
```

List. 5.8. Implementacja algorytmu generowania macierzy tranzycji dla łańcucha Markowa pierwszego rzędu w środowisku Matlab. Opracowanie własne.

```
1  function [transMatrix, uniqueStates] = markov1Order(
      inputNoteMatrix)
2  uniqueStates = unique(inputNoteMatrix(:,4));
3  sizeTransMatrix = length(uniqueStates);
4  transMatrix = zeros(sizeTransMatrix,sizeTransMatrix);
5          for i = 1 : sizeTransMatrix
6          statesLocation = find(inputNoteMatrix(:,4) ==
              uniqueStates(i)) - 1;
7          statesLocation = statesLocation(statesLocation(:)
               > 0);
8          findStates = inputNoteMatrix(statesLocation,4);
9                  for j = 1 : sizeTransMatrix
10                         isFind = findStates( findStates
                              == uniqueStates(j));
11                         if ( ~isempty(isFind) )
12                         transMatrix(j,i) = length(isFind)
                              ;
13                         end
14                 end
15         end
16         for i=1:sizeTransMatrix
17                 transMatrix(i,:) = transMatrix(i,:) / sum
                      (transMatrix(i,:));
18         end
```

List. 5.9. Implementacja systemu generującego kompozycje na podstawie zdefiniowanej gramatyki formalnej w środowisku Matlab. Opracowanie własne.

```
1  I = [1:13];
2  K = ['/','\','-'];
3  S = cell(1,2);
4  S{1} = ['(' num2str(I(5)) ',' K(1) ')' ...
5  '+' '(' num2str(I(8)) ',' K(1) ')' ...
6  '+' '(' num2str(I(11)) ',' K(1) ')'];
7  S{2} = ['(' num2str(I(5)) ',' K(2) ')' ...
8  '+' '(' num2str(I(8)) ',' K(2) ')'];
9  ZS = cell(1,1);
10 ZS{1} = [S{1} '+' S{2}];
11 D = 60;
12 noteOutputVector = [];
13 for j=1:size(ZS,1)
14         pairs = find(ZS{j} == ',');
15         begins = find(ZS{j} == '(');
16         noteVector = zeros(size(pairs));
17         for i=1:length(begins)
18                 note = ZS{1}(begins(i)+1:pairs(i)-1);
19                 if ( ZS{1}(pairs(i)+1) == K(1) )
20                         noteVector(i) = D + str2num(note)
                               ;
21                 end
22                 if ( ZS{1}(pairs(i)+1) == K(2) )
23                         noteVector(i) = D - str2num(note)
                               ;
24                 end
25                 if ( ZS{1}(pairs(i)+1) == K(3) )
26                         noteVector(i) = D;
27                 end
28         end
29         noteOutputVector = [noteOutputVector, noteVector
               ];
30         noteVector = [];
31 end
32 seqOut = createnmat(noteOutputVector);
33 pianoroll(seqOut,'num');
```

List. 5.10. Zdefiniowanie operatorów krawędzi a(pattern1), b(pattern2), c(pattern3) i d(pattern4) w środowisku Matlab. Opracowanie własne.

```
basepattern = createnmat(zeros(1,16),0.125,127,10);
pattern1 = basepattern;
pattern1([1,5,9,13],4) = [35,38,35,38];
pattern2 = basepattern;
pattern2([1,5,9,11,13],4) = [35,38,35,35,38];
pattern3 = basepattern;
pattern3([1,5,9,11,13],4) = [35,38,35,38,38];
pattern4 = basepattern;
pattern4([1,5,9,10,11,12,13],4) = [35,38,35,38,38,35,49];
pattern4(13,:) = shift(pattern4(13,:),'dur',pattern4
   (13,2)*3,'sec');
```

List. 5.11. Implementacja przykładowego automatu skończonego generującego zdefiniowane sekwencje rytmiczne w środowisku Matlab. Opracowanie własne.

```
1  sequence=[];
2  state=1;
3  while (1)
4    if (state==1)
5      state=2;
6          if (~isempty(sequence))
7            sequence = [sequence; shift(pattern1,'onset',
               sequence(end,1) + sequence(end,2))];
8          else
9            sequence = [sequence; pattern1];
10         end
11   end
12   if (state==2)
13     choice = rand(1);
14     if choice < 0.33
15       sequence = [sequence; shift(pattern2,'onset',
           sequence(end,1) + sequence(end,2))];
16       state=1;
17         end
18     if choice < 0.66
19       sequence = [sequence; shift(pattern3,'onset',
           sequence(end,1) + sequence(end,2))];
20       state=1;
21         end
22         if choice < 1
23           sequence = [sequence; shift(pattern4,'onset',
               sequence(end,1) + sequence(end,2))];
24           state=3;
25         end
26   end
27   if (state==3) break end
28 end
29 sequence = sequence(sequence(:,4)>0,:);
```

List. 5.12. Implementacja modelu generującego kompozycje bazując na analizie siatek dwóch automatów komórkowych. Opracowanie własne.

```
1  siatkaWymiar=50;
2  generacje = 10;
3  lut = [35:1:59, 59:-1:35];
4  lut2=[24,26,28,29,31,33,36,38,40,41,
5  43,45,48,50,52,53,55,57,60,62,
6  64,65,67,69,69,67,65,64,62,60,
7  48,50,52,53,55,57,45,43,41,40,
8  38,36,33,31,29,28,26,24,60,24];
9  kompozycja=[];
10 siatka = zeros(siatkaWymiar,siatkaWymiar);
11 siatka2 = siatka;
12 siatka(fix(rand(20,1)*siatkaWymiar)+1,fix(rand(30,1)*
      siatkaWymiar)+1)=1;
13 siatka2(fix(rand(10,1)*siatkaWymiar)+1,fix(rand(30,1)*
      siatkaWymiar)+1)=1;
14 tempsiatka = zeros(siatkaWymiar,siatkaWymiar);
15 tempsiatka2 = tempsiatka;
16 offset=0;
17 for x=1:generacje
18         for i=1:siatkaWymiar
19                 for j=1:siatkaWymiar
20                 zywe = skanowanie(i,j,siatka);
21                 tempsiatka(i,j) = reguly(zywe,siatka(i,j)
                      );
22                 zywe = skanowanie(i,j,siatka2)
23                 tempsiatka2(i,j) = reguly(zywe,siatka2(i,
                      j));
24     end
25 end
26 siatka=tempsiatka;
27 siatka2=tempsiatka2;
28 tempsiatka=zeros(siatkaWymiar,siatkaWymiar);
29 tempsiatka2=tempsiatka;
30 [kompozycja]=generowanieKompozycji(kompozycja,siatka,
      offset,lut2,1,0.2,1.2);
31 [kompozycja,offset]=generowanieKompozycji(kompozycja,
      siatka2,offset,lut,10,0.2,1.2);
32 end
33 writemidi(kompozycja,'automatKomorkowy.mid',120,100);
```

List. 5.13. Implementacja analizy sąsiedztwa Moore'a w przykładowym modelu automatu komórkowego. Opracowanie własne.

```
1  function[zywe] = skanowanie(i,j,siatka)
2  zywe = 0;
3  siatkaWymiar = size(siatka,1);
4  if (i==1 && j==1)
5          zywe = siatka(i,j+1) + siatka(i+1,j) + siatka(i
             +1,j+1);
6  end
7  if (i==1 && j>1 && j<siatkaWymiar)
8  zywe = siatka(i,j-1) + siatka(i,j+1) + siatka(i+1,j-1) +
     siatka(i+1,j) + siatka(i+1,j+1);
9  end
10 if (i==1 && j==siatkaWymiar)
11 zywe = siatka(i,j-1) + siatka(i+1,j-1) + siatka(i+1,j);
12 end
13 if (i>1 && i<siatkaWymiar && j==1)
14 zywe = siatka(i-1,j) + siatka(i-1,j+1) + siatka(i,j+1) +
     siatka(i+1,j) + siatka(i+1,j+1);
15 end
16 if (i>1 && i<siatkaWymiar && j>1 && j<siatkaWymiar)
17 zywe = siatka(i-1,j-1) + siatka(i-1,j) + siatka(i-1,j+1)
     + siatka(i,j-1) + siatka(i,j+1) + siatka(i+1,j-1) +
     siatka(i+1,j) + siatka(i+1,j+1);
18 end
19 if (i>1 && i<siatkaWymiar && j==siatkaWymiar)
20 zywe = siatka(i-1,j-1) + siatka(i-1,j) + siatka(i,j-1) +
     siatka(i+1,j-1) + siatka(i+1,j);
21 end
22 if (i==siatkaWymiar && j==1)
23 zywe = siatka(i-1,j) + siatka(i-1,j+1) + siatka(i,j+1);
24 end
25 if (i==siatkaWymiar && j>1 && j<siatkaWymiar)
26 zywe = siatka(i-1,j-1) + siatka(i-1,j) + siatka(i-1,j+1)
     + siatka(i,j-1) + siatka(i,j+1);
27 end
28 if (i==siatkaWymiar && j==siatkaWymiar)
29 zywe = siatka(i-1,j-1) + siatka(i-1,j) + siatka(i,j-1);
30 end
```

List. 5.14. Implementacja funkcji reguly w przykładowym modelu automatu komórkowego. Opracowanie własne.

```
function[nowyStanKomorki] = reguly(liczbaZywychSasiadow,
   poprzedniStanKomorki)
nowyStanKomorki=0;
if (liczbaZywychSasiadow==3)
nowyStanKomorki=1;
end
if (  ((liczbaZywychSasiadow==2) || (liczbaZywychSasiadow
   ==3)) && (poprzedniStanKomorki==1))
nowyStanKomorki=1;
end
```

List. 5.15. Implementacja funkcji interpretującej siatkę komórek w przykładowym modelu automatu komórkowego. Opracowanie własne.

```
function [kompozycja,offset] = generowanieKompozycji(
   kompozycja,siatka,offset,lut,kanalMidi,dlugosc,delta)
for i=1:size(siatka,1)
buffor = createnmat(lut.*siatka(:,i)',dlugosc,100,
   kanalMidi);
buffor = buffor(buffor(:,4)>0,:);
if (~isempty(buffor))
buffor = shift(buffor,'onset',offset,'sec');
kompozycja = [kompozycja;buffor];
offset=offset+delta;
end
end
```

List. 5.16. Przykład generowania kompozycji bez opisu formy utoworu w środowisku Matlab. Opracowanie własne.

```
1  clc
2  clear all
3  pattern_l = 16;
4  n = 32; %liczba generowanych wzorców
5  seq_patterns = 128; %liczba wzorców wykorzystanych w
      kompozycji
6  basepattern = createnmat(zeros(1,pattern_l),0.125,127,10)
      ;
7  bass_pattern = createnmat(zeros(1,pattern_l),0.25,127,1);
8  bass_pattern2 = createnmat(zeros(1,pattern_l),0.25,127,2)
      ;
9  bass_pattern3 = createnmat(zeros(1,pattern_l),0.25,127,3)
      ;
10 bassoctave = KMM();
11 [rhytmic_step, rhytmic_value] = rhytmicStructure3(
      pattern_l);
12 temppattern = basepattern;
13 pcdistribution = zeros(n,12);
14 pattern_struct = [];
15 CC_size = size(bassoctave,1);
16 %generowanie wzorców i definiowanie klas wzorców
17 for i=1:n
18 %bass
19         step = fix( pattern_l/ fix(rand(1)*rhytmic_step
              (1)+1) );
20         temppattern(rhytmic_value{1}(1:step:end) , 4) =
              36;
21         bass = temppattern;
22 %snare
23         temppattern = basepattern;
24     step = fix( pattern_l/ fix(rand(1)*rhytmic_step(2)+1)
          );
25         temppattern(rhytmic_value{2}(1:step:end) , 4) =
              40;
26         snare = temppattern;
27 %closed hihat
28         temppattern = basepattern;
29         step = fix( pattern_l/ fix(rand(1)*rhytmic_step
              (3)+1) );
30     temppattern(rhytmic_value{3}(1:step:end) , 4) = 42;
31     closedhihat = temppattern;
32 %low mid tom
33         temppattern = basepattern;
34         step = fix( pattern_l/ fix(rand(1)*rhytmic_step
              (4)+1) );
35     temppattern(rhytmic_value{4}(1:step:end) , 4) = 47;
```

```
36          low_mid_tom = temppattern;
37  %high mid tom
38          temppattern = basepattern;
39          step = fix( pattern_l/ fix(rand(1)*rhytmic_step
               (5)+1) );
40          temppattern(rhytmic_value{5}(1:step:end) , 4) =
               48;
41          high_mid_tom = temppattern;
42  %tambourine
43          temppattern = basepattern;
44          step = fix( pattern_l/ fix(rand(1)*rhytmic_step
               (6)+1) );
45          temppattern(rhytmic_value{6}(1:step:end) , 4) =
               54;
46          tambourine = temppattern;
47  %ornaments
48          temppattern = createnmat(zeros(1,pattern_l)
               ,0.25,127,10);
49          step = fix( pattern_l/ fix(rand(1)*rhytmic_step
               (7)+1) );
50          vec = rhytmic_value{8}(1:step:end);
51          temppattern(vec,4) = [60:1:60+length(vec)-1];
52          harmonization = temppattern;
53          temppattern = createnmat(zeros(1,pattern_l*2)
               ,0.25,127,10);
54          step = fix( pattern_l/ fix(rand(1)*rhytmic_step
               (7)+1) );
55          vec = rhytmic_value{7}(1:step:end);
56          temppattern(vec,4) = [60:1:60+length(vec)-1];
57          harmonization2 = temppattern;
58          temppattern = basepattern;
59  %fill
60          if ( mod(i,2) == 1)
61                  temppattern = createnmat(zeros(1,
                       pattern_l*2),0.25,127,10);
62                  step = fix( pattern_l/ fix(rand(1)*
                       rhytmic_step(7)+1) );
63                  temppattern(rhytmic_value{7}(1:step:end)
                       , 4) = 57;
64                  fill = temppattern;
65          else
66                  fill = [];
67          end
68  %mini-generator monofoniczny 1
69          temppattern = bass_pattern;
70          step = fix( pattern_l/ fix(rand(1)*rhytmic_step
               (9)+1) );
71          vec = rhytmic_value{9}(1:step:end);
```

```
72          mm_index = fix(rand(1)*CC_size+1);
73          mm = bassoctave(mm_index,:);
74          basspiano_vec = zeros(1,length(vec));
75          k=1;
76          for j=1:length(vec)
77                  basspiano_vec(j) = mm(k);
78                  if (k < length(mm))
79                          k=k+1;
80                  else
81                          k=1;
82                  end
83          end
84          if ( mod(i,2) == 1)
85                  basspiano_vec = basspiano_vec + 12;
86          end
87          temppattern(vec , 4) = basspiano_vec;
88          basspiano = temppattern;
89  %mini-generator monofoniczny 2
90          temppattern = bass_pattern2;
91          step = fix( pattern_l/ fix(rand(1)*rhytmic_step
               (9)+1) );
92          vec = rhytmic_value{10}(1:step:end);
93          if (mm_index<CC_size)
94                  mm_index=mm_index+1;
95          else
96                  mm_index=mm_index-1;
97          end
98          mm = bassoctave(mm_index,:);
99          basspiano_vec = zeros(1,length(vec));
100         k=1;
101         for j=1:length(vec)
102                 basspiano_vec(j) = mm(k);
103                 if (k < length(mm))
104                         k=k+1;
105                 else
106                         k=1;
107                 end
108         end
109         if ( mod(i,2) == 1)
110                 basspiano_vec = basspiano_vec + 12;
111         end
112         temppattern(vec , 4) = basspiano_vec;
113         basspiano1 = temppattern;
114         basspiano1(:,3) = 2;
115 %mini-generator monofoniczny 3
116         temppattern = bass_pattern3;
117         step = fix( pattern_l/ fix(rand(1)*rhytmic_step
               (9)+1) );
```

```
118         vec = rhytmic_value{9}(1:step:end);
119         if (mm_index<3)
120                 mm_index=mm_index+2;
121         else
122                 mm_index=mm_index-2;
123         end
124         mm = bassoctave(mm_index,:);
125         basspiano_vec = zeros(1,length(vec));
126         k=1;
127         for j=1:length(vec)
128                 basspiano_vec(j) = mm(k);
129                 if (k < length(mm))
130                         k=k+1;
131                 else
132                         k=1;
133                 end
134         end
135         if ( mod(i,2) == 1)
136                 basspiano_vec = basspiano_vec + 12;
137         end
138         temppattern(vec , 4) = basspiano_vec;
139         basspiano2 = temppattern;
140         basspiano2(:,3) = 3;
141         if ( (i/n*100) <= 25 )
142                 pattern = [bass; snare; closedhihat;
                       low_mid_tom; high_mid_tom; tambourine;
                        harmonization; basspiano; basspiano1;
                        basspiano2 ];
143         end
144         if ( (i/n*100) > 25 && (i/n*100) <= 50 )
145         pattern = [bass; snare; closedhihat; low_mid_tom;
146         high_mid_tom; tambourine; basspiano2; basspiano1
               ];
147         end
148         if ( (i/n*100) > 50 && (i/n*100) <= 75 )
149                 pattern = [bass; snare; closedhihat;
                       low_mid_tom; high_mid_tom; tambourine;
                        fill; basspiano; basspiano1;
                       basspiano2 ];
150         end
151         if ( (i/n*100) > 75 && (i/n*100) <= 100 )
152                 pattern = [bass; closedhihat; tambourine;
                        fill; basspiano ];
153         end
154         pattern = pattern(pattern(:,4)>0,:);
155         pattern = sortrows(pattern);
156         pattern_struct{i}.data = pattern;
157         pcdistribution(i,:) = pcdist1(pattern);
```

```
158         temppattern = basepattern;
159 end
160 %macierz ńpodobiestwa
161 pattern_differences = [];
162 for i=1:n
163         pattern_to = repmat(pcdistribution(i,:),n,1);
164         d =  sqrt(sum((pattern_to - pcdistribution).^2));
165         e = [sum(d,2) , [1:n]'];
166         pattern_differences{i} = sortrows(e);
167 end
168 %generowanie sekwencji
169 sequence = [];
170 shift_value = basepattern(end,1) + basepattern(end,2);
171 %%lista rankingowa
172 for i=1:seq_patterns
173         if (i==1)
174                 state =  fix(rand(1)*n+1);
175                 temppattern = pattern_struct{state}.data;
176         end
177         if (i>1)
178         %wybór tylko 40% najbardziej podobnych wzorców
179                 choice = fix(rand(1)*0.40*n + 1);
180                 state = pattern_differences{state}(choice
                       ,2);
181                 temppattern = pattern_struct{state}.data;
182                 temppattern = shift(temppattern, 'onset',
                        shift_value);
183                 shift_value = shift_value + basepattern(
                       end,1) + basepattern(end,2);
184         end
185         sequence = [sequence; temppattern];
186 end
187 writemidi_java(sequence,'sequence.mid',120,100);
```

List. 5.17. Funkcja generująca zbiór motywów KMM w środowisku Matlab. Opracowanie własne.

```
function [bassoctave] = KMM()
bassoctave = perms( [24, 26, 28, 31, 33, 36] );
bassoctave = [bassoctave; perms( [36, 38, 40, 43, 45, 48]
    ) ];
bassoctave = [bassoctave; perms( [24, 26, 28, 31, 48, 52]
    ) ];
bassoctave = [bassoctave, round(bassoctave+12)];
```

List. 5.18. Funkcja generująca wektory WPN dla przykładowych zdefiniowanych partii insturmentów w środowisku Matlab. Opracowanie własne.

```
function [rhytmic_step, rhytmic_value] =
   rhytmicStructure3(pattern_l)
bass_step = pattern_l/4;
snare_step = pattern_l/4;
closedhihat_step = pattern_l/2;
low_mid_tom_step = pattern_l/2;
high_mid_tom_step = pattern_l/4;
tambourine_step = pattern_l/4;
fill_step = pattern_l/4;
harmonization_step = pattern_l/3;
basspiano_step = pattern_l/2;
basspiano2_step = pattern_l/2;
clap_step = pattern_l/2;
rhytmic_step = [bass_step; snare_step; closedhihat_step;
   ...
low_mid_tom_step; high_mid_tom_step; tambourine_step; ...
fill_step; harmonization_step; basspiano_step;
   basspiano2_step; clap_step ];

bass_value = [1:4:pattern_l];
bass_value = unique(bass_value);
snare_value = [2:3:pattern_l];
closedhihat_value = [1:3:pattern_l];
low_mid_tom_value = [fix(pattern_l/2):2:pattern_l];
high_mid_tom_value = [fix(pattern_l/2):2:pattern_l];
tambourine_value = [1:4:pattern_l];
fill_value = [fix(pattern_l/2)+1:4:pattern_l];
harmonization_value = [1:3:pattern_l];
basspiano_value = [1:1:pattern_l];
basspiano2_value = [2:1:pattern_l/2];
clap_value = [1:3:pattern_l];

rhytmic_value{1} = bass_value;
rhytmic_value{2} = snare_value;
rhytmic_value{3} = closedhihat_value;
rhytmic_value{4} = low_mid_tom_value;
rhytmic_value{5} = high_mid_tom_value;
rhytmic_value{6} = tambourine_value;
rhytmic_value{7} = fill_value;
rhytmic_value{8} = harmonization_value;
rhytmic_value{9} = basspiano_value;
rhytmic_value{10} = basspiano2_value;
rhytmic_value{11} = clap_value;
```

List. 5.19. Przykład generowania kompozycji z opisem formy utworu w środowisku Matlab. Opracowanie własne.

```
1  clc
2  clear all
3  %forma
4  form = [1,1,3,3,2,2,1,1,4,4,3,3,2,2,1,1,4,4,2,2,1,1];
5  tempo = 120;
6  %odmiany skali durowej
7  dur_naturalna = [0,2,4,5,7,9,11,12];
8  durowa_harmoniczna = [0,2,4,5,7,8,11,12];
9  durowa_miekka = [0,2,4,5,7,8,10,12];
10 %odmiany skali molowej
11 mol_naturalna = [0,2,3,5,7,8,10,12];
12 mol_harmoniczna = [0,2,3,5,7,8,11,12];
13 mol_dorycka = [0,2,3,5,7,9,11,12];
14 skala = mol_naturalna ;
15 p = createnmat(skala+60);
16 chord = [p(1,:); shift(p(1,:),'pitch',skala(3)); shift(p
       (1,:),'pitch',skala(5)); ];
17 time=0;
18 composition = [];
19 part1=[];
20 part2=[];
21 part3=[];
22 part4=[];
23 possible_location = [0,8,8,16];
24 possible_location_snare = [0,8,16];
25 possible_offset = [0,4];
26 possible_offset_snare = [4,8,8];
27 possible_duration = [1,0.5];
28 lpos = length(possible_location);
29 lpos_snare = length(possible_location_snare);
30 lof = length(possible_offset);
31 lof_snare = length(possible_offset_snare);
32 %mel1
33 possible_duration_mel1 = [0.5,0.5];
34 offset_mel = [0,2];
35 %mel2
36 possible_duration_mel2 = [0.25,0.5];
37 offset_mel2 = [0,0.25,1];
38 lp1=4;
39 step1=4;
40 lp2=4;
41 step2=4;
42 lp3=4;
43 step3=4;
44 lp4=4;
45 step4=4;
```

```
46 fill = [36,50,48,47,45,43,40,49,57,66,65,40,39];
47 lfill = length(fill);
48 shift_chords=0;
49 l_part1 = 0;
50 l_part2 = 0;
51 l_part3 = 0;
52 l_part4 = 0;
53 offset_part1 = 0;
54 offset_part1_begin = 0;
55 for i=1:length(form)
56   %petla po strukturze utworu
57   if (form(i)==1)
58     if isempty(part1)
59           [chordprogression,shift_value] = progresje(chord,
               step1,offset_part1, 0,lp1,skala);
60           chordprogression = setvalues(chordprogression,'dur
               ',lp1,'beat');
61           shift_chords = chordprogression(end,1)+
               chordprogression(end,2);
62           shift_value = shift_chords+offset_part1;
63           l_part1 = shift_value;
64           %przewrot
65           if (rand < 0.3)
66             chordprogression(1,4) = chordprogression(1,4) +
                 12;
67           end
68           chordprogression = shift(chordprogression,'onset',
               offset_part1_begin,'beat');
69           melody = makemelody(chordprogression,lp1,2,
               possible_duration_mel1, offset_mel,1);
70           bass =        minigenerator('D',36,
               possible_location(floor(rand(1)*lpos+1)),
               possible_offset(floor(rand(1)*lof+1)),0.25,
               l_part1,tempo,10);
71           if (~isempty(bass))
72             bass=shift(bass,'onset',-offset_part1_begin,'
                 beat');
73           end
74           snare = minigenerator('D',40,
               possible_location_snare(floor(rand(1)*
               lpos_snare +1)),possible_offset_snare(floor(
               rand(1)*lof_snare+1)),0.25,l_part1,tempo,10);
75           if (~isempty(snare))
76             snare=shift(snare,'onset',-offset_part1_begin,'
                 beat');
77           end
78           hihat =        minigenerator('D',42,
               possible_location(floor(rand(1)*lpos+1)),
```

```
                possible_offset(floor(rand(1)*lof+1)),0.25,
                l_part1,tempo,10);
79            if (~isempty(hihat))
80              hihat=shift(hihat,'onset',-offset_part1_begin,'
                  beat');
81            end
82            clap = minigenerator('D',51,possible_location(
                floor(rand(1)*lpos+1)),possible_offset(floor(
                rand(1)*lof+1)),0.25,l_part1,tempo,10);
83            if (~isempty(clap))
84              clap=shift(clap,'onset',-offset_part1_begin,'
                  beat');
85            end
86            part1 = [melody;bass;snare;clap;hihat;shift(
                chordprogression,'pitch',-12)];
87            if isempty(composition)
88              composition = [composition;part1];
89              time=0;
90            else
91              shift_chords = composition(end,1)+composition(
                  end,2);
92              shift_value = shift_chords;
93              time=time+shift_value;
94            end
95      else
96            shift_value=composition(end,1)+composition(end,2);
97            time=shift_value+offset_part1;
98            composition = [composition;shift(part1,'onset',
                time,'beat')];
99      end
100   end
101
102   if (form(i)==2)
103     if (isempty(part2))
104           [chordprogression,shift_value] = progresje(chord,
                step2,0, 0,lp2,skala);
105           chordprogression = setvalues(chordprogression,'dur
                ',lp2,'beat');
106           shift_chords = chordprogression(end,1)+
                chordprogression(end,2);
107           while (mod(shift_chords,4) ~= 0 )
108             shift_chords=shift_chords+1;
109           end
110           shift_value = shift_chords;
111           l_part2 = shift_value;
112           %przewrot
113           if (rand < 0.3)
```

```
114                 chordprogression(1,4) =  chordprogression(1,4) +
                        12;
115                 chordprogression(2,4) =  chordprogression(2,4) +
                        12;
116               end
117               if isempty(composition)
118               time = time + ceil(shift_value);
119               else
120                 shift_value=composition(end,1)+composition(end
                        ,2);
121                 while (mod(shift_value,4)~=0)
122                   shift_value = shift_value+0.0625;
123             end
124                 time=shift_value;
125           end
126           melody=makemelody(chordprogression,lp2,3,
                possible_duration_mel2, offset_mel,1);
127           melody=trim(melody);
128       bass = minigenerator('D',36,possible_location(floor(rand
            (1)*lpos+1)),possible_offset(floor(rand(1)*lof+1))
            *0,0.25,l_part2,tempo,10);
129       snare = minigenerator('D',40,possible_location_snare(floor
            (rand(1)*lpos_snare+1)),possible_offset_snare(floor(
            rand(1)*lof_snare+1)),0.25,l_part2,tempo,10);
130       clap = minigenerator('D',51,possible_location(floor(rand
            (1)*lpos+1)),possible_offset(floor(rand(1)*lof+1))
            ,0.25,l_part2,tempo,10);
131       harmonization = minigenerator('D',fill(fix(rand(1,8)*lfill
            )+1),possible_location(floor(rand(1)*lpos+1)),lp2*step2
            *2,0.25,l_part2,tempo+20,10);
132       crash = minigenerator('D',49,possible_location(floor(rand
            (1)*lpos+1)),possible_offset(floor(rand(1)*lof+1))
            *0,0.5,l_part2,tempo,10);
133           part2=[melody;bass;snare;clap;harmonization;crash;
                shift(chordprogression,'pitch',-12)];
134           composition = [composition;shift(part2,'onset',time,'
                beat')];
135         else
136                 shift_value=composition(end,1)+composition(end,2);
137           while (mod(shift_value,4)~=0)
138             shift_value = shift_value+0.0625;
139           end
140                 time=shift_value;
141                 composition = [composition;shift(part2,'onset',
                      time,'beat')];
142         end
143       end
144
```

```
145   if (form(i)==3)
146     if (isempty(part3))
147       [chordprogression,shift_value] = progresje(chord,
            step3,0, 0,lp3,skala);
148       chordprogression = setvalues(chordprogression,'dur',
            lp3,'beat');
149         %przewrot
150         if (rand < 0.3)
151           chordprogression(1,4) =  chordprogression(1,4) +
                12;
152         end
153         if isempty(composition)
154           time=0;
155         else
156           shift_value=composition(end,1)+composition(end
                ,2);
157           while (mod(shift_value,4)~=0)
158                 shift_value = shift_value+0.0625;
159           end
160           time=shift_value;
161         end
162         chordprogression(:,3)=8;
163         melody = makemelody(chordprogression,lp3,4,
              possible_duration_mel2, offset_mel,1);
164         melody = trim(melody);
165         melody2 = makemelody(chordprogression,lp3,12,
              possible_duration_mel2*2, offset_mel,1);
166         bass = minigenerator('D',36,possible_location(
              floor(rand(1)*lpos+1)),possible_offset(floor(
              rand(1)*lof+1))*0,0.25,lp3*step3,tempo,10);
167         snare = minigenerator('D',40,
              possible_location_snare(floor(rand(1)*
              lpos_snare +1)),possible_offset_snare(floor(
              rand(1)*lof_snare+1)),0.25,lp3*step3,tempo,10);
168         clap = minigenerator('D',51,possible_location(
              floor(rand(1)*lpos+1)),possible_offset(floor(
              rand(1)*lof+1)),0.25,lp3*step3,tempo,10);
169         crash = minigenerator('D',49,possible_location(
              floor(rand(1)*lpos+1))*2,possible_offset(floor(
              rand(1)*lof+1))*2,0.5,lp3*step3,tempo,10);
170         harmonization = minigenerator('D',fill(fix(rand
              (1,8)*lfill)+1),possible_location(floor(rand(1)
              *lpos+1))/2,lp3*step3*2,0.25,l_part2,tempo
              +20,10);
171         part3 = [melody;melody2;bass;snare;clap;crash;
              harmonization;shift(chordprogression,'pitch'
              ,-12)];
```

```
172             composition = [composition;shift(part3,'onset',
                    time,'beat')];
173       else
174             shift_value=composition(end,1)+composition(end,2);
175             while (mod(shift_value,4)~=0)
176               shift_value = shift_value+0.0625;
177             end
178         time=shift_value;
179         composition = [composition;shift(part3,'onset',time,'
                beat')];
180       end
181     end
182
183     if (form(i)==4)
184       if (isempty(part4))
185             [chordprogression,shift_value] = progresje(chord,
                    step4,0, 0,lp4,skala);
186             chordprogression = setvalues(chordprogression,'dur
                    ',lp4,'beat');
187             shift_chords = chordprogression(end,1)+
                    chordprogression(end,2);
188             while (mod(shift_chords,4) ~= 0 )
189               shift_chords=shift_chords+1;
190             end
191             shift_value = shift_chords;
192             l_part4 = shift_value;
193             %przewrot
194             if (rand < 0.3)
195               chordprogression(1,4) =  chordprogression(1,4) +
                      12;
196             end
197             if isempty(composition)
198               time=0;
199             else
200               shift_value=composition(end,1)+composition(end
                      ,2);
201               while (mod(shift_value,4)~=0)
202                     shift_value = shift_value+0.0625;
203               end
204               time=shift_value;
205             end
206             melody = makemelody(chordprogression,lp4,5,
                    possible_duration_mel1, offset_mel,1);
207             melody2 = makemelody(chordprogression,lp4,11,
                    possible_duration_mel2, offset_mel2,0);
208             if (~isempty(melody))
209               melody = trim(melody);
210             end
```

```
211             if (~isempty(melody2))
212               melody2 = trim(melody2);
213               melody2 = shift(melody2,'pitch',12);
214             end
215             bass = minigenerator('D',36,possible_location(
                    floor(rand(1)*lpos+1)),possible_offset(floor(
                    rand(1)*lof+1))*0,0.25,l_part4,tempo,10);
216             snare = minigenerator('D',40,
                    possible_location_snare(floor(rand(1)*
                    lpos_snare +1)),possible_offset_snare(floor(
                    rand(1)*lof_snare+1)),0.25,l_part4,tempo,10);
217             clap = minigenerator('D',54,possible_location(
                    floor(rand(1)*lpos+1)),possible_offset(floor(
                    rand(1)*lof+1))*0,0.25,l_part4,tempo,10);
218             crash = minigenerator('D',49,possible_location(
                    floor(rand(1)*lpos+1))*2,possible_offset(floor(
                    rand(1)*lof+1))*0,0.5,l_part4,tempo,10);
219             harmonization = minigenerator('D',fill(fix(rand
                    (1,8)*lfill)+1),possible_location(floor(rand(1)
                    *lpos+1))/2,l_part4*2,0.25,l_part4,tempo+20,10)
                    ;
220             part4 = [melody;melody2;bass;snare;clap;crash;
                    harmonization;shift(chordprogression,'pitch'
                    ,-12)];
221             composition = [composition;shift(part4,'onset',
                    time,'beat')];
222       else
223             shift_value=composition(end,1)+composition(end,2);
224             while (mod(shift_value,4)~=0)
225               shift_value = shift_value+0.0625;
226             end
227             time=shift_value;
228             composition = [composition;shift(part4,'onset',
                    time,'beat')];
229       end
230     end
231 end
232 drumsp =
        [1,2,3,4,9,10,11,12,13,15,16,17,18,25,26,27,28,29,30,
233 31,32,33,34,41,49,51,57,128];
234 ldrumsp = length(drumsp);
235 nm = [1,floor(0+rand(1)*10);2,floor(10+rand(1)*10);3,floor
        (20+rand(1)*10);4,floor(30+rand(1)*10);5,floor(40+rand(1)
        *10);6,floor(50+rand(1)*10);7,floor(60+rand(1)*10);8,
        floor(70+rand(1)*10);9,floor(80+rand(1)*30);10,drumsp(
        floor(rand(1)*ldrumsp+1))];
236 writemidi_java(composition,'progression_19.mid',120,tempo
        ,4,4,nm);
```

List. 5.20. Funkcja generująca progresje akordów w środowisku Matlab. Opracowanie własne.

```
function [out,shift_value] = progresje(chord, steps, offset,
    shift_value,chord_l,skala)
lo = length(offset);
out = [];
chordorg = chord;
prog=zeros(1,steps);
for k=1:steps
  chord = shift(chord, 'onset', shift_value);
  if ( k==1 )
    p=rand(1);
    if (p<=0.2)
          prog(k)=4;
          chord(chord(:,3) ~= 10,4) = chord(chord(:,3) ~=
              10,4) + skala(4);
        elseif (p>0.2 && p<=0.4)
          prog(k)=5;
          chord(chord(:,3) ~= 10,4) = chord(chord(:,3) ~=
              10,4) + skala(5);
        else
          prog(k)=1;
        end
  end
  if (k==2)
    if (prog(1)~=4)
          p=rand(1);
          if (p>0 && p<=0.2)
            chord(chord(:,3) ~= 10,4) = chord(chord(:,3) ~=
                10,4) + skala(4);
                prog(k)=4;
          elseif (p>0.2 && p<=0.4)
            chord(chord(:,3) ~= 10,4) = chord(chord(:,3) ~=
                10,4) + skala(5);
                prog(k)=5;
          elseif (p>0.4 && p<=0.6)
                chord(chord(:,3) ~= 10,4) = chord(chord(:,3)
                    ~= 10,4) + skala(6);
                prog(k)=6;
          elseif (p>0.6 && p<=0.8)
                chord(chord(:,3) ~= 10,4) = chord(chord(:,3)
                    ~= 10,4) + skala(3);
                prog(k)=2;
          else
                prog(k)=1;
          end
        end
  end
```

```
40    if (k==3)
41      if prog(k-1)==4
42        if (rand(1) < 0.5)
43              chord(chord(:,3) ~= 10,4) = chord(chord(:,3) ~=
                    10,4) + skala(5);
44            else
45                  chord(chord(:,3) ~= 10,4) = chord(chord(:,3)
                        ~= 10,4) + skala(4);
46            end
47          elseif prog(k-1)==5
48            if (rand(1) < 0.5)
49              chord(chord(:,3) ~= 10,4) = chord(chord(:,3) ~=
                    10,4) + skala(5);
50            end
51          elseif prog(k-1)==2
52            chord(chord(:,3) ~= 10,4) = chord(chord(:,3) ~=
                  10,4) + skala(4);
53          elseif prog(k-1)==6
54            if (rand(1) < 0.5)
55              chord(chord(:,3) ~= 10,4) = chord(chord(:,3) ~=
                    10,4) + skala(2);
56            else
57              chord(chord(:,3) ~= 10,4) = chord(chord(:,3) ~=
                    10,4) + skala(4);
58            end
59          else
60            if (rand(1) < 0.5)
61              prog(k)=1;
62            else
63                  chord(chord(:,3) ~= 10,4) = chord(chord(:,3)
                        ~= 10,4) + skala(4);
64                  prog(k)=4;
65            end
66          end
67          if (rand(1) < 0.25)
68            chord = [chord; shift(chord(1,:),'pitch',skala(7))
                  ];
69          end
70    end
71    if (k==4)
72      p=rand(1);
73          if prog(k-1)==4
74            if (p>0 && p<=0.2)
75              chord(chord(:,3) ~= 10,4) = chord(chord(:,3) ~=
                    10,4) + skala(4);
76            elseif (p>0.2 && p<=0.4)
77              chord(chord(:,3) ~= 10,4) = chord(chord(:,3) ~=
                    10,4) + skala(5);
```

```
78              else
79              end
80            end
81    end
82    if (k>4)
83      p=rand(1);
84      if (p>0 && p<=0.2)
85              chord(chord(:,3) ~= 10,4) = chord(chord(:,3) ~=
                    10,4) + skala(4);
86            elseif (p>0.2 && p<=0.4)
87              chord(chord(:,3) ~= 10,4) = chord(chord(:,3) ~=
                    10,4) + skala(5);
88            elseif (p>0.4 && p<=0.6)
89              chord(chord(:,3) ~= 10,4) = chord(chord(:,3) ~=
                    10,4) + skala(6);
90            else
91            end
92    end
93    shift_value = shift_value + chord_l + offset(floor(rand(1)
          * lo + 1));
94    out = [out; chord];
95    chord = chordorg;
96  end
```

List. 5.21. Funkcja generująca melodię i linię basową w środowisku Matlab. Opracowanie własne.

```
1  function [melody,bassline] = makemelody(chordprogression,ch,
       possible_duration, offset, makebass)
2  %makebass - 1 yes, 0 no
3  lpd  = length(possible_duration);
4  lo = length(offset);
5  melody=[];
6  bassline=[];
7  tottime = chordprogression(end,1)+chordprogression(end,2);
8  prevduration=0;
9  for i=0:tottime-1
10   window = onsetwindow(chordprogression,i,i+1);
11   if (isempty(window))
12     continue
13   end
14   pitch = window(:,4);
15   melfragment = [];
16   for j=window(1,1):window(1,1)+window(end,2)
17     if (~isempty(melfragment) && melfragment(end,1)+
         melfragment(end,2) >= window(end,1)+window(end,2))
18       break;
19     end
20         selectedpitch = pitch(floor(rand(1)*length(pitch)+1)
             );
21         duration = possible_duration(floor(rand(1)*lpd+1));
22         seloffset = offset(floor(rand(1)*lo+1));
23         note = createnmat(selectedpitch,0.25,127,ch(1));
24         note = setvalues(note,'dur',duration,'beat');
25         if (~isempty(melfragment) && j + seloffset <
             melfragment(end,1)+melfragment(end,2))
26           note = setvalues(note,'onset',melfragment(end,1)+
               melfragment(end,2),'beat');
27         else
28       note = setvalues(note,'onset',j+seloffset,'beat');
29         end
30         if (mod(j,4) ~= 0)
31           note(:,5) = floor(rand(1)*17+110);% dynamika
32         end
33         prevduration=duration;
34         melfragment = [melfragment; note];
35   end
36   window_duration = window(:,1) + window(:,2);
37   melfragment_duration = melfragment(:,1) + melfragment(:,2)
       ;
38   melfragment = melfragment(melfragment_duration <= max(
       window_duration),:);
39   if (~isempty(melfragment) && makebass==1)
```

```
40        bassline = [bassline; shift(melfragment([1:fix(rand(1)
             *8+1):end],:),'pitch',-24)];
41            bassline(:,3) = 9;
42    end
43    melody = [melody;melfragment];
44 end
45 if (~isempty(bassline))
46    bassline = quantize(bassline,1/4,1/16);
47 end
48 melody = [melody;bassline];
49 melody = sortrows(melody);
50 return
```

List. 5.22. Funkcja opisująca działanie mini-generatora w środowisku Matlab. Opracowanie własne.

```
1  function [out] = minigenerator(K,MM,cc,offset,res,l,tempo,ch
   )
2  if (K == 'D' || K == 'C' || K == 'M')
3          if (l>0 && res>0)
4                  dur=res*60/tempo;
5                  if (K=='D')
6                          notematrix = createnmat(zeros(1,fix(
                               l/res)),dur,127,10);
7                          notematrix = settempo(notematrix,
                               tempo);
8                  else
9                          notematrix = createnmat(zeros(1,fix(
                               l/res)),dur,127,ch);
10                         notematrix = settempo(notematrix,
                               tempo);
11                 end
12                 onsets = [1+offset:cc:size(notematrix,1)];
13                 if (length(MM)==1)
14                         notematrix(onsets,4)=MM;
15                         notematrix = notematrix(notematrix
                               (:,4)>0,:);
16                 if (K=='C')
17                                 out = [ notematrix; shift(
                                       notematrix,'pitch',4);
18                                 shift(notematrix,'pitch',5)
                                       ];
19                 return
20                         end
21                         out = notematrix;
22                         return
23                 end
24                 pitch_vec = zeros(1,length(onsets));
25                 i=1;
26                 for j=1:length(onsets)
27                         if (i==length(MM))
28                                 i=1;
29                         end
30                         pitch_vec(j) = MM(i);
31                         i=i+1;
32                 end
33                 notematrix(onsets , 4) = pitch_vec;
34                 notematrix = notematrix(notematrix(:,4)>0,:)
                       ;
35                 if (K=='C')
36                         k = kkkey(notematrix);
37                         if (k<13) %major
```

```
                                    out = [ notematrix; shift(notematrix,'pitch',4);
                                    shift(notematrix,'pitch',5)];
                            else %minor
                                    out = [ notematrix; shift(notematrix,'pitch',3);
                                    shift(notematrix,'pitch',5)];
                            end
                            return
                    end
                    out = notematrix;
            else
                    out=[];
            end
end
```

List. 5.23. Funkcja generująca przykładową macierz KMM w środowisku Matlab. Opracowanie własne.

```
function [kmm] = kmm1()
kmm = [];
p = ( [33,33,33,33,43,45,43,45,33,33,33,33,43,45,43,45] );
kmm = [kmm; p];
for pp=1:length(p)
  kmm = [kmm; circshift(p,[1, pp])];
end
p = p + 4;
for pp=1:length(p)
  kmm = [kmm; circshift(p,[1, pp])];
end
p = p + 5;
for pp=1:length(p)
  kmm = [kmm; circshift(p,[1, pp])];
end
p = p - 12;
for pp=1:length(p)
  kmm = [kmm; circshift(p,[1, pp])];
end
p = p + 12+4;
for pp=1:length(p)
  kmm = [kmm; circshift(p,[1, pp])];
end
p = p + 12+5;
for pp=1:length(p)
  kmm = [kmm; circshift(p,[1, pp])];
end
```

List. 5.24. Funkcja generująca przykładową macierz KMM dla kompozycji $s1 - 415.flac$ w środowisku Matlab. Opracowanie własne.

```
1  function [bassoctave] = bassOctaveGen11()
2  bassoctave = [];
3  p = ( [50,48,50,53,55,43,41,43,36,38] );
4  base = p;
5  bassoctave = [bassoctave; p];
6  for pp=1:length(p)
7          bassoctave = [bassoctave;
8          circshift(p,[pp, pp+1])];
9  end
10 p = p + 4;
11 bassoctave = [bassoctave; p];
12 for pp=1:length(p)
13         bassoctave = [bassoctave; circshift(p,[pp, pp+1])];
14 end
15 p = base;
16 p = p + 7;
17 bassoctave = [bassoctave; p];
18 for pp=1:length(p)
19         bassoctave = [bassoctave; circshift(p,[pp, pp+1])];
20 end
```

Dodatek B. Zawartość dołączonej płyty CD

Nazwa pliku	Opis	Strona
$w_411_1_A.flac$	przykładowy wzorzec klasy A	100
$w_411_2_A.flac$	przykładowy wzorzec klasy A	100
$k_raw_411.flac$	kompozycja wygenerowana na podstawie ustawień ze strony 97	101
$dur_0-125.flac$	kompozycja wygenerowana z dźwięków o długości 0.125 s.	103
$dur_0-25.flac$	kompozycja wygenerowana z dźwięków o długości 0.25 s.	103
$dur_0-5.flac$	kompozycja wygenerowana z dźwięków o długości 0.5 s.	103
$dur_1.flac$	kompozycja wygenerowana z dźwięków o długości 1 s.	103
$dur-mix.flac$	kompozycja wygenerowana na podstawie danych z tab. 4.4	103
$pattern_l-4.flac$	kompozycja wygenerowana ze wzorców o długości 4 dźwięków	105
$pattern_l-8.flac$	kompozycja wygenerowana ze wzorców o długości 8 dźwięków	105
$pattern_l-16.flac$	kompozycja wygenerowana ze wzorców o długości 16 dźwięków	105
$pattern_l-32.flac$	kompozycja wygenerowana ze wzorców o długości 32 dźwięków	105
$pattern_n16-u8.flac$	kompozycja wygenerowana na podstawie 8 wzorców wybranych ze zbioru 16	107
$pattern_n16-u16.flac$	kompozycja wygenerowana na podstawie 16 wzorców wybranych ze zbioru 16	107
$pattern_n16-u32.flac$	kompozycja wygenerowana na podstawie 32 wzorców wybranych ze zbioru 16	107
$pattern_n32-u8.flac$	kompozycja wygenerowana na podstawie 8 wzorców wybranych ze zbioru 32	107
$pattern_n32-u32.flac$	kompozycja wygenerowana na podstawie 32 wzorców wybranych ze zbioru 32	107

$pattern_n32-u64.flac$	kompozycja wygenerowana na podstawie 64 wzorców wybranych ze zbioru 32	107
$pattern_rt-1.flac$	kompozycja wygenerowana na podstawie tablicy rt o strukturze [1]	108
$pattern_rt-2.flac$	kompozycja wygenerowana na podstawie tablicy rt o strukturze [2]	108
$pattern_rt-1-2.flac$	kompozycja wygenerowana na podstawie tablicy rt o strukturze $[1,2]$	108
$pattern_rt-1-4.flac$	kompozycja wygenerowana na podstawie tablicy rt o strukturze $[1,4]$	108
$pattern_rt-2-4.flac$	kompozycja wygenerowana na podstawie tablicy rt o strukturze $[2,4]$	108
$pattern_rt-1-2-4.flac$	kompozycja wygenerowana na podstawie tablicy rt o strukturze $[1,2,4]$	108
$s1-415.flac$	przykładowa kompozycja wygenerowana na podstawie schematu z rys. 4.8	112
$k_raw_421.flac$	kompozycja wygenerowana na podstawie ustawień ze strony 113	117
$f_1111.flac$	kompozycja wygenerowana na podstawie wektora formy $[1,1,1,1]$	116
$f_4444.flac$	kompozycja wygenerowana na podstawie wektora formy $[4,4,4,4]$	116
$f_1234.flac$	kompozycja wygenerowana na podstawie wektora formy $[1,2,3,4]$	116
$f_12344321.flac$	kompozycja wygenerowana na podstawie wektora formy $[1,2,3,4,4,3,2,1]$	116
$s_mol_nat.flac$	kompozycja wygenerowana na podstawie wektora skali $[0,2,3,5,7,8,10,12]$	120
$s_mol_har.flac$	kompozycja wygenerowana na podstawie wektora skali $[0,2,3,5,7,8,11,12]$	120
$s_mol_dor.flac$	kompozycja wygenerowana na podstawie wektora skali $[0,2,3,5,7,9,11,12]$	120
$s_dur_nat.flac$	kompozycja wygenerowana na podstawie wektora skali $[0,2,4,5,7,9,11,12]$	120
$s_dur_har.flac$	kompozycja wygenerowana na podstawie wektora skali $[0,2,4,5,7,8,11,12]$	120
$s_dur_soft.flac$	kompozycja wygenerowana na podstawie wektora skali $[0,2,4,5,7,8,10,12]$	120

$pl_0.flac$	kompozycja wygenerowana na podstawie par. $possible_location = 0$	124
$pl_4.flac$	kompozycja wygenerowana na podstawie par. $possible_location = 4$	124
$pl_0248.flac$	kompozycja wygenerowana na podstawie par. $possible_location = [0, 2, 4, 8]$	124
$po_4_om_005.flac$	kompozycja wygenerowana na podstawie parametru $possible_offset = 4$ oraz $offset_mel = [0, 0.5]$	128
$po_3_om_005.flac$	kompozycja wygenerowana na podstawie parametru $possible_offset = 3$ oraz $offset_mel = [0, 0.5]$	128
$po_0_om_2.flac$	kompozycja wygenerowana na podstawie parametru $possible_offset = 0$ oraz $offset_mel = 2$	128
$po_024_om_24.flac$	kompozycja wygenerowana na podstawie parametru $possible_offset = [0, 2, 4]$ oraz $offset_mel = [2, 4]$	128
$pdm1_00125_pdm2_05.flac$	kompozycja wygenerowana na podstawie parametru $possible_duration_mel1 = 0.0125$ oraz $possible_duration_mel2 = 0.5$	129
$pdm1_025_pdm2_00125.flac$	kompozycja wygenerowana na podstawie parametru $possible_duration_mel1 = 0.25$ oraz $possible_duration_mel2 = 0.0125$	129
$pdm1_2_pdm2_05.flac$	kompozycja wygenerowana na podstawie parametru $possible_duration_mel1 = 2$ oraz $possible_duration_mel2 = 0.5$	129
$pdm1_02524_pdm2_05.flac$	kompozycja wygenerowana na podstawie parametru $possible_duration_mel1 = [0.25, 2, 4]$ oraz $possible_duration_mel2 = 0.5$	129
$lp_2222_s_4888.flac$	kompozycja wygenerowana na podstawie parametrów: $lp1 = 4$, $lp2 = 4$, $lp3 = 4$, $lp4 = 4$, $step1 = 4$, $step2 = 8$, $step3 = 8$ oraz $step4 = 8$	132

$lp_8888_s_4888.flac$	kompozycja wygenerowana na podstawie parametrów: $lp1 = 8$, $lp2 = 8$, $lp3 = 8$, $lp4 = 8$, $step1 = 4$, $step2 = 8$, $step3 = 8$ oraz $step4 = 8$	132
$lp_4444_s_2444.flac$	kompozycja wygenerowana na podstawie parametrów: $lp1 = 4$, $lp2 = 4$, $lp3 = 4$, $lp4 = 4$, $step1 = 2$, $step2 = 4$, $step3 = 4$ oraz $step4 = 4$	132
$lp_4444_s_8121212.flac$	kompozycja wygenerowana na podstawie parametrów: $lp1 = 4$, $lp2 = 4$, $lp3 = 4$, $lp4 = 4$, $step1 = 8$, $step2 = 12$, $step3 = 12$ oraz $step4 = 12$	132
kompozycje.zip	przykładowe autorskie kompozycje muzyki generatywnej	140
$TEWIProject.zip$	kod źródłowy aplikacji "Wirtualne Laboratorium Muzykologa"	140

Składam serdeczne podziękowania mojemu Promotorowi
Panu Profesorowi Antoniemu Wilińskiemu
za okazaną życzliwość, wsparcie
oraz wskazówki udzielone mi podczas pisania pracy.

Dziękuję Recenzentom
za trud włożony w ocenę niniejszej dysertacji
oraz za konstruktywną krytykę moich osiągnięć.

Gorąco dziękuję również mojej Rodzinie
za wiarę we mnie,
za podnoszenie mnie na duchu w chwilach słabości
oraz za wsparcie podczas powstawania tej pracy.