Česká zemědělská univerzita v Praze

Provozně ekonomická fakulta

Katedra informačního inženýrství

Bakalářská práce

Procedurální generování hudby

Marek Juřica

ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE

Provozně ekonomická fakulta

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Marek Juřica

Informatika

Název práce

Procedurální generování hudby

Název anglicky

Procedural music generation

Cíle práce

Aplikace je zaměřena na problematiku algoritmizace skládání hudby. Hlavním cílem je vytvořit aplikaci, která dokáže na základě zadaných parametrů vygenerovat skladbu a uložit jí do souboru midi.

Metodika

Práce sestává ze dvou částí, teoretické a praktické.

Teoretická východiska pro zpracování praktické části, představující první část práce, budou popsána na základě syntézy poznatků získaných studiem odborné literatury. Součástí je průzkum možností algoritmizace skládání hudby.

Praktická část práce spočívá v analýze, návrhu a implementaci aplikace aplikace pro automatizované generování hudby s výstupem do formátu midi. Během analýzy a návrhu bude využito standardních metod softwarového inženýrství, implementace bude provedena s využitím jazyka Java a souvisejících technologií.

Aplikace bude dále otestována, budou shrnuty poznatky získané během jejího vývoje a navrženy případné další možnosti jejího budoucího rozvoje.

Doporučený rozsah práce

35-40 stran

Klíčová slova

hudba, java, procedurální generování, midi, javafx

Doporučené zdroje informací

JELÍNEK, Stanislav, Skladba – úvod do hudební kompozice. 1. vyd. Kladno : Vladimír Beneš, 2009. 52 s. ISBN: 978-0-706512-77-8

PECINOVSKÝ, Rudolf. Java 8: Úvod do objektové architektury pro mírně pokročilé. 1. vyd. Praha : Grada Publishing a.s., 2014. 656 s. ISBN: 978-80-247-4638-8

Procedural Content Generation Wiki [online]. 11th of May 2008, last revision 18th of October 2013 . <http://www.pcg.wikidot.com/>

SHAKER Noor, TOGELIUS Julian, NELSON Mark J. Procedural Content Generation in Games: A Textbook and an Overview of Current Research. Springer, 2016. ISBN 978-3-319-42714-0

TOUSSAINT, G. T. The Euclidean algorithm generates traditional musical rhythms, Proceedings of BRIDGES: Mathematical Connections in Art, Music, and Science, Banff, Alberta, Canada, July 31 to August 3, 2005, pp. 47–56

Předběžný termín obhajoby

2017/18 LS – PEF

Vedoucí práce

Ing. Jiří Brožek, Ph.D.

Garantující pracoviště

Katedra informačního inženýrství

Elektronicky schváleno dne 7. 3. 2018

Ing. Martin Pelikán, Ph.D.

Vedoucí katedry

Elektronicky schváleno dne 7. 3. 2018

Ing. Martin Pelikán, Ph.D.

Děkan

V Praze dne 12. 03. 2018

Čestné prohlášení

Prohlašuji, že svou bakalářskou práci "Procedurální generování hudby" jsem vypracoval samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou citovány v práci a uvedeny v seznamu použitých zdrojů na konci práce. Jako autor uvedené bakalářské práce dále prohlašuji, že jsem v souvislosti s jejím vytvořením neporušil autorská práva třetích osob.

V Praze dne 14.3.2018 _____

Poděkování

Rád bych touto cestou poděkoval panu Ing. Jiřímu Brožkovi, Ph.D. za cenné rady, ochotu a trpělivost při vedení mé práce.

Procedurální generování hudby

Abstrakt

Práce se zabývá problematikou algoritmické kompozice hudby. Po nastudování odborné literatury autor nejprve shrnuje teoretická východiska z oblasti hudební teorie a technologií používaných při vývoji softwaru. Na jejich základě se pak pokouší navrhnout techniky procedurálního generování, které lze použít ke skládání hudby počítačem.

Praktická část této práce spočívá v návrhu a implementaci aplikace v jazyce Java 8 za použití souvisejících technologií. Jejím výstupem je soubor MIDI obsahující hudební skladbu vygenerovanou na základě uživatelem zadaných parametrů pomocí již zmíněných technik procedurálního generování.

Cílem práce je prozkoumat, do jaké míry lze skladatelskou činnost popsat pomocí algoritmů, hledání hranice mezi uměleckou a automatizovanou prací a návrh možností budoucího vývoje v tomto směru.

Klíčová slova: hudba, java, procedurální generování, midi, javafx

Procedural Generation of Music

Abstract

This thesis deals with the issue of algorithmic music composition. After studying scholarly texts, the author sums theoretical foundations related to music theory and software development. Based on this theoretical background, the author tries to suggest techniques of procedural generation, which can be used for computer driven music composition.

Practical part of this thesis concerns design and implementation of an application written in Java language using related technologies. Its output is a MIDI file that contains a music track generated by using previously mentioned procedural generation techniques.

The aim of this thesis is to observe to which extent the process of composing music can be described using algorithms. Related task is to search for a boundary between artistic and automated work and to propose some possibilities of the future development in this issue.

Keywords: music, java, procedural generation, midi, javafx

Obsah

Seznam obrázků

Seznam tabulek

Seznam zdrojových kódů

1 Úvod

Většina běžných lidských činností se dá popsat algoritmem. Ať už jde o řízení auta nebo třeba vaření či jakoukoli práci, často je to činnost plně automatická a determinovaná, a i když u ní člověk přemýšlí, v podstatě jen provádí výpočty stanovené jednoznačným postupem.

Důvodem pro to, že ji stále dělají lidé a ne stroje, je často pouze absence nutné technologie nebo její vysoká cena pro konkrétní použití. Stroje navíc dokáží algoritmizované procesy vykonávat efektivněji, rychleji a s menším počtem chyb. Stále naléhavější je otázka, do jaké míry lze strojem člověka nahradit.

Existují odvětví, ve kterých to rozhodně nebude jednoduché. Jak algoritmizovat uměleckou činnost, jako je tvůrčí psaní či skládání hudby? Pojmy jako intuice, kreativita nebo emoce jsou pro umění naprosto zásadní, ale pro jejich nejednoznačnost je práce s nimi v procesu algoritmizace složitá. I samotná definice umění je příliš nejistá na to, aby se dal jednoduše vymyslet rozhodovací proces pro jeho tvorbu.

Přesto se už odpradávna odehrávají pokusy vzít člověku jeho výsostnou pozici v umělecké produkci. Na poli hudby existuje mnoho příkladů snahy o mechanizaci, ať už jde o zvonkohry ze starověké Asie nebo „generativní hudbu“ Bryana Eno. Hudební teorie byla po celou dobu svého vývoje doplňována pravidly, která umožňují skladatelské umění popsat a vysvětlit. Část z nich by se jistě dala využít pro návrh algoritmů, jejichž výstupem by byla hudba. I někteří tradiční skladatelé, jako byl Bach nebo Mozart, se někdy snažili postupovat způsobem, který bychom dnes nazvali algoritmickou kompozicí.

Snahou této práce je shrnout znalosti z oborů hudební teorie, algoritmizace a programování a na jejich základě navrhnout postupy, které mohou sloužit k procedurálnímu generování hudby. Praktickým výstupem je aplikace, která dokáže po zadání parametrů, jako tempo a tónina, vygenerovat hudební skladbu ve formátu MIDI. Tento formát má dle názoru autora mezi běžnými způsoby ukládání hudby nejblíže k lidsky čitelnému notovému zápisu. Aplikace je napsána v jazyce Java 8, její grafické uživatelské rozhraní je vytvořeno pomocí platformy JavaFX a sestavena je s využitím nástroje Apache Maven. Zdrojové kódy jsou verzovány systémem Git a jednotlivé verze nahrávány na server GitHub.

2 Cíl práce a metodika

2.1 Cíl práce

Aplikace je zaměřena na problematiku algoritmizace skládání hudby. Hlavním cílem je vytvořit aplikaci, která dokáže na základě zadaných parametrů vygenerovat skladbu a uložit jí do souboru MIDI.

2.2 Metodika

Práce sestává ze dvou částí, teoretické a praktické.

Teoretická východiska pro zpracování praktické části, představující první část práce, budou popsána na základě syntézy poznatků získaných studiem odborné literatury. Součástí je průzkum možností algoritmizace skládání hudby.

Praktická část práce spočívá v analýze, návrhu a implementaci aplikace pro automatizované generování hudby s výstupem do formátu MIDI. Během analýzy a návrhu bude využito standardních metod softwarového inženýrství, implementace bude provedena s využitím jazyka Java a souvisejících technologií.

Aplikace bude dále otestována, budou shrnuty poznatky získané během jejího vývoje a navrženy případné další možnosti jejího budoucího rozvoje.

3 Teoretická východiska

V této části práce jsou popsány základy hudební teorie nutné pro implementaci aplikace, formát MIDI, ve kterém se ukládá vygenerovaná hudební skladba a programovací jazyk Java 8, ve kterém je aplikace napsána. Dalšími použitými nástroji při vývoji aplikace, kterými se v této části autor také zabývá, jsou systém správy verzí Git a nástroj pro řízení buildů aplikací Apache Maven. Zmíněn je také testovací framework JUnit a program pro snadnou tvorbu grafických uživatelských rozhraní JavaFX aplikací Scene Builder. V závěru tohoto oddílu je věnován určitý prostor vývojovému prostředí, ve kterém probíhala tvorba aplikace. Pro tento účel bylo autorem vybráno prostředí IntelliJ IDEA.

3.1 Hudební teorie

Pro návrh algoritmů, které by se daly použít ke generování hudby, je nutné znát základy hudební teorie, jimiž se zabývá tato kapitola. Hudební teorie je velice rozsáhlý obor a pro tuto práci je pouze nástrojem k dosažení cíle a nikoli cílem samotným. Proto je nutné se uchýlit k mnohým zjednodušením. V následujících řádcích se nachází pouze neúplný popis elementární hudební teorie platné mezi západními kulturami, o kterou se budou posléze opírat následující kapitoly.

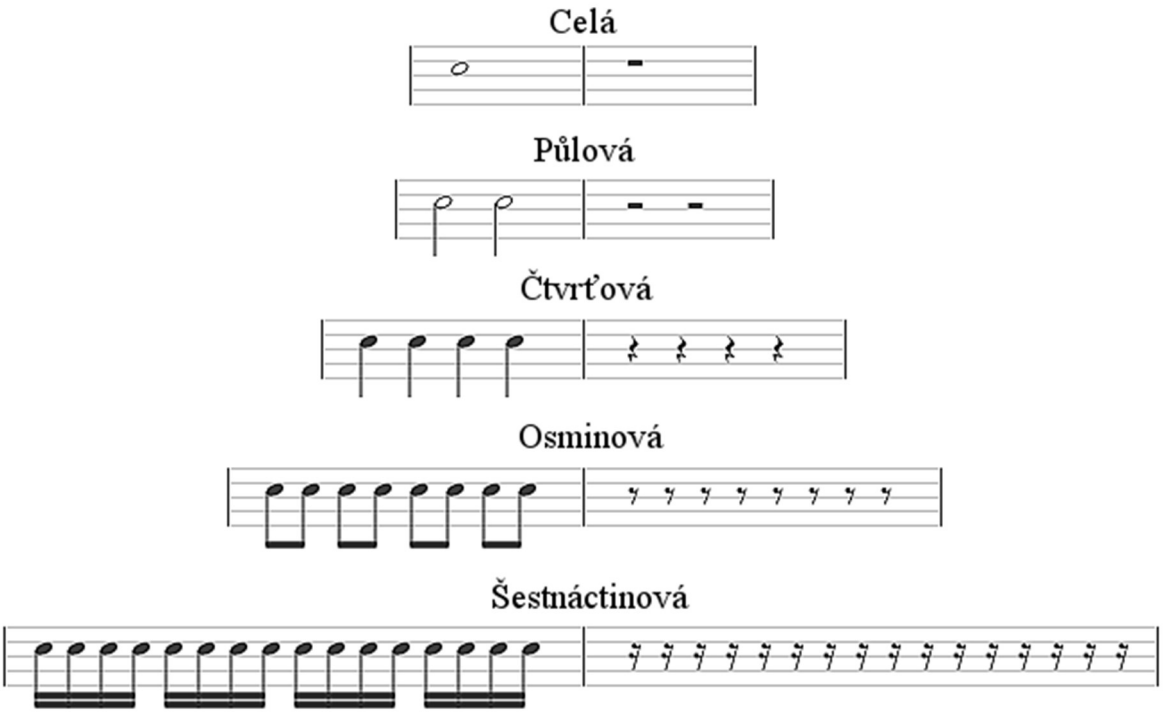
3.1.1 Tón a nota

Tón je základním prvkem hudby. Je definován jako zvuk se stálým, periodickým frekvenčním průběhem. V hudbě se rozlišují čtyři vlastnosti tónu: délka, výška, síla a barva. Délka popisuje dobu, po kterou tón zní, výška je určena jeho frekvencí a síla amplitudou. Barva je dána jeho spektrálním složením, popisuje jeho zvukové kvality a je určena například nástrojem, který daný tón hraje [1 str. 8].

Nota je grafickou reprezentací tónu, její tvar popisuje délku trvání tónu a její umístění na notové osnově udává jeho výšku. Notová osnova je pět rovnoběžných čar, na které se zakreslují noty [2 str. 7]. Vizuální stránka notového zápisu však není předmětem této práce, není tedy nutné se zde notovou osnovou zabývat podrobněji.

Běžně používané typy not podle délky trvání, kterými se tato práce dále zabývá, jsou celá, půlová, čtvrťová a šestnáctinová. Každá z nich má dobu trvání poloviční než předchozí (Obrázek 1). Noty, které mají určitou délku, ale místo tónu reprezentují tichá místa, se nazývají pomlky [1 str. 31].

Obrázek 1 – Délky not a pomlk



Zdroj: autor

3.1.2 **Rytmus a tempo**

Rytmus je pravidelné střídání časových úseků, kterým se říká doby. Doba může být těžká nebo lehká, lze si to představit například jako slabší a silnější ťuknutí metronomu [1]. Základní rytmickou jednotkou v hudbě je takt. Jeho velikost je popsána zlomkem, přičemž jmenovatel určuje délku a čitatel počet not, které definují jeho doby [2 str. 11]. Tato práce se bude zabývat pouze taktem čtyřčtvrťovým (4/4), kde první a třetí doba jsou těžké a druhá a čtvrtá lehké. Tempo popisuje rychlost skladby, a i když existuje více způsobů jeho zápisu, zde bude určováno počtem úderů (dob) za minutu. Pro tuto cestu se používá zkratka bpm (z angličtiny – beats per minute).

3.1.3 **Stupnice, tónina a hudební intervaly**

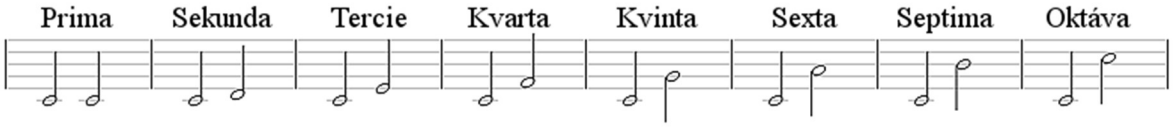
Nejmenší vzdáleností mezi dvěma výškami tónu v hudebním systému západního světa je takzvaný půltón, vzdálenost rovna dvěma půltónům se nazývá celý tón [1 str. 20].

Běžně se v souvislosti se vzdáleností mezi dvěma výškami tónů používá pojem hudební interval. Pokud vynásobíme frekvenci tónu dvěma, výška výsledného tónu bude o 12 půltónů větší než výška tónu původního, což odpovídá intervalu, který se nazývá čistá oktáva [2 str. 27].

Tóny, mezi kterými je interval čisté oktávy, jsou označovány stejným způsobem a v mnohých případech se s nimi pracuje, jako by se jednalo o tóny totožné. V rámci oktávy lze podle výšky vybrat řadu tónů, které tvoří takzvanou stupnici. Se stupnicí souvisí pojem tónina, což je soustava tónů z dané stupnice a vztahů mezi nimi. V této práci budou dále rozebírány pouze dvě stupnice diatonické, a to dur a moll. Diatonická stupnice je složena ze sedmi tónů, přičemž výšky sousedních tónů se liší o dva, případně jeden půltón [1].

Nejznámější stupnice se nazývá C dur a obsahuje 7 tónů, které tvoří hudební abecedu, jsou to tóny C, D, E, F, G, A, H. Někdy se na konci stupnice uvádí i tón o oktávu vyšší než první uvedený, v tomto případě tedy C. Její název je dán prvním tónem a přízvisko dur znamená, že je v ní použit interval velké tercie, který odpovídá čtyřem půltónům. Intervaly mezi tóny jsou následující (Obrázek 2): Nulový rozdíl mezi dvěma výškami tónů (C–C) se nazývá čistá prima. Mezi tóny C a D je interval velké sekundy, který odpovídá dvěma půltónům a nazývá se velká sekunda. Mezi tóny C a E je již zmíněný interval velké tercie. Tóny C a F jsou od sebe vzdáleny pět půltónů, což je interval čisté kvarty. C a G dělí sedm půltónů neboli interval čisté kvinty. Rozdíl devíti půltónů je mezi tóny C a A – velká sexta a rozdíl jedenácti půltónů neboli velká septima, je mezi tóny C a H [3].

Obrázek 2 – Základní hudební intervaly



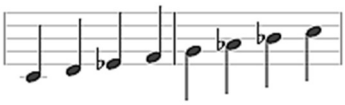
Zdroj: autor

Z uvedeného popisu stupnice C dur je vidět, že mezi každými sousedními dvěma tóny je rozdíl dva půltóny s výjimkou dvojic E–F a H–C, kde je rozdíl pouze jeden půltón. Existují i jiné intervaly než čisté a velké, například intervaly zvětšené, zmenšené a malé. Zvětšený interval vznikne z čistého intervalu, když je k němu jeden půltón přidán a zmenšený, pokud je jeden tón odebrán. Malý interval vzniká z velkého intervalu sníženého o půltón [3]. Zvětšené a zmenšené intervaly lze vytvořit i z velkých a malých, ty ale společně s některými dalšími nezmíněnými přesahují rámec této práce.

Pokud je tón z hudební abecedy zvýšen o půltón, je navíc označen symbolem # a k jeho slovnímu označení je přidána slabika „is“ (C# – cis, F# – fis atd.). Stejným způsobem může být tón snížen, v tomto případě se používá značka ♭ a slabika „es“ (C♭ – ces, E♭ – es, A♭ – as). Značkám # a ♭ se v hudební teorii říká posuvky [1 str. 21]. Z výše uvedených intervalů mezi jednotlivými tóny hudební abecedy lze vyčíst, že tóny C# a D♭ nebo třeba E# a F znějí stejně. Výpis všech tónů v rámci jedné oktávy by mohl vypadat následovně: C, C#, D, D#, E, F, F#, G, G#, A, A#, H.

Kromě durové stupnice je v této práci nutno zmínit ještě stupnici molovou, pro kterou je typická malá tercie. Intervaly v mollové stupnici jsou tyto: čistá prima, velká sekunda, malá tercie, čistá kvarta, čistá kvinta, malá sexta a malá septima [3]. Stupnice C moll tedy vypadá následovně: C, D, E♭, F, G, A♭, H♭ (Obrázek 3). Dá se říci, že durová stupnice působí na člověka vesele, zatímco mollová smutně. Kromě výše uvedené mollové stupnice, které se říká aiolská, existují ještě dvě její obměny – mollová harmonická a mollová melodická stupnice, těmi se však tato práce více zabývat nebude.

Obrázek 3 – Stupnice C moll aiolská



Zdroj: autor

3.1.4 **Akordy**

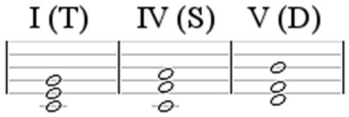
Předchozí část textu se stručně zabývala hudebními intervaly. Ty mohou být buď melodické, pokud se týkají dvou tónů zahraných po sobě, nebo harmonické, pokud jsou tyto dva tóny zahrány současně. Souzvuk dvou tónů může znít příjemně neboli libozvučně či konsonantně, nebo nepříjemně, nelibozvučně, disonantně. Disonance se v hudbě běžně používá pro zvýšení napětí, zatímco konsonance pro jeho uvolnění. Názor na to, které hudební intervaly jsou konsonantní a které disonantní, se v průběhu dějin vyvíjel a měnil, dnes jsou však za konsonantní považovány všechny čisté intervaly (prima, oktáva, kvinta a kvarta) spolu s velkou a malou tercií a velkou a malou sextou [1].

Souzvuk alespoň tří tónů se nazývá akord. V této práci se bude dále pracovat pouze s kvintakordy a septakordy. Kvintakordy jsou akordy složené ze tří tónů, kde intervaly mezi dvěma sousedními tóny odpovídají velké či malé tercii. Interval mezi nejvyšším a nejnižším tónem je tedy čistá, zvětšená, nebo zmenšená kvinta. Septakord vznikne tak, že se nad nejvyšší tón kvintakordu přidá ještě jeden v intervalu velké nebo malé tercie [2].

3.1.5 **Základní harmonické funkce**

Na každém stupni diatonické stupnice lze pomocí jejích tónů postavit kvintakord. Podle toho, na kterém stupni byl postaven, se liší jeho harmonické funkce. V této práci se budeme zabývat pouze třemi harmonickými funkcemi, a to tónikou, subdominantou a dominantou (Obrázek 4), které odpovídají prvnímu, čtvrtému a pátému stupni [1 str. 175]. Protože kvintakordy postavené na těchto stupních obsahují vždy všechny tóny diatonické stupnice, dá se s jejich pomocí doprovázet většina jednoduchých skladeb.

Obrázek 4 – Základní harmonické funkce



Zdroj: autor

3.1.6 **Kadence**

Kadence je v hudbě jakýsi pohyb využívající harmonické funkce směřující k nějakému závěru [3]. Celá autentická kadence je směrem od dominanty k tónice, poloviční autentická kadence od tóniky k dominantě. Dále se používá pojem plagální kadence v souvislosti s tónikou a subdominantou, přičemž tato kadence může být také celá nebo poloviční, podle toho, zda se jedná o pohyb směrem od subdominanty k tónice nebo naopak. Jakýkoli pohyb od dominanty jinam než k tónice, se nazývá klamná kadence (nebo také klamný závěr) [4].

3.2 **Poznámky k hudební kompozici**

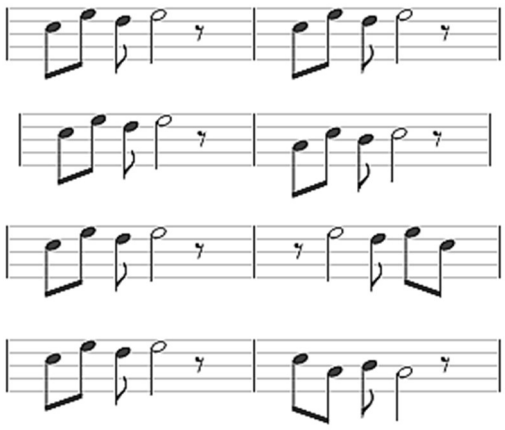
Kompozice hudby je proces, jehož osvojení je náročné a trvá dlouho. Lze jej jen těžko jednoznačně popsat a pokud se přeci v jeho souvislosti zmiňují nějaká jasně daná pravidla, pak pouze s dodatkem, že dobře odvedená práce skladatele spočívá v jejich porušování. Není se čemu divit, jedná se o umění, a to má z principu k exaktnosti daleko.

Přesto se autor pokusil z množství dostupných materiálů vybrat několik základních pojmů a pokynů, které by mohly sloužit jako základ pro algoritmizaci procesu skládání. Jelikož si poučky z různých materiálů často protiřečily, jejich výběr byl určen především tím, aby jako celek dávaly co největší smysl při doslovné interpretaci. Jejich cena pro skutečného skladatele není relevantní.

3.2.1 **Motiv a Perioda**

Motiv je jakýsi atomický díl skladby. Představuje nejmenší strukturální část, která nese nějakou hudební myšlenku [5]. Má délku obvykle jeden až dva takty a v průběhu skladby se se s ním různým způsobem pracuje [2]. Existuje více možností aplikace motivů uvnitř skladby, které se dají navzájem kombinovat (Obrázek 5). Nejjednodušší z nich je doslovné opakování, případně opakování s posunutím v rámci tóniny. Motiv lze prodloužit o několik tónů, zkrátit, přehrát pozpátku, nebo také inverzně, kdy se například místo sekvence tónů C, E, D, E použije sekvence C, A, H, A. Rozdíly mezi výškami dvou sousedních tónů jsou v prvním případě +2, -1, +1 a ve druhém -2, +1, -1 [6].

Obrázek 5 – Příklady práce s motivem



Zdroj: autor

Perioda je větší část skladby než motiv, která se skládá ze dvou dílů, z předvětí a závětí. Tyto dva díly mají každý délku kolem čtyř taktů a existuje u nich určitá motivická souvislost [2].

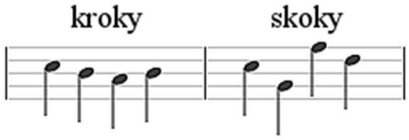
3.2.2 **Harmonická struktura**

Doprovod k melodii je definován její harmonickou strukturou. Je to v podstatě řada harmonických funkcí, která postupuje spolu s melodií. Jak bylo zmíněno výše, pohyb od jedné harmonické funkce ke druhé se nazývá kadence. Důležité je, aby skladba nestála dlouho na jednom místě, proto by se měl k melodii přiřazený harmonický stupeň měnit alespoň jednou za takt. První akord doprovázející nějakou melodickou sekvenci by měl být postaven na tónice nebo subdominantě. Řada by měla končit nejlépe celou autentickou kadencí, tedy dominantou a posléze tónikou [7].

3.2.3 **Melodie**

Mezi jednotlivými tóny, které tvoří melodii, mohou být takzvané kroky nebo skoky. Krok je mezi tóny, které spolu v dané tónině sousedí, v ostatních případech se jedná o skok (Obrázek 6). Ideálně by měla melodie obsahovat oboje.

Obrázek 6 – Kroky a skoky



Zdroj: autor

Melodie by měla odpovídat harmonické struktuře. V podstatě to znamená, že pokud si harmonickou strukturu představíme jako sled akordů, vybrané tóny v melodii by měly být součástí těchto akordů. Jedná se často o tóny, které připadnou na těžkou nebo lehkou dobu, dále jsou to tóny, ke kterým se melodie dopracuje pomocí skoku a také tóny které se často opakují [8].

3.3 **MIDI**

MIDI, celým názvem Musical Instruments Digital Interface, je rodina průmyslových standardů vyvinutá v první polovině osmdesátých let kvůli snaze o sjednocení rozhraní digitálních hudebních nástrojů. Standardy definují mimo jiné druh použitých konektorů, komunikační a datový protokol. Standard General MIDI pak specifikuje indexy pro různé zvukové sady reprezentující různé hudební nástroje [9]. Pro tuto práci je však zásadní definice souborového formátu Standard MIDI File (SMF), který říká, jakým způsobem se informace o událostech specifikovaných formátem MIDI ukládají do souborů.

3.3.1 **Základní informace o formátu MIDI**

Standard MIDI File je formát pro ukládání hudebních souborů, který na rozdíl od jiných neuchovává informace o zvuku jako o mechanickém vlnění. Namísto toho pracuje se záznamy jednotlivých tónů obsahující časové údaje, výšku tónu a jeho intenzitu. Dále uchovává informace, jako je tempo a předznamenání, či parametry použitých hudebních nástrojů [9].

Soubor ve formátu SMF (v běžných případech má koncovku .mid) je poskládán ze samostatných bloků (v originále chunk), z nichž jeden reprezentuje hlavičku (header) souboru a ostatní jednotlivé stopy (track). První 4 bajty obsahují identifikátor bloku, další 4

potom určují velikost zbylé části bloku. Formát MIDI je takzvaně big endian, což znamená, že se vícebajtové informace ukládají v pořadí, kde nejvýznamnější bajt je na nejnižší adrese [10].

3.3.2 Typy MIDI souborů

Existují celkem 3 typy souborů ve formátu SMF. První, a zároveň nejjednodušší z nich, je označován číslem 0 a obsahuje pouze jednu stopu, do které jsou shrnuta data ze všech kanálů. U každé události se poté předpokládá, že bude identifikován kanál, pro který je určena. Druhý typ s číslem 1 je v současnosti nejpoužívanější. Obsahuje více stop přehrávaných současně, typicky jednu pro každý kanál. U nečisté varianty jsou možné i vícekanálové stopy. Poslední typ označovaný číslem 2 se prakticky nepoužívá. Obsahuje několik vícekanálových stop reprezentujících vzory, které jsou na sobě nezávislé, mohou mít různá tempa a nemusí být nutně přehrávány současně [11].

3.3.3 Hlavička MIDI souboru

Hlavičkový blok (Tabulka 1) se v souboru MIDI nachází právě jednou, má identifikátor odpovídající ASCII znakům „MThd“ a jeho velikost je vždy 6 bajtů (délka identifikátoru a informace o velikosti se nepočítá). Nejprve je zde ve 2 bajtech uložena informace o formátu (0, 1, nebo 2), další dva bajty určují počet stop, které budou definovány (minimálně jedna, u typu 0 právě jedna). Na posledních dvou bajtech se nachází informace o použitých časových jednotkách (tzv. „delta time“). Pokud se nejvýznamnější bit v této části rovná nule, následujících 15 bitů reprezentuje, kolik časových atomických jednotek obsahuje čtvrťová nota. Lze se setkat se zkratkou PPQ neboli parts per quarter note. Pokud je nejvýznamnější bit roven jedné, je pro rozlišení použit formát SMPTE (podle Society of Motion Picture and Television Engineers), kde následujících 7 bitů je záporné číslo odpovídající počtu snímků za sekundu (-24, -25, -29, nebo -30) a nejméně významných 8 bitů určuje počet atomických časových jednotek v rámci jednoho SMPTE snímku [12].

Tabulka 1 – Hlavička MIDI souboru

Podsekce	Počet bajtů	Data (hexadecimálně)	Popis
Identifikátor	4	4D 54 68 64	MThd
Informace o délce	4	00 00 00 06	6 bajtů
Typ MIDI souboru	2	00 01	Typ 1 formátu SMF
Počet stop	2	00 02	2 stopy
PPQ	2	01 E0	480 PPQ

Zdroj: autor

3.3.4 **MIDI stopa**

Po hlavičkovém bloku následuje jeden či více bloků MIDI stop. Blok MIDI stopy je identifikován čtveřicí znaků MTrk a může být různě velký. V MIDI stopách jsou již uložena samotná hudební (ale i nehudební) data v podobě takzvaných událostí (v překladu events), přičemž poslední z nich musí být patička MIDI stopy reprezentovaná znaky 0xFF 0x2F 0x00 [10]. Následující tabulka reprezentuje jednoduchou MIDI stopu (Tabulka 2).

Tabulka 2 – Příklad jednoduché MIDI stopy

Data (hexadecimálně)	Popis
4D 54 72 6B	MTrk
00 00 00 XX	Informace o délce
00 C0 01	Nastavení MIDI nástroje 1 (klavír)
00 90 48 64	Zapnutí noty C
87 40 80 48 00	Vypnutí noty C
01 FF 2F 00	Patička stopy

Zdroj: autor

3.3.5 **Události**

Existují 3 druhy událostí používaných v MIDI stopách. Nejdůležitější z nich jsou ty, které se zabývají zvukovou stránkou stopy. Mezi ně patří například zapnutí a vypnutí noty, či změny hlasitosti. Druhým typem jsou meta události, kam spadá například informace o tónině, tempu nebo třeba text písně či již zmíněná patička stopy. Posledním typem jsou sysex (systém exclusive) události. Jedná se o specifická data a parametry pro různá zařízení pracující s MIDI, která mohou být definovaná jako univerzální nebo stanovená výrobcem [9]. Sysex události nejsou v kontextu této práce podstatné.

Před každou událostí musí být časová značka, která určuje, kolik předem definovaných časových jednotek ji dělí od události předchozí. Časové značky jsou zapisovány ve formátu variable length quantity (VLQ), kde jsou data rozdělena po sedmi bitech. Do bajtů jsou uspořádána tak, že na pozici nejvýznamnějšího bitu se před každou sedmici zapíše buď nula, v případě že se jedná o poslední bajt, nebo jednička [12].

3.4 Java

Java je objektově orientovaný, silně typovaný a na platformě nezávislý programovací jazyk s automatickou správou paměti [13]. Počátky jeho vývoje sahají do roku 1990, kdy byl ve společnosti Sun Microsystems sestaven tým pod vedením Jamese Goslinga, který si dal za cíl vytvořit jazyk pro digitální domácí spotřebiče [14]. Skupina vývojářů, která si říkala Green Team, ho pojmenovala nejprve Greentalk a posléze Oak [15], název Java přišel až s rokem 1995. To už jazyk našel své využití, ovšem nikoli v embedded zařízeních, ale v takzvaných apletech na tehdy jinak statických webových stránkách. Od té doby jazyk prošel značným vývojem a dnes patří mezi nejpoužívanější programovací jazyky napříč všemi platformami [14].

3.4.1 Základní informace

Technologie Java je programovací jazyk a zároveň platforma, ve které aplikace napsané v programovacím jazyce běží [16]. Programovací jazyk Java stojí na pomezí mezi kompilovanými a interpretovanými jazyky. Zdrojový kód je kompilátorem přeložen do tzv. bytecode, který je poté interpretován virtuálním strojem (JVM – Java Virtual Machine). Opakovaně spouštěné části kódu navíc mohou být takzvaným Just-In-Time kompilátorem přeloženy do strojového kódu procesoru. Tím se dá běh aplikací napsaných v Javě značně zrychlit [17]. Z výše uvedeného vyplývá již zmíněná nezávislost na platformě. Ke spuštění Java aplikací je však potřeba mít nainstalováno Java Runtime Environment (JRE), což je běhové prostředí, jehož nedílnou součástí je již zmíněný virtuální stroj. JRE společně s nástroji nutnými pro vývoj aplikací pak dohromady tvoří vývojovou sadu JDK (Java Development Kit) [18].

3.4.2 Edice Javy

Existují čtyři edice platformy Java. První, a zřejmě nejznámější, je Java SE (Standard Edition), která poskytuje klíčovou funkcionalitu pro programovací jazyk a v něm napsané

aplikace od definice typů a objektů, až po vysokoúrovňové třídy pro práci v síti, zabezpečení a práci s databázemi. Součástí Java SE je také virtuální stroj, vývojové nástroje a nejrůznější knihovny využívané v Java aplikacích. Java EE (Entrprise Edition) je nadstavbou nad Java SE. Poskytuje API a běhové prostředí pro rozsáhlé a vícevrstvé, převážně webové aplikace, se zřetelem na jejich zabezpečení a spolehlivost. Java ME (Mobile Edition) nachází využití u aplikací pro malá zařízení, jako jsou mobilní telefony. Její API je v podstatě podmnožinou Java SE s přidanými speciálními knihovnami. Poslední z platforem je JavaFX, která může sloužit mimo jiné k vytváření RIA webových aplikací s využitím odlehčeného uživatelského rozhraní. Nyní je již plně integrovaná do edice Java SE. JavaFX aplikace využívají akcelerovanou grafiku klientů a současně čerpají výhody vysokoúrovňových API pro připojení ke vzdáleným zdrojům dat. Aplikace JavaME a JavaFX bývají často klienty pro služby v platformě Java EE [16].

3.4.3 **JavaFX aplikace**

JavaFX je soustava mediálních a grafických balíků pro vývoj multiplatformních aplikací. Využívá výhod oddělení kódu pro části aplikace, které se zobrazují uživatelům, od business logiky. Prezentační vrstva je napsána ve značkovacím jazyce FXML, případně ostylována pomocí CSS. Pro snadnou tvorbu uživatelského prostředí lze použít open source nástroj Scene Builder [19]. Ten byl nejprve poskytován společností Oracle, nicméně od vydání Java 8 update 40 přestaly být k dispozici instalační soubory a je zveřejněn pouze zdrojový kód v rámci projektu OpenJFX. Poté začala nabízet program k instalaci společnost Gluon. Rozdíly mezi původním JavaFX Scene Builderem a Gluon Scene Builderem jsou nepatrné a oba programy lze v podstatě považovat za totožné [20].

3.4.4 **JUnit**

Při vývoji jakékoli aplikace, snad s výjimkou těch opravdu malých, je velice užitečná možnost automaticky testovat její jednotlivé součásti a komponenty. V průběhu vývoje pak stačí po každé změně spustit tyto testy a ujistit se, zda byla zachována správná funkčnost jednotlivých komponent. Tento způsob testování se nazývá unit testing. Byla dokonce vymyšlena agilní metodika vývoje softwaru založená na unit testech. Je známá jako test-driven development a stojí na myšlence psaní unit testů ještě před samotným programováním testovaných úseků [21].

JUnit je framework pro automatické spouštění unit testů a reporting jejich výsledků vývojářům. Je specifickou instancí pro Javu testovací architektury xUnit, jejíž další inkarnace existují i pro jiné programovací jazyky. V JUnitu se píší testovací metody, které jsou součástí testovacích tříd. Jednotlivé stavy jsou porovnávány s očekávanými hodnotami pomocí příkazu assert uvnitř testovacích metod [21].

Je obtížné jednoduše odpovědět na otázku, na co všechno psát unit testy. Ze zkušenosti autora je častějším jevem nedostatek testů než jejich přebytek. To je pochopitelné, protože vývojář je placen za hotový produkt a pro zákazníka nejsou unit testy nijak užitečné. Dokáží však usnadnit a zrychlit vývoj, hrají roli v zabezpečení softwaru a často ulehčí práci při jeho rozšiřování. Při jejich psaní je doporučeno držet se tří zásad. Každý test by měl testovat pouze malou část kódu tak. aby bylo při selhání testu jasné, co přesně nefunguje. Průběh testu by neměl být závislý na externích zdrojích, jako je například připojení k databázi a provedení testů by mělo být rychlé [21].

3.5 Git

Při práci na jakémkoli větším projektu je potřeba udržet si přehled o provedených dílčích změnách a mít možnost je případně vrátit zpět. Je třeba mít kontrolu nad jednotlivými verzemi a dokázat je sloučit ve výsledný produkt. Za tímto účelem se používají takzvané systémy správy verzí (v angličtině version control system – VCS). Ty mohou být lokální, které spočívají v jednoduché databázi změn provedených ve vybraných souborech na konkrétním počítači, nebo centralizované. Ty uchovávají verzované soubory na serverové části, ze které si je jednotliví klienti stahují, a na kterou posléze nahrávají (anglicky commit) své změny. To řeší případy, kdy na projektu spolupracuje více lidí. Třetím typem jsou takzvaně distribuované systémy správy verzí, ve kterých nejsou data uložena na žádném centrálním serveru, ale každý pracovník má jejich kopii včetně všech předchozích verzí uloženou u sebe. To má velkou výhodu v tom, že se případně ztracená data a celá historie projektu dají obnovit z kopie od jakéhokoli pracovníka. U centralizovaných systémů toto možné není a je velký problém, pokud se ztratí nebo znepřístupní data uložená na centrálním serveru. Zástupcem distribuovaných VCS je právě systém Git [22].

3.5.1 Stručná historie

Git byl vytvořen týmem vývojářů spravujících linuxové jádro v čele s Linusem Torvaldsem. Jádro Linuxu bylo původně verzováno pomocí systému BitKeeper, který se stal

pro Git částečnou inspirací. Poté, co bylo v roce 2005 rozhodnuto o zpoplatnění BitKeeperu, začala komunita s vývojem svobodného VCS, který by splňoval požadavky potřebné pro tak velké projekty, jako je jádro Linuxu [22]. Od té doby se Git vypracoval na jeden z nejpopulárnějších systémů správy verzí, oblíbený je především ve světě svobodného softwaru [23].

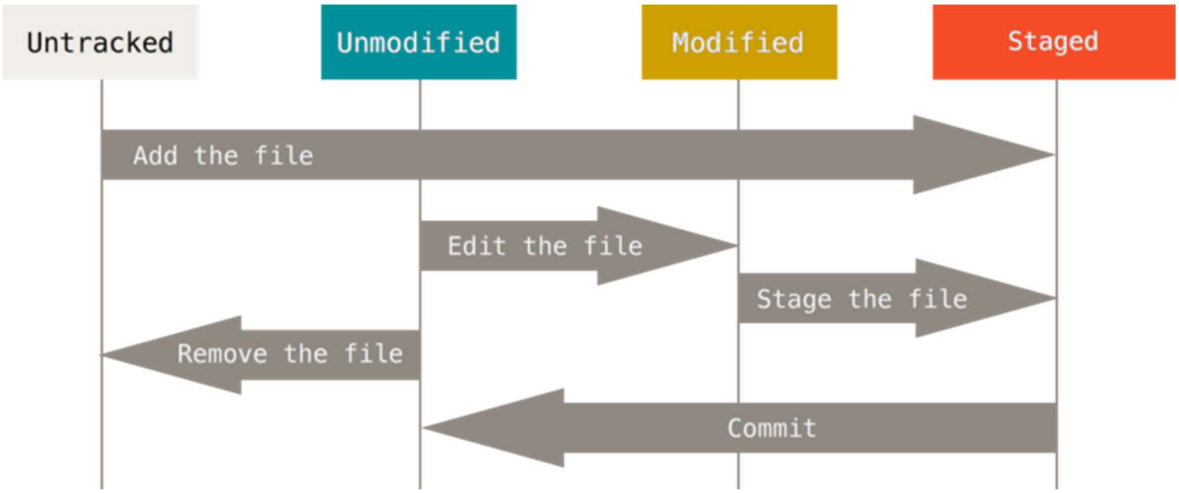
3.5.2 **Princip fungování**

Tato práce se rozsahem nijak nepřibližuje velkým projektům, které využívají více vývojových větví či jiných pokročilejších možností Gitu. Autor se ale rozhodl tento nástroj využít hlavně jako rychlý a jednoduchý způsob zálohování. V této práci tedy popisuje využití Gitu především za tímto účelem.

Jednotlivé verze ukládá Git ve formě jakýchsi snímků celého projektu. Do snímků jsou zahrnuty i soubory, ve kterých žádná změna nenastala, ale pouze ve formě odkazu na soubor z předchozího snímku. To vede k úspoře paměti. K veškerému verzovanému obsahu jsou mechanismem SHA-1 vytvářeny kontrolní součty, které slouží pro identifikaci souborů v databázi Gitu [22].

Verzovaný soubor se může nacházet v jednom ze tří stavů: commited (zapsáno), modified (změněno) a staged (připraveno k zápisu). Zapsaný soubor je uložen v databázi a pokud v něm nastane jakákoli změna, přejde do stavu modified. Uživatel poté může soubor přepnout do stavu připraveno k zápisu. Následně může uživatel provést takzvaný commit (zápis), který uloží změny v souborech ve stavu staged do databáze a u těchto souborů nastaví opět stav commited (Obrázek 7). Veškeré ovládání Gitu probíhá buď pomocí příkazového řádku, nebo grafického uživatelského prostředí [22].

Obrázek 7 – Životní cyklus souboru verzovaného systémem Git



Zdroj: [22]

Soubory ve stavu staged lze pomocí příkazu reset dostat zpět do stavu modified. Zahodit změny a vrátit se k libovolnému commitu lze pomocí příkazu checkout. Tento příkaz je často využíván také ke změně vybrané vývojové větve [22].

Jak již bylo výše zmíněno, Git je v této práci používán jako nástroj pro zálohování dat. K tomuto účelu je nutné umět pracovat se vzdálenými (remote) repozitáři. Ty lze vytvořit kdekoli v síti, často na internetových stránkách jako GitHub nebo GitLab, které se zabývají právě jejich hostingem. Do vytvořeného vzdáleného repozitáře lze lokální data odeslat příkazem push. Pokud na projektu pracuje více lidí, lze si lokální repozitář z toho vzdáleného aktualizovat pomocí příkazů fetch a pull. S nimi souvisí ještě integrační příkaz merge. Tyto příkazy nejsou v kontextu této práce příliš podstatné. Při potřebě práce na projektu z jiného počítače se pomocí příkazu clone na určeném místě vytvoří lokální repozitář, který se nalinkuje na repozitář vzdálený a zkopíruje si z něj veškerá data. To se hodí právě při případném selhání počítače s původním lokálním repozitářem [22],

3.6 Apache Maven

Apache Maven je silný nástroj, který standardizuje správu a sestavování aplikací napsaných převážně v jazyce Java. Maven řeší prakticky vše, co je třeba udělat na cestě od surového kódu ke zveřejněné fungující aplikaci. Projekt, který ho využívá, je nezávislý na použitém vývojovém prostředí a jeho nasazení je v mnohém usnadněno. Zásadní funkcí tohoto systému je, že za uživatele řeší veškeré závislosti na externích knihovnách [24].

Je na místě otázka, zdali je nutno při relativní jednoduchosti aplikace, která je výstupem této práce, použít takovýto nástroj a nestačilo by se spolehnout na možnosti

nabízené vývojovým prostředím. Patrně stačilo, ale podle zkušeností autora je použití nějakého nástroje pro správu buildů dnes již standardem, navíc se tím zjednoduší případný další vývoj této aplikace, pokud k němu dojde. Dle autorova názoru se použitím Mavenu nedá nic zkazit.

3.6.1 **Základní principy**

Maven používá soubor pom.xml, který určuje objektový model projektu (pom = project object model). Veškerá potřebná specifika projektu, pro který se v terminologii Mavenu používá pojem artefakt, jeho závislosti a případně nějaké další informace, jsou obsaženy právě v tomto souboru [25]. K ovládání Mavenu se používá příkazová řádka, nicméně vývojová prostředí také běžně skýtají možnost ovládání přes grafické uživatelské rozhraní, což plně dostačuje pro potřeby této práce.

Artefakt je jednoznačně určen trojicí identifikátorů groupid (skupina), artifactid (název artefaktu) a version (verze artefaktu). Jeho funkčnost může být závislá na jiných artefaktech spravovaných systémem Maven. Tyto závislosti, označené tagem dependency, obsahují kromě informací o skupině, názvu a verzi daného artefaktu, také vlastnost scope. Ta říká, kdy a v jaké míře je závislost potřebná [24]. Kromě závislostí jsou pro tuto práci důležité ještě pluginy, na jejichž spouštění v podstatě Maven stojí [26]. Veškeré pluginy a závislosti si Maven při práci stahuje ze vzdálených repozitářů do lokálního repozitáře v počítači uživatele [24].

3.6.2 **Životní cyklus**

Maven je založen na myšlence životního cyklu projektu. Součástí Mavenu jsou tři životní cykly, default, clean a site. Default řídí sestavení a zveřejnění, clean čištění a site se zabývá stránkou s dokumentací projektu. Každý cyklus se skládá z několika částí neboli fází, které se provádějí sekvenčně a při provedení jakékoli fáze (většinou pomocí příkazové řádky) se provedou všechny fáze předchozí. Cyklus default sestává z fází validate, compile, test, package, verify, install a deploy. Validate ověří, jestli projekt obsahuje veškeré potřebné informace. Compile provede kompilaci, test spustí unit testy. Package zkompilovaný kód zabalí do distribuovatelného formátu, jako je třeba soubor s koncovkou jar. Verify prověří výsledky integračních testů a install projekt nainstaluje do lokálního repozitáře pro použití jako závislost k jiným projektům. Deploy výsledný balík zkopíruje na vzdálený repozitář [27]. Často volaným příkazem při vývoji apliakcí podobných té, která je součástí této práce,

je mvn clean package, který provede čištění a poté všechny fáze cyklu default až po fázi package, která vytvoří spustitelný soubor.

3.7 IntelliJ IDEA

IntelliJ IDEA je vývojové prostředí pro široké spektrum programovacích jazyků od firmy JetBrains. Jeho klíčovou vlastností je rychlá hloubková kontrola kódu, která kromě odhalování syntaktických a sémantických chyb dokáže nalézt i pachy v kódu. Tato inspekce například dokáže identifikovat opakující se fragmenty, které by mohly být nahrazeny jednou metodou. To do jisté míry usnadňuje refaktoring, pro nějž nabízí také řadu dalších nástrojů. Mezi ostatní funkce patří chytré automatické dokončování kódu, zabudovaná podpora hlavních systémů správy verzí, jako je Git a SVN, podpora různých nástrojů pro sestavování, zabudovaný terminál, dekompilátor pro knihovny v Javě, podpora testovacích frameworků, databázových nástrojů a mnoho dalších [28].

IDEA existuje ve dvou edicích. K dispozici je buď omezená bezplatná verze s open-source licencí Apache 2.0, nebo verze komerční, která nabízí například výše zmíněnou kontrolu duplicit, podporu Java EE, JavaScriptu a SQL. Studenti a učitelé mají přístup ke komerční edici zdarma [29].

4 Vlastní práce

Tato kapitola se zabývá návrhem a implementací Java aplikace, která je schopná náhodně generovat hudbu. Kromě samotného algoritmu pro generování hudby zde autor postupně popisuje uživatelské rozhraní, datový model a řídící logiku aplikace. Protože se autor rozhodl nepoužívat žádnou cizí knihovnu pro práci s MIDI, součástí jsou také autorem napsané třídy pro export skladby do MIDI souboru.

4.1 Algoritmus pro procedurální generování hudby

Nejjednodušším způsobem algoritmické hudební kompozice, jaký autora napadl, je jakýsi „útok hrubou silou“. Spočívá v náhodném rozházení not v čase, které se bude opakovat tak dlouho, dokud nebude výsledek odpovídat představám zadavatele. Problém je ale v tom, že počítač není schopen ohodnotit kvalitu skladby a poznat tak, kdy má s náhodným generováním skončit. Je ale možné tento postup omezit pravidly, při jejichž dodržování bude výsledek poslouchatelný hned na první pokus.

4.1.1 Harmonická, motivická a rytmická struktura

Jako základní jednotku pro stavbu výsledné skladby si autor vybral osmitaktovou hudební periodu. Celá skladba je tvořená určitým počtem period poskládaných za sebou. V rámci každé periody, která zde sestává z melodie a jednoduchého doprovodu, si autor vymezil několik struktur, které splňují jistá pravidla. Ty představují jakýsi rámec pro generování již zmíněné melodie a doprovodu.

První z nich je harmonická struktura. V případě této práce je velice jednoduchá, tvoří ji řada kvintakordů postavených na tónice, subdominantě, nebo dominantě. Řada by měla začínat buď tónikou nebo dominantou a končit musí vždy v tónice. Se začátkem každého taktu se musí harmonický stupeň změnit. Při dodržení výše zmíněného je řada generována náhodně.

Motivická struktura poskytuje informaci o podobnostech mezi některými částmi periody. Je to řada motivů (částí) dlouhých jeden nebo dva takty. Motiv může být kopií některého z předešlých a může být oproti kopii zapsán pozpátku. Žádný motiv by neměl být kopírován více než jednou. Motivy jsou generovány náhodně tak, aby součet jejich délek byl osm taktů.

Jako rytmická struktura je zde chápána řada délek tónů, kterou lze vnitřně rozdělit na osm taktů. Rytmická struktura by měla dodržovat strukturu motivickou, jednotlivé takty či dvojice taktů by měly kopírovat jiné stejným způsobem. Délky tónů jsou určeny náhodným výběrem mezi půlovou, čtvrťovou a osminovou notou. Poslední tón by měl mít délku noty celé nebo půlové.

4.1.2 **Melodie a doprovod**

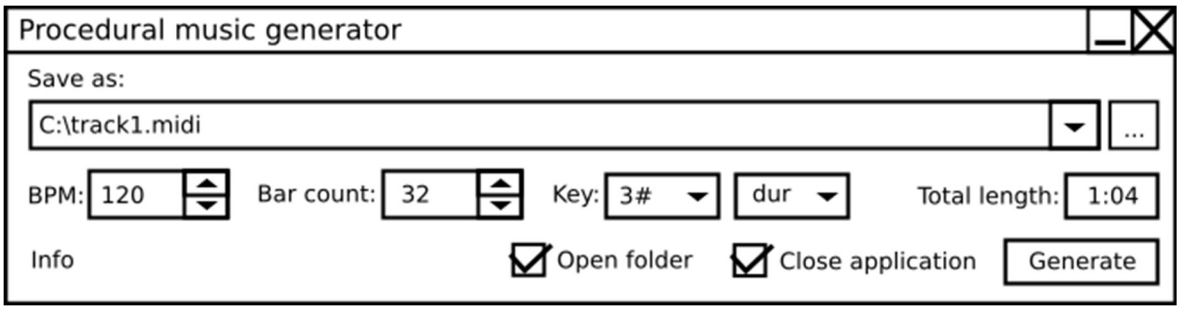
Po stanovení rytmické a harmonické struktury lze přejít ke generování melodie. Kvůli zjednodušení má motivická struktura na podobu melodie pouze nepřímý vliv skrze rytmickou strukturu. Ta určuje délku jednotlivých tónů melodie. Jejich výška je vybírána náhodně, tak aby melodie opisovala harmonickou strukturu. To znamená, že tóny, které připadnou na těžkou či lehkou dobu a tóny, ke kterým melodie dojde pomocí skoku, by měly být součástí odpovídajícího kvintakordu v harmonické struktuře. Dále přišlo autorovi vhodné dodržet pravidlo, že poslední tón celé skladby musí být tónem, na kterém je postavena tónika.

Doprovod je dán harmonickou strukturou, na začátku každého taktu je zahrán odpovídající kvintakord.

4.2 **Návrh a implementace uživatelského rozhraní**

U aplikace, která náhodně generuje hudbu, patrně ergonomie nehraje takovou roli, jako například u různých kancelářských aplikací. Pro kvalitní výsledek je ale i zde nutné se zabývat uživatelským zážitkem a nezanedbat návrh uživatelského rozhraní. V tomto případě se jedná o jedno relativně jednoduché okno (Obrázek 8), ke kterému se vztahuje jeden scénář a jeden use case.

Obrázek 8 – Wireframe



Zdroj: autor

4.2.1 **Use case**

Uživatel očekává možnost výběru adresáře, do kterého se bude generovaná skladba ukládat, možnost zadání názvu generované skladby a zobrazení cesty ke skladbě v souborovém systému. Uživatel očekává možnost zadat tempo, tóninu a počet taktů generované skladby. Uživatel také očekává zobrazení celkové délky skladby. Uživatel požaduje tlačítko, po jehož stisknutí bude skladba vygenerována a standardní lištu s tlačítky minimalizace a ukončení aplikace. Uživatel očekává možnosti automatického ukončení aplikace a otevření složky s vygenerovanou skladbou. Dále také očekává rychlou možnost přechodu na webovou stránku s informacemi o projektu.

4.2.2 **Scénář**

Systém zobrazí combobox s cestou ke generované skladbě. Kromě výběru ze tří přednastavených adresářů (Hudba, Dokumenty a adresář ve kterém se nachází aplikace) bude poskytovat i možnost ručně vyplnit cestu k cílové složce. Vedle comboboxu systém zobrazí tlačítko, po jehož stisknutí uživatelem se otevře dialogové okno, ve kterém může uživatel vybrat název a cílový adresář pro generovanou skladbu způsobem, na který je zvyklý z jiných aplikací.

Ve výchozím stavu bude jako cílový adresář vybrána složka Hudba. Název skladby bude zpočátku nastaven na „track“ + nejvyšší číslo skladby uložené ve zvoleném adresáři zvýšené o jedna. Pokud v tomto adresáři žádná skladba nebude, nastaví se číslo 1. Systém zobrazí dva spinnery. Pomocí prvního může uživatel nastavit tempo generované skladby, pomocí druhého potom počet taktů, tak aby to bylo číslo dělitelné osmi (nejmenší část skladby – perioda má osm taktů).

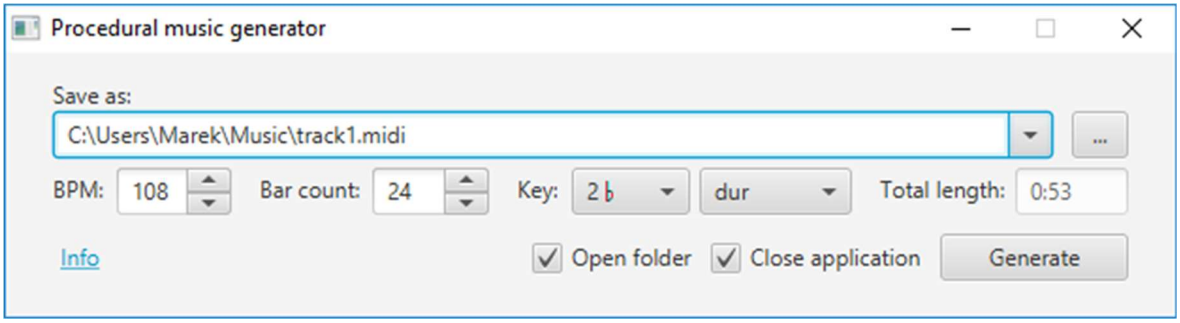
Systém zobrazí dva comboboxy, pomocí kterých bude moci uživatel nastavit tóninu generované skladby. V prvním comboboxu bude výběr z možností posunutí oproti základnímu tónu pomocí symbolů # a ♭, ve druhém pak výběr mezi durovou a mollovou variantou tóniny. Systém zobrazí needitovatelné textové pole s celkovou dobou trvání generované skladby, která se vypočítá ze zadaného tempa a počtu taktů, systém toto textové pole překreslí s aktuální dobou trvání pokaždé, když uživatel změní tempo nebo počet taktů. Systém zobrazí tlačítko Generate, po jehož stisknutí bude do vybraného adresáře vytvořen soubor MIDI se skladbou s uživatelem zadanými parametry.

Systém zobrazí systémovou lištu s tlačítky minimalizace a ukončení aplikace. Systém zobrazí dva checkboxy, jejichž zaškrtnutí uživatelem bude určovat, zdali se po vygenerování skladby aplikace ukončí a jestli se otevře okno průzkumníku s vygenerovanou skladbou. Systém zobrazí hypertextový odkaz na webovou stránku s informacemi o aplikaci a se vzdáleným repozitářem. Tempo a tónina budou při spuštění aplikace nastaveny náhodně.

4.2.3 **Implementace uživatelského rozhraní**

Uživatelské rozhraní (Obrázek 9) bylo vytvořeno na platformě JavaFX pomocí programu SceneBuilder. Uloženo je v souboru mainForm.fxml. Reakce na jeho ovládání uživatelem má dále na starosti třída Controller, která bude popsána níže.

Obrázek 9 – Uživatelské rozhraní



Zdroj: autor

4.3 Struktura aplikace

Při návrhu aplikace se autor do jisté míry inspiroval architekturním vzorem MVC a snažil se tak oddělit uživatelské rozhraní od vlastní řídící logiky. Zde jsou popsány jednotlivé třídy a jejich funkce.

4.3.1 Main

Tato základní třída (Zdrojový kód 1) má na starosti spuštění aplikace. Obsahuje hlavní statickou metodu main a kromě ní ještě metodu start, která je v případě platformy JavaFX jakýmsi vstupním bodem. Tato metoda načte uživatelské rozhraní ze souboru mainForm.fxml a zobrazí ho doprostřed obrazovky.

Zdrojový kód 1 – Třída Main

```
package mainForm;

import javafx.application.Application;
import javafx.application.HostServices;
import javafx.fxml.FXMLLoader;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.stage.Stage;

public class Main extends Application {

    private static HostServices hostServices;

    public static HostServices getMainHostServices() {
        return hostServices ;
    }

    @Override
    public void start(Stage primaryStage) throws Exception{
        hostServices = getHostServices();

        Parent root = FXMLLoader.load(getClass().getResource("/mainForm.fxml"));

        primaryStage.setTitle("Procedural music generator");
        primaryStage.setScene(new Scene(root));
        primaryStage.centerOnScreen();
        primaryStage.setResizable(false);
        primaryStage.show();
    }



    public static void main(String[] args) {
        launch(args);
    }
}
```

Zdroj: autor

4.3.2 **Model**

Třída Model reprezentuje část datového modelu, která přímo souvisí s uživatelským rozhraním. Jsou v ní nadefinovány proměnné, jako je tempo, počet taktů, či cesta ke generované skladbě. Tato třída je implementována jako singleton (jedináček). To znamená, že se v celém programu vyskytne pouze jedna její instance. Její konstruktor (Zdrojový kód 2) je neveřejný a k instanci se přistupuje pomocí metody getInstance.

Zdrojový kód 2 – Konstruktor třídy Model

```
private Model() {
    Random rand = new Random();

    this.bpm = 100 + (int)Math.round(rand.nextGaussian() * 15);
    if (this.bpm < MIN_BPM){
        this.bpm = MIN_BPM;
    } else if (this.bpm > MAX_BPM){
        this.bpm = MAX_BPM;
    }

    this.barCount = 24;
    this.keys = FXCollections.observableArrayList("7♭","6♭","5♭","4♭","3♭","2♭","♭","-",
                                                  "#","2#","3#","4#","5#","6#","7#");
    this.modes = FXCollections.observableArrayList("dur","moll");

    this.modeIndex = Math.abs(rand.nextInt()) % 2;

    this.keyIndex = 7 + (int)Math.round(rand.nextGaussian() * 2);
    if (this.keyIndex < 0){
        this.keyIndex = 0;
    } else if (this.keyIndex >= this.keys.size()){
        this.keyIndex = this.keys.size() -1;
    }

    this.openFolder = true;
    this.closeApp = true;

    this.filePaths = FXCollections.observableArrayList();

    Path path = Paths.get(System.getProperty("user.home") + "\\Music");
    if (Files.exists(path))
        this.filePaths.add(path.toString());

    path = Paths.get(System.getProperty("user.home") + "\\Documents");
    if (Files.exists(path))
        this.filePaths.add(path.toString());

    this.filePaths.add(System.getProperty("user.dir"));

    this.targetFilePath = targetDistinctFilePath(this.filePaths.get(0));
}
```

Zdroj: autor

4.3.3 Controller

Interakci uživatele a grafického prostředí zpracovává třída Controller. Implementuje rozhraní Initializable a pomocí metody initialize zajišťuje prvotní nastavení formuláře. Controller má také na starosti nastavování proměnných instance třídy Model podle hodnot zadaných uživatelem a aktualizaci zobrazených informací. Kliknutí uživatele na tlačítko Generate zpracovává následující zdrojový kód (Zdrojový kód 3).

Zdrojový kód 3 – Metoda handleBtnGenerateAction ve třídě Controller

```
@FXML
private void handleBtnGenerateAction(){

    try {
        File file = new File(model.getTargetFilePath());
        if (!file.isDirectory()){
            file = file.getParentFile();
        }

        if (file == null){
            throw new FileNotFoundException("Invalid path!");
        }

        Generator generator = new Generator(model.getBpm(),
                model.getBarCount(),
                model.getKeys().indexOf(cmbKey.getValue()) -7,
                (model.getModes().indexOf(cmbMode.getValue()) == 1));

        generator.generate(model.getTargetFilePath());

        if (model.getOpenFolder()){
            Desktop.getDesktop().open(file);
        }

        if (model.getCloseApp()){
            System.exit(0);
        }

        String str = Paths.get(model.getTargetFilePath()).getParent().toString();
        model.setTargetFilePath(model.targetDistinctFilePath(str));

    } catch (IOException e){
        showError(e.getMessage());
    } finally {
        updateForm();
    }
}
```

Zdroj: autor

4.3.4 Generator

Instance třídy Generator je vytvořena Controllerem po stisknutí tlačítka Generate. Poté, co je zavolána její metoda generate (Zdrojový kód 4), vytvoří se hudební skladba. Ta je pomocí metody writeScoreToMIDITrack zapsána do MIDI stopy a uložena do souboru. Pro vygenerování skladby jsou využívány metody statické třídy GeneratorUtil.

Zdrojový kód 4 – Metoda generate ve třídě Generator

```
public void generate(String path) throws IOException {
    MIDIFile midiFile = new MIDIFile();

    MIDITrack tempoTrack = new MIDITrack();

    tempoTrack.getEventsList().add(MIDIEvent.timeSignature(0, 4, 4));
    tempoTrack.getEventsList().add(MIDIEvent.keySignature(0, keyShift, !moll));
    tempoTrack.getEventsList().add(MIDIEvent.tempoSignature(0, bpm));
    tempoTrack.getEventsList().add(MIDIEvent.trackFooter(1));
    midiFile.getTrackList().add(tempoTrack);

    MIDITrack musicTrack = new MIDITrack();
    musicTrack.getEventsList().add(MIDIEvent.programChange(0, 0,1));

    Score song = new Score();

    int periodCount = barCount / 8;
    for (int i = 0; i < periodCount; i++) {
        song.add(GeneratorUtil.getRandomPeriod(getTones(), i == periodCount -1));
    }

    song.keyShift(keyShift);

    writeScoreToMIDITrack(song, musicTrack);
    musicTrack.getEventsList().add(MIDIEvent.trackFooter(0));
    midiFile.getTrackList().add(musicTrack);
    midiFile.generateMIDIFile(path);
}
```

Zdroj: autor

4.3.5 **Note**

Třída Note (Zdrojový kód 5) reprezentuje hudební notu. Jsou v ní nadefinovány dvě proměnné, tone a duration, které určují výšku tónu a dobu trvání. Proměnná tone je typu integer a její hodnota odpovídá číslu tónu podle specifikace MIDI. Pokud je hodnota rovna -1, znamená to, že se jedná o pomlku. Proměnná duration je typu double a její hodnota v rozsahu 0 až 1 odpovídá délce tónu v poměru s jedním taktem. V této aplikaci se vyskytují pouze celé, půlové, čtvrťové a osminové noty.

Zdrojový kód 5 – Třída Note

```
package mainForm;

public class Note{
    private int tone;
    private double duration;

    public int getTone() {
        return tone;
    }
    public void setTone(int tone) {
        this.tone = tone;
    }

    public double getDuration() {
        return duration;
    }

    Note(int tone, double duration) {
        this.tone = tone;
        this.duration = duration;
    }
}
```

Zdroj: autor

4.3.6 **NoteList**

Jednohlasá melodie je reprezentována třídou NoteList (Zdrojový kód 6), která je potomkem třídy ArrayList<Note>. Jedná se tedy o jednorozměrný seznam objektů typu Note.

Zdrojový kód 6 – Třída NoteList

```
package mainForm;

import java.util.ArrayList;

public class NoteList extends ArrayList<Note> {

    public double length(){
        double length = 0;
        for (Note note : this) {
            length = length + note.getDuration();
        }
        return length;
    }

    public void addNotes(NoteList noteList){
        for (Note note : noteList) {
            add(new Note(note.getTone(), note.getDuration()));
        }
    }

    public void fillRests(double targetLength){
        double diff = targetLength - length();
        while (diff > 0.01){
            double restDuration;
            if (diff >= GeneratorUtil.getWHOLE()){
                restDuration = GeneratorUtil.getWHOLE();
            } else if(diff >= GeneratorUtil.getHALF()){
                restDuration = GeneratorUtil.getHALF();
            } else if(diff >= GeneratorUtil.getQUARTER()){
                restDuration = GeneratorUtil.getQUARTER();
            } else if(diff >= GeneratorUtil.getEIGHTH()){
                restDuration = GeneratorUtil.getEIGHTH();
            } else{
                restDuration = GeneratorUtil.getSIXTEENTH();
            }
            add(new Note(-1, restDuration));
            diff = targetLength - length();
        }
    }
}
```

Zdroj: autor

4.3.7 **Score**

Třída Score je potomkem třídy ArrayList<NoteList> a je tedy jakýmsi seznamem seznamů not. Reprezentuje více hlasů znějících dohromady. Výsledná skladba před převodem do formátu MIDI je instancí třídy Score. Tato třída obsahuje několik metod usnadňujících práci se skladbou. Jednou z nich je keyShift (Zdrojový kód 7), která slouží k chromatickému posunu všech obsažených not. Díky tomu lze snadno skladbu převést do jiné tóniny.

Zdrojový kód 7 – Metoda keyShift ve třídě Score

```
public void keyShift(int value){
    value = value % 8;

    for (NoteList voice : this) {
        for (Note n : voice) {
            if (n.getTone() != -1) {
                n.setTone(n.getTone() + value);

                while (n.getTone() < 0){
                    n.setTone(n.getTone() + 12);
                }

                while (n.getTone() > 127){
                    n.setTone(n.getTone() -12);
                }
            }
        }
    }
}
```

Zdroj: autor

4.3.8 **Motif**

Hudební motivy v této aplikaci zaštiťuje třída Motif (Zdrojový kód 8). Každý motiv je dlouhý určitý počet taktů a může být buď původní, nebo odvozený od jiného motivu. Pokud je odvozený, může být vůči původnímu motivu zapsaný pozpátku. Vzhledem k tomu, že se autor nakonec rozhodl využívat pouze rytmickou strukturu motivů, nejsou zde definovány inverzní a jiné motivy se změnou ve výškách tónů.

Zdrojový kód 8 – Třída Motif

```
package mainForm;

public class Motif {
    private Motif relatedMotif;
    private boolean retrograde;
    private int barCount;

    Motif(Motif relatedMotif, boolean retrograde, int barcount) {
        this.relatedMotif = relatedMotif;
        this.retrograde = retrograde;
        this.barCount = barcount;
    }

    public Motif getRelatedMotif() {
        return relatedMotif;
    }

    public boolean isRetrograde() {
        return retrograde;
    }


    public int getBarCount() {
        return barCount;
    }
}
```

Zdroj: autor

4.3.9 GeneratorUtil

GeneratorUtil je statická třída obsahující logiku nutnou pro samotné generování. Jsou v ní deklarovány konstanty s používanými délkami tónů, indexy tónů durové a mollové stupnice a metody, pomocí nichž je hudební skladba vytvořena. Jednou z nich je metoda getRandomPeriod (Zdrojový kód 9), která vytváří jednotlivé hudební periody.

Jak bylo popsáno výše, nejprve jsou vygenerovány harmonická, motivická a rytmická struktura. Ty slouží jako vstup pro vygenerování melodie a doprovodu tvořících hudební periodu. Výsledná skladba je zde řada hudebních period poskládaných za sebou.

Zdrojový kód 9 – Metoda getRandomPeriod ve třídě GeneratorUtil

```
public static Score getRandomPeriod(int [] tones, boolean lastNoteIsTonic){
    int lowestMelodyTone = 60;
    int highestMelodyTone = 84;
    int lowestAccompanimentTone = 48;

    ArrayList<Motif> motivicStructure = getMotivicStructure();
    ArrayList<ArrayList<Double>> rhythmicStructure =
          getRhythmicStructure(motivicStructure);
    int [] harmonicStructure = getHarmonicStructure();
    NoteList melody = getRandomMelody(tones,
                                      motivicStructure,
                                      harmonicStructure,
                                      rhythmicStructure,
                                      lowestMelodyTone,
                                      highestMelodyTone,
                                      lastNoteIsTonic);

    Score result = new Score();
    result.add(melody);

    result.add(new NoteList());
    result.add(new NoteList());
    result.add(new NoteList());

    for (int harmonicDegree : harmonicStructure) {
        int[] chord = getChord(tones, harmonicDegree);
        for (int j = 1; j <= 3; j++) {
            Note n = new Note(chord[j -1] + lowestAccompanimentTone, 1);
            result.get(result.size() -j).add(n);
        }
    }

    return result;
}
```

Zdroj: autor

4.3.10 MIDIEvent

Jak bylo zmíněno v teoretické části, základním kamenem MIDI souboru je takzvaná událost. Ta je zde reprezentována třídou MIDIEvent. Je v ní deklarována proměnná typu integer deltaPPQ, která určuje časový posun oproti předchozí události. Dále je zde deklarováno pole typu integer s názvem data. Toto pole reprezentuje samotný obsah MIDI události. Třída obsahuje několik statických továrních metod, které umožňují snadnou

konstrukci běžně používaných MIDI událostí. Nachází se zde také metoda getBytes (Zdrojový kód 10), která se stará o převod události do podoby definované formátem MIDI.

Zdrojový kód 10 – Metoda getBytes ve třídě MIDIEvent

```
public byte[] getBytes(){
    byte[] buffer = new byte[byteCount()];
    long deltaVLQ = MIDIUtil.encodeVLQ(deltaPPQ);
    byte[] deltaBytes = MIDIUtil.longToBytes(deltaVLQ, MIDIUtil.byteCount(deltaVLQ));

    for (int i = 0; i < buffer.length; i++){
        if (i < deltaBytes.length){
            buffer[i] = deltaBytes[i];
        } else {
            buffer[i] = (byte) data[i - deltaBytes.length];
        }
    }

    return buffer;
}
```

Zdroj: autor

4.3.11 **MIDITrack**

Třída MIDITrack reprezentuje MIDI stopu. Ta kromě hlavičky obsahuje seznam MIDI událostí, který zde představuje proměnná eventList typu ArrayList<MIDIEvent>. Stejně jako třída MIDIEvent, i tato třída obsahuje metodu getBytes (Zdrojový kód 11) zajišťující převod do formátu MIDI.

Zdrojový kód 11 – Metoda getBytes ve třídě MIDITrack

```
public byte[] getBytes(){
    final int lengthByteCount = 4;

    byte[] dataLength = MIDIUtil.longToBytes(dataLength(), lengthByteCount);
    int byteCount = trackHeader.length + dataLength.length;

    for (MIDIEvent event : eventsList) {
        byteCount = byteCount + event.byteCount();
    }

    byte[] buffer = new byte[byteCount];

    System.arraycopy(MIDIUtil.intArrayToByteArray(trackHeader), 0, buffer, 0,
          trackHeader.length);
    System.arraycopy(dataLength, 0, buffer, trackHeader.length, dataLength.length);

    int pos = trackHeader.length + dataLength.length;
    for (MIDIEvent event : eventsList) {
        byte[] eventBuffer = event.getBytes();
        System.arraycopy(eventBuffer, 0, buffer, pos, eventBuffer.length);
        pos = pos + eventBuffer.length;
    }

    return buffer;
}
```

Zdroj: autor

4.3.12 **MIDIFile**

MIDIFile je třída zastupující celý MIDI soubor. Je zde deklarována proměnná trackList typu ArrayList<MIDITrack>, která představuje seznam stop. Mimo jiné se zde nachází také metoda generateMIDIFile (Zdrojový kód 12), která má na starosti uložení skladby do souboru zadaného ve vstupním parametru.

Zdrojový kód 12 – Metoda generateMIDIFile ve třídě MIDIFile

```
public void generateMIDIFile(String pathname) throws IOException {
    File output = new File(pathname);

    FileOutputStream fout = new FileOutputStream(output);

    // header
    fout.write(MIDIUtil.intArrayToByteArray(fileHeader));
    fout.write(MIDIUtil.longToBytes(TYPE_BYTECOUNT + TRACKCOUNT_BYTECOUNT +
          PPQ_BYTECOUNT, LENGTH_BYTECOUNT));
    fout.write(MIDIUtil.longToBytes(TYPE, TYPE_BYTECOUNT));
    fout.write(MIDIUtil.longToBytes(trackList.size(), TRACKCOUNT_BYTECOUNT));
    fout.write(MIDIUtil.longToBytes(PPQ, PPQ_BYTECOUNT));

    for (MIDITrack track : trackList) {
        fout.write(track.getBytes());
    }

    fout.flush();
    fout.close();
}
```

Zdroj: autor

4.3.13 **MIDIUtil**

Poslední třídou v tomto projektu je MIDIUtil. Je statická a obsahuje metody usnadňující práci při vytváření MIDI souborů. Příkladem budiž metoda encodeVLQ (Zdrojový kód 13), která převádí čísla do formátu variable length quantity.

Zdrojový kód 13 – Metoda encodeVLQ ve třídě MIDIUtil

```
public static long encodeVLQ(long val) {
    double output = 0;
    double value = val;
    int counter = 0;
    final int bitCount = 7;
    int exponent = 0;
    double signBit;
    double modulo;

    do {
        exponent = exponent + bitCount;
        signBit = Math.pow(2, exponent);
        modulo = (value % signBit);
        value = value - modulo;
        output = output + (modulo * Math.pow(2, counter));

        if (counter > 0) {
            output = output + (signBit * Math.pow(2, counter));
        }

        counter++;
    }
    while (value >= signBit);

    return (long) output;
}
```

Zdroj: autor

5 Výsledky a diskuse

Očekávaným problémem, na který autor během vývoje aplikace narážel, je nutnost převádět výukové materiály o hudební kompozici do formálnějšího znění pro potřeby algoritmizace. Zdroje, které měl autor k dispozici, navíc u čtenáře předpokládaly jistou fantazii a schopnost posoudit kvalitu hudby. To lze jen těžko nahradit pseudonáhodnými čísly a vykonáváním operací zadaných programátorem.

Autor se nejprve pokusil vytvářet skladatelské algoritmy na základě striktního dodržování toho, co se dočetl v učebnici Stanislava Jelínka Skladba – úvod do hudební kompozice. Takto vygenerované skladby však z výše popsaných důvodů působily značně neuspokojivě. Zmíněná publikace se celkem rychle dostává až ke tvorbě komplexních hudebních forem, jako je například sonáta. Zde bylo ale třeba se soustředit na to nejzákladnější – melodii, harmonii a rytmus. Autor tedy program přepsal tak, aby generoval jednodušší skladby s důrazem na vzájemný soulad těchto tří složek. Poté se výsledky zlepšily.

Hotová aplikace dokáže generovat skladby, které složitostí příliš nepřevyšují lidové písně, ovšem nevyskytuje se v nich nadměrné množství disonancí a jejich melodie opisují jasně danou harmonickou strukturu. To může být dobrým základem pro další rozvoj.

5.1 Možnosti rozšíření aplikace

Jedním z možných dalších kroků ve vývoji aplikace by mohla být schopnost pracovat s vedlejšími harmonickými funkcemi, například se spodní mediantou nebo střídavou dominantou. Harmonická struktura by se také mohla měnit i jindy než se začátkem taktu. V některých případech by bylo vhodně rozšířit použité kvintakordy na septakordy. Rytmická struktura postavená na jednoduchém čtyřčtvrťovém taktu je také pouze základem, který by v budoucnu mohl být vylepšen. Existuje celá řada rytmů, které by se zde daly použít. K jejich generování by mohl zčásti posloužit Euklidův algoritmus [30].

Do budoucna by mohla být rozšířena implementace motivické práce, tak aby aplikace s motivy pracovala stejným způsobem, jako skladatel – člověk. Také samotný koncept jedné melodie a doprovodu může být rozvinut ve složitější polyfonii aplikováním pravidel kontrapunktu. Poněkud vzdálená se jeví možnost využít v této aplikaci strojové učení, nicméně byla by to cesta, jak při generování pracovat s jinak těžko popsatelnou subjektivní lidskou představou o hudební kvalitě.

Vylepšit by se dalo také samotné přehrávání. Pokud by autor využil pro zápis do MIDI souboru standardní Java Sound API, setkal by se s problémem, kdyby v budoucnu chtěl aplikaci portovat na Android [31]. Proto raději použil vlastní třídy pro práci s MIDI. Vylepšení by zde spočívalo v použití různých nástrojů či implementaci dynamiky. Základní vlastností MIDI souborů je, že jejich zvuk určuje zvuková karta, na které jsou přehrávány. V budoucnu by však součástí této aplikace mohl být VSTi sampler, který by podle MIDI not přehrával zvuky skutečných nástrojů.

V rámci dalšího vývoje této aplikace by také bylo vhodné zabývat se platformou, na které je spuštěna. V současné podobě byla aplikace testována na operačních systémech Windows 7, Windows 8.1 a Windows 10. Jelikož byla napsána v jazyce Java, kompatibilita s jinými operačními systémy by neměla činit zásadní problémy a je zde možnost vytvořit její port pro mobilní operační systém Android. Dále by bylo možno tento program použít jako základ pro webovou aplikaci. Generování hudby by tak bylo uživatelům dostupné z kteréhokoli zařízení s připojením na internet a webovým prohlížečem.

6 Závěr

V rámci této bakalářské práce se autor snažil prozkoumat možnosti algoritmického generování hudby bez přispění lidské tvořivosti. Hlavním cílem bylo vytvořit aplikaci, která dokáže složit jednoduchou skladbu a uložit jí do MIDI souboru. Za tímto cílem autor postupoval podle metodiky.

Nejprve v kapitole Teoretická východiska představil základy hudební teorie, na které celá logika později vytvořené aplikace stojí. Dále v této kapitole popsal zvukový formát MIDI a technologie, které při vývoji aplikace využil. Mezi ně patří programovací jazyk Java 8 spolu se souvisejícími nástroji, jako je testovací framework JUnit či grafická platforma JavaFX. Kromě toho se také v teoretické části věnoval nástrojům Apache Maven a Git společně s vývojovým prostředím IntelliJ IDEA.

V kapitole Vlastní práce autor nastínil způsob, jak k algoritmické kompozici přistupovat, představil návrh a implementaci uživatelského rozhraní a posléze i celé aplikace. Ta byla úspěšně otestována na operačních systémech Windows 7, Windows 8.1 a Windows 10. V rámci této kapitoly jsou stručně popsány jednotlivé třídy aplikace spolu s ukázkami zdrojového kódu.

V kapitole Výsledky a diskuze autor popsal cestu, kterou k výslednému použitému algoritmu došel a navrhl několik možných budoucích vylepšení hotové aplikace. Zveřejnění zdrojových kódů pod svobodnou licencí MIT na serveru Github podporuje tyto možnosti dalšího vývoje.

7 Seznam použitých zdrojů

1. **GRIGOVÁ, Věra.** *Všeobecná hudební nauka.* Olomouc : ALDA, 1998. ISBN 80-85600-46-3.
2. **NEUŽIL, Jiří, BENKO, Matěj.** Úvodní skripta (nejen) pro uchazeče o studium oboru Hra na klavír na KJJ: 1. díl – základy hudební teorie. [Online] [Citace: 18. 11 2017.] http://download.kjj.cz/pub/vyuka/2010/kl/skripta_kl.pdf.
3. **HYBRID PEDAGOGY PUBLISHING.** *Open Music Theory.* [Online] [Citace: 18. 11 2017.] http://openmusictheory.com/.
4. **Ottův slovník naučný.** Praha : J. Otto, 1888 - 1909.
5. **ZENKL, Luděk.** *ABC hudebních forem.* Praha : Editio Supraphon, 1984. ISBN 09/22 02-012-84.
6. **JELÍNEK, Stanislav.** *Skladba - úvod do hudební kompozice.* Kladno : Vladimír Beneš, 2009. ISBN: 978-0-706512-77-8.
7. **WILLIAMS, Victoria.** Composing a Melody: General Tips. *MyMusicTheory.* [Online] 29. 3 2016. [Citace: 3. 2 2018.] https://www.mymusictheory.com/learn-music-theory/for-students/grade-5/58-12-composing-a-melody-general-tips.
8. —. MyMusicTheory. *Composition - Cadences.* [Online] 28. 9 2011. [Citace: 5. 2 2018.] https://www.mymusictheory.com/learn-music-theory/for-students/grade-6/online-course/179-b3-composition-cadences.
9. **SCHIMMEL, Jiří.** Komunikační rozhraní MIDI. *Elektrorevue: Časopis pro elektrotechniku, č. 69.* [Online] 2002. [Citace: 19. 11 2017.] http://www.elektrorevue.cz/clanky/02069/index.html.
10. **ELLINGER, John.** Standard MIDI Files (SMF). *Introduction to Music Technology.* [Online] 9 2013. [Citace: 19. 11 2017.] http://acad.carleton.edu/courses/musc108-00-f14/pages/04/04StandardMIDIFiles.html.
11. **The MIDI Association.** Getting Started with MIDI: About MIDI – Part 4: MIDI Files. [Online] [Citace: 19. 11 2017.]
12. **BACK, David.** Standard MIDI-File Format Spec. 1.1, updated. [Online] 1999. [Citace: 19. 11 2017.] http://www.music.mcgill.ca/~ich/classes/mumt306/StandardMIDIfileformat.html.
13. **SEMECKÝ, Jiří.** Naučte se Javu – úvod. *Interval.cz.* [Online] 2002. [Citace: 20. 11 2017.] https://www.interval.cz/clanky/naucte-se-javu-uvod/.
14. **NOVOTNÝ, Luděk.** Historie a vývoj jazyka Java. [Online] 2003. [Citace: 20. 11 2017.] http://www.fi.muni.cz/usr/jkucera/pv109/2003p/xnovotn8.htm.
15. **Javatpoint.** *History of Java.* [Online] [Citace: 20. 11 2017.] https://www.javatpoint.com/history-of-java.
16. **ORACLE.** *Your First Cup: An Introduction to the Java EE Platform.* [Online] 2012 4. [Citace: 28. 12 2017.] https://docs.oracle.com/javaee/6/firstcup/doc/gkhoy.html.
17. **TRONÍČEK, Zdeněk.** *Učebnice jazyka Java.* [Online] 30. 09 2011. [Citace: 20. 11 2017.] http://java.cz/article/ucebnicejazykajava.

18. **JavatPoint.** *Difference between JDK, JRE and JVM.* [Online] [Citace: 20. 11 2017.] https://www.javatpoint.com/difference-between-jdk-jre-and-jvm.

19. **PAWLAN, Monica.** Oracle. *What is JavaFX.* [Online] [Citace: 28. 12 2017.] https://docs.oracle.com/javafx/2/overview/jfxpub-overview.htm.

20. **SCHULZ, Bennet.** Bye Bye JavaFX Scene Builder, Welcome Gluon Scene Builder 8.0.0. *DZone.* [Online] 15. 3 2015. [Citace: 19. 12 2017.] https://dzone.com/articles/bye-bye-javafx-scene-builder.

21. **STEGEMAN, John.** Unit Testing Your Application with JUnit. *Oracle.* [Online] 2 2010. [Citace: 29. 12 2017.] http://www.oracle.com/technetwork/articles/adf/part5-083468.html.

22. **CHACON, Scott.** *Pro Git.* Praha : CZ.NIC, 2009. ISBN 978-80-904248-1-4.

23. **DOČEKAL, Michal.** Správa linuxového serveru: Git na vlastním serveru. *LinuxEXPRES.* [Online] 8. 11 2011. [Citace: 25. 11 2017.] https://www.linuxexpres.cz/praxe/sprava-linuxoveho-serveru-git-na-vlastnim-serveru.

24. **HORDEJČUK, Vojtěch.** Maven. [Online] [Citace: 30. 11 2017.] http://voho.eu/wiki/maven/.

25. **The Apache Software Foundation.** Introduction to the POM. *Apache Maven Project.* [Online] 29. 11 2017. [Citace: 2. 12 2017.] https://maven.apache.org/guides/introduction/introduction-to-the-pom.html.

26. —. Available Plugins. *Apache Maven Project.* [Online] 29. 11 2017. [Citace: 4. 12 2017.] https://maven.apache.org/plugins/index.html.

27. —. Introduction to the Build Lifecycle. *Apache Maven Project.* [Online] 28. 12 2017. [Citace: 28. 12 2017.] https://maven.apache.org/guides/introduction/introduction-to-the-lifecycle.html.

28. **JetBrains s.r.o.** Features – IntelliJ IDEA. *JetBrains.* [Online] [Citace: 29. 1 2018.] https://www.jetbrains.com/idea/features/.

29. —. For students: Free Professional Developer Tools by JetBrains. *JetBrains.* [Online] [Citace: 29. 1 2018.] https://www.jetbrains.com/student/.

30. **TOUSSAINT, Godfried.** [Online] 2005. [Citace: 21. 2 2018.] http://cgm.cs.mcgill.ca/~godfried/publications/banff.pdf.

31. **BOONE, Kevin.** Generating simple MIDI files using Java, without using the Java Sound API. *kevinboone.net.* [Online] 8. 2 2013. [Citace: 23. 2 2018.] http://kevinboone.net/javamidi.html.

8 Přílohy

Příloha A – CD s aplikací a zdrojovými kódy