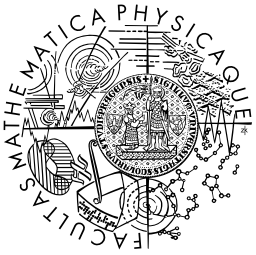


MATEMATICKO-FYZIKÁLNÍ FAKULTA
Univerzita Karlova

BAKALÁŘSKÁ PRÁCE

Matouš Helikar

Generování melodií pomocí genetického algoritmu

Katedra softwarového inženýrství

Vedoucí bakalářské práce:	Mgr. Ladislav Maršík
Studijní program:	Informatika
Studijní obor:	Programování a softwarové systémy

Praha 2016

Prohlašuji, že jsem tuto bakalářskou práci vypracoval(a) samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona v platném znění, zejména skutečnost, že Univerzita Karlova má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle §60 odst. 1 autorského zákona.

V ..... dne ..... Podpis autora

Název práce: Generování melodií pomocí genetického algoritmu

Autor: Matouš Helikar

Katedra: Katedra softwarového inženýrství

Vedoucí bakalářské práce: Mgr. Ladislav Maršík, Katedra softwarového inženýrství

Abstrakt: Pro skládání hudby je, podobně jako u jakékoli jiné tvůrčí činnosti, důležitá prvotní inspirace. Jednou z možností je stavět na melodiích vytvořených počítačem. Tato práce se věnuje generování skladeb pomocí stromové reprezentace jejich struktury a zásuvných modulů, které tvoří nebo upravují jednotlivé hudební motivy. Vytvořené stromy pak lze vzájemně kombinovat pomocí křížícího algoritmu založeném na hodnocení od uživatele. Skladby se tak vyvíjí na různých úrovních, jako použité nástroje či hudební motivy, rytmus a celková struktura. Vhodným nastavením parametrů generátoru a dílčích modulů pak lze docílit rozmanitých skladeb, například pro tvůrčí podnět nebo relaxaci. Součástí práce je naprogramovaná aplikace využívající tento postup ke generování hudby a uživatelská studie spokojenosti s výslednými skladbami.

Klíčová slova: genetický algoritmus, evoluční algoritmus, syntéza hudby, generování melodie

Title: Melody generation using a genetic algorithm

Author: Matouš Helikar

Department: Department of Software Engineering

Supervisor: Mgr. Ladislav Maršík, Department of Software Engineering

Abstract: Music composition, as all other creative activities, requires original inspiration, which can also come from melodies generated by a computer. This thesis describes generation of music tracks represented by tree structures with pluggable modules that create or alter individual musical motives. The trees can subsequently be combined by a crossbreeding algorithm driven by user ratings. This results in music tracks evolving on multiple levels, such as the selected instruments or musical motives, rhythm and overall structure. Appropriate settings of parameters for the generator and constituent modules can then produce varied tracks for inspiration or relaxation. The thesis is accompanied by a complete application using these techniques for music generation and a user study of satisfaction with the resulting tracks.

Keywords: genetic algorithm, evolutionary algorithm, music synthesis, melody generator

Chtěl bych poděkovat Mgr. Ladislavu Maršíkovi za vedení mé bakalářské práce, cenné rady a ochotu, kterou mi po celou dobu věnoval.

Obsah

Úvod

Tvůrčí činnost lidstva obvykle vychází z dosavadních výsledků, úspěchů a neúspěchů, těch celosvětově proslulých nebo specifických pro malé skupiny lidí. Nicméně, z něčeho se téměř vždy vychází. V dnešní době není příliš obvyklé, či snad ani možné, přijít s něčím zcela novým a originálním, s něčím vytvořeným bez jakékoli inspirace, ať už jde o literaturu, film, obrazy, jídlo, budovy nebo, pochopitelně, hudbu.

Zároveň však někdy samotná lidská představivost a tvůrčí schopnost nemusí stačit. Mnohdy je pro lidskou tvorbu přirozené převzít materiál, kterým se inspirujeme, přímo jako vzor, jenž stačí jen lehce poupravit a následně jej předvést světu jako vlastní tvorbu. A v hudebním průmyslu to platí obzvlášť.

Kde nestačí člověk, může ovšem přijít na řadu počítač. Můžeme algoritmicky vygenerovat hudební motivy — základní stavební prvky hudby tvořené několika tóny a pauzami — nebo celé skladby. Důležitou motivací pro tuto práci je pak využití takové hudby buď přímo pro určitý účel — například jako podkresovou hudbu pro práci, relaxaci nebo hry — či jako inspiraci pro následnou vlastní odvozenou tvorbu. Významným využitím počítačem tvořené hudby v praxi, a tím pádem i motivací k jejímu vývoji, je například získání inspirace pro komponování filmové hudby, aniž by bylo nutné řešit rizika a náklady spjaté s celou hudební produkcí a autorskými právy, jenž využívání jiné původní tvorby doprovází.

K počítačové tvorbě hudby lze přistupovat různě, a způsob vybraný v této práci je založený na reprezentaci skladby pomocí stromové struktury, z nichž se generuje hudba ve formátu MIDI (The MIDI Association). V listech a větvích tohoto stromu se nachází instance dodávané zásuvnými moduly, které mají na zodpovědnosti samotnou tvorbu jednotlivých hudebních motivů, zatímco vyšší vrstvy stromu vytváří kontext pro použité melodie a strukturu celé skladby. Tento kontext obsahuje mj. informace o pořadí motivů ve skladbě, různých melodiích hrajících současně, používaném tempu nebo hudebních nástrojích.

Program využívá celkem dva typy modulů. Jeden typ přímo vytváří nové melodie, jenž tvoří základní hudební motivy skladby. Druhý typ, tzv. „filtrovací moduly“, určitým způsobem upravuje jiné motivy, kupříkladu změnou výšky tónů,

jejich vynecháváním nebo prodlužováním. Přidáváním těchto modulů nad konkrétní hudební motiv můžeme získat melodie odvozené, podobné, ale nikoliv jejich přímé kopie. Toto odvozování je ovšem nezávislé na konkrétní melodii, kterou upravují, tudíž je lze využívat v různém kontextu k dosažení obdobného efektu.

Vygenerované stromy si pak uživatel může přehrát, hodnotit a uložit. Jakmile ohodnotí všechny skladby, program umožní spustit fázi křížení. Během této operace se na základě obdržených hodnocení vybírají jednotlivé stromy a nechají se „zmutovat“ (tj. provedou se na něm určité úpravy, jako změna tempa, odstranění některých motivů či přidání jiných) nebo vzájemně zkombinovat s ostatními stromy.

Struktura skladby v paměti je založena na hierarchickém popisu způsobu, jak vytvořit základní motivy a jak je přetvořit do finální skladby. Křížení dvou skladeb tento strom využívá k tomu, aby se vhodně a různorodě nakombinovaly dílčí vlastnosti v nich zakódované. Díky tomu může například nastat situace, kdy se z jedné skladby přebere velká část kostry její struktury, kdežto z té druhé se přeberou použité hudební nástroje, tempo a nejčastější hudební motivy, jenž náležitě nahradí zahozené motivy z první skladby.

Celý postup generování a křížení skladeb spolu s některými z dodaných zásuvných modulů jsou řízeny mnohými parametry, pomocí nichž lze výrazně ovlivnit různé vlastnosti výsledných skladeb, jako rozmanitost používaných hudebních motivů, počty použitých hudebních nástrojů, míra mutace individuálních skladeb během fáze křížení apod.

Kapitola 1 se věnuje analýze problematiky generování hudby, rozvádí řešení použité v této práci a porovnává ho s podobnými pracemi. V kapitole 2 se podrobněji popisují důležité informace o standardu MIDI, který je zde využíván. Kapitola 3 popisuje uživatelskou práci s programem, zatímco kapitola 4 se věnuje implementaci programu z hlediska programování. Kapitola 5 se nakonec zabývá zpětnou vazbou od uživatelů programu.

1. Analýza

Na začátku práce nejprve projdeme překážky, které se u generování hudby mohou vyskytnout a navrhneme několik možných řešení, jež by se dala využít. Navržená řešení podrobíme analýze výhod a nevýhod, následně je podrobněji popsán vybraný způsob implementace dodané aplikace založený na modulární stromové reprezentaci skladby a jak se využívá evoluční algoritmus založený na uživatelském hodnocení vygenerovaných skladeb pro získávání nových generací. V poslední sekci kapitoly je pak prozkoumáno několik jiných prací týkajících se generování hudby.

1.1 Generování hudby

Hudba je komplexní forma umění, která provází naši civilizaci již po tisíce let, během nichž se výrazně vyvíjela a postupně proměňovala. Vnímání hudby je z principu zcela subjektivní zážitek, který se dá jen velice těžko uchopit objektivně, vysvětlit logicky, předvídat s jistotou. Samozřejmě však nejde o naprosto nahodilý jev, a tak existují snahy pochopit lidský pohled na věc, co se nám na hudbě líbí či nikoliv, co od ní očekáváme a co nám v ní vadí, jakým způsobem by se dala vylepšit. Hudbou se zabývá velké množství teorie, terminologie, kategorií a disciplín, dohromady tvořící celek vědy zvané *muzikologie*.

Ovšem přesto všechno neexistuje jednotná odpověď na otázku, jak zní dobrá, kvalitní nebo populární hudba. Můžeme si všimnout jistých společných rysů, opakujících se vlastností: rytmu, melodií, akordů, hudebních nástrojů, struktury, zpěvu nebo textu písně. Nicméně výsledné reakce výrazně závisí na posluchači, na jeho vkusu, náladě či rozpoložení, ale i na době, kultuře, ostatní hudbě, která nás obklopuje a dalších okolnostech. Hudba se nám může líbit okamžitě, až po nějaké době, nebo čistě protože ji slyšíme často a někdo jiný se rozhodl, že bude celosvětově populární. Navíc mnohé reakce mohou působit protichůdně: co se jednomu líbí, jinému se může příčit. Ani jinak celkem běžná, rozumně popsatelná hudební vlastnost, jakou je pravidelný rytmus, neplatí všeobecně, jelikož některé hudební žánry tento koncept záměrně porušují.

Jakmile se rozhodneme hudbu generovat počítačem, nečeká nás lehký úkol již proto, že není nijak jednoduché definovat cíl nebo měřit dosažené výsledky. Ať bude samotné generování fungovat jakkoli, snaha bude spočívat v tom vytvořit melodii nebo skladbu s určitými žádoucími vlastnostmi, hudbu, která by se někomu mohla líbit. Nejen, že tyto vlastnosti — představu, jakou hudbu chceme získat — je velmi těžké popsat pomocí měřitelných parametrů, ale jakákoli změna naší představy by v takovém případě způsobila nutnou změnu všech hodnot parametrů, k nimž se chceme přiblížit. Pokud je naše představa například zpočátku pouze mlhavá a postupně se proměňuje nebo zpřesňuje, či pokud chce aplikaci využít více lidí s rozdílnými představami o hudbě, kterou chtějí vytvořit, samotné porovnávání měření nám stačit nebude a budeme se muset spolehnout na zpětnou vazbu přímo od uživatelů.

1.2 Možná řešení

1.2.1 Struktura skladby

V první řadě se musíme rozhodnout o míře strukturovanosti skladby, aneb do jaké míry bude vznik melodie na místě improvizovaný a vycházející jen z bezprostředního okolí, nebo do jaké míry bude skladba vycházet ze základních, společných informací a původního předpřipraveného plánu.

Vysoká míra improvizace a lokality se hodí například na tvorbu klasickou nebo jazzovou, díla dlouhá, zahrnující mnoho různých nápadů a celků, které se volně proměňují. Jazzový hudebník ani nemusí předem mít žádný plán pro svoji skladbu, místo toho ji postupně vyvíjí, určitou dobu se věnuje nějakému hudebnímu nápadu, než ho opustí a obvykle se k němu již nevrací. Tento myšlenkový proces je možné napodobit programem s podobným konceptem improvizace bez centrálního řízení s celkovou pamětí. Toho lze docílit kupříkladu namapováním not do několikarozměrného prostoru podle jejich výšky, hlasitosti, délky či barvy tónů, jímž procházíme částečně náhodným pohybem závislým jen na aktuální pozici, rychlosti a směru (Blackwell, 2007).

Naproti tomu mnohé jiné žánry závisí na celkové myšlence a plánu. Skladba může být rozdělena na jednotlivé části: úvod a závěr, sloky, refrény, mosty. Me-

lodie začne díky této struktuře být do větší či menší míry pravidelná, až předvídatelná. Na celou skladbu se pak dá dívat seshora: začneme od úplného celku a pomocí jistého plánu ji postupně dělíme a dotváříme. Dílo může využívat společné hudební motivy, nejen v rámci odpovídajících části jako mezi instancemi refrénu, které mohou být totožné nebo jen podobné, ale i mezi jinými částmi skladby či současně mezi různými hudebními nástroji.

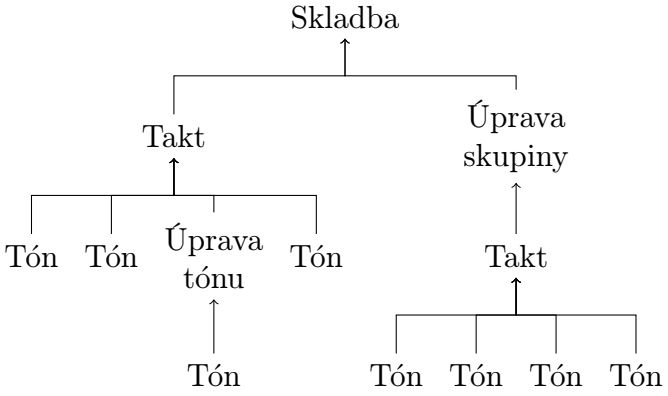
1.2.2 Reprezentace v paměti

Pro strukturované skladby se nabízí zejména dvě techniky řešení a jejich reprezentace v paměti. Za prvé by se dala využít *formální generativní gramatika* (McCormack, 1996). Skladba by byla reprezentována přechodovými pravidly, která by nejprve ze základního symbolu vytvořila obecnou strukturu a tu by postupně dotvářela do detailů finální podoby. Vygenerované slovo se poté interpretuje jako seznam instrukcí pro vytvoření hudební skladby. Při zvolení odlišných semínek pseudonáhodného číselného generátoru by navíc jedna konkrétní gramatika mohla vytvářet velké množství různých výsledků.

Alternativní reprezentací by mohla být *stromová struktura* (Hirata, Tojo a Hamanaka, 2014). Ta by udržovala informace o dané skladbě v určité hierarchii a dovolovala by s celkovou strukturou zacházet pomocí dalších operací, např. redukcí. Obdobně jako u gramatik by se výsledný strom chápal jako posloupnost instrukcí pro vytvoření výsledné skladby, ovšem tentokrát s možností využití hierarchie podstromů pro lokální změny v této interpretaci. Oproti metodě s gramatikami tím na jednu stranu můžeme ztratit jistou stručnost, nepředvídatelnost, flexibilitu a snadné zpracovávání, které gramatika přináší. Na tu druhou nám však strom umožní komplexnější práci nad generovanou skladbou a potenciálně i přehlednější reprezentaci výsledku.

V uzlech stromu mohou být uložena různorodá data, a to za jediného předpokladu, že z nich nakonec půjde vygenerovat zvuková stopa. Listy mohou obsahovat jednotlivé tóny v různých časových okamžicích a vyšší vrstvy stromu je budou jednoduše shlukovat do dvojic, taktů apod. Ve složitějších případech se mohou ve stromě nacházet uzly s jinou funkcí, které např. změní hlasitost nebo hudební nástroj všech svých potomků. Obecně vzato lze na každý uzel stromu nahlížet

jako na samostatný modul, který realizuje určitou operaci, např. vygenerování tónu. Své případné potomky uzel využívá jako „černou skříňku“, jejíž výsledky dále zpracovává bez konkrétní znalosti vnitřní funkce. Naopak svůj produkt pouze předá rodičovskému uzlu pro další úpravy.



Obrázek 1.1: Příklad možné stromové struktury reprezentující skladbu v paměti. Čas ve skladbě plyne zleva doprava. Šipky znázorňují proud dat, v kořenovém uzlu vznikne celkový výsledek.

Na rozdíl od generování skladeb pomocí gramatiky však stromová struktura automaticky nevede na metodu jejího vzniku, pouze definuje model skladby a převod z něj do hudebního díla. Proto je nutné samostatným algoritmem tento strom nejprve vytvořit.

1.2.3 Vývoj stromů

Ať už budeme skladbu v paměti reprezentovat jakkoliv a tuto strukturu získáme jakýmkoliv postupem, skládání hudby bývá obvykle proces iterativní. Ne každý nápad, který člověk či počítač vymyslí a vyzkouší, bude stejně úspěšný a povede ke shodnému cíli, tudíž výsledné skladby se hodí dále zpracovat, vylepšit a vyvíjet. První vytvořená skladba nemusí být finální, ale měli bychom z ní být schopni se poučit a příště získat lepší výsledek.

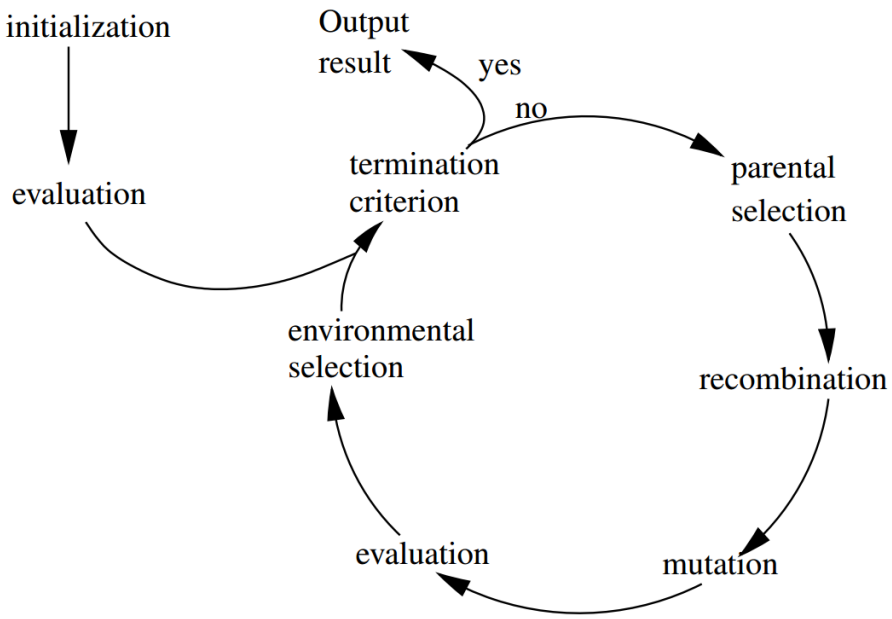
V první řadě potřebujeme umět posoudit kvalitu vygenerované skladby. Kvalita sama o sobě výrazně závisí na našem plánu, co chceme vytvořit. Pokud tuto představu, jak má znít dobrý výsledek, máme popsanou pomocí různých měřitelných parametrů, můžeme naši skladbu automaticky zanalyzovat a vypočítat jejich míru podobnosti. V případě, že skladbu posuzuje člověk, musí si skladbu poslechnout a ohodnotit podle své představy.

Jakmile takto vygenerujeme a ohodnotíme několik různých skladeb, můžeme vybrat úspěšnější jedince. Pokud z těchto dat budeme schopni nalézt nějaký vzor, např. podobné hodnoty některých počátečních parametrů, můžeme je náležitě přizpůsobit a generovat nové skladby, které se tak mohou přiblížit našemu cíli.

Případně můžeme vycházet přímo ze skladeb, jenž jsme získali, a nadále je přímo upravovat pomocí *evolučního algoritmu* (Weicker, 2003). Základní myšlenka evolučních algoritmů spočívá v opakovaném pokusu odstranit nežádoucí vlastnosti doposud dosaženého výsledku a zvýraznit nebo doplnit rysy žádoucí. Po vygenerování první generace výsledků se všechny určitým způsobem ohodnotí a vyberou ty nejúspěšnější pro následné zpracování. Během této fáze se provádí dvě hlavní operace: *mutace*, která na jednotlivých datech provádí náhodné samovolné změny a která experimentuje s možnými novými vlastnostmi, jenž do té doby žádný prvek generace neměl; a *křížení*, které vybere dva jedince a zkombinuje jejich data s cílem využít současně výhody obou. Noví jedinci se opět ohodnotí a případně zařadí do celkové datové sady. Ty méně úspěšné výsledky se naopak vyřadí a celý proces se opakuje dokud nemají výsledky adekvátní vlastnosti. Evoluční algoritmy se uplatňují v rozmanitém rozsahu odvětví pro hledání řešení problémů, pro něž není známý žádný (uspokojivě rychlý) postup pro hledání optimálního stavu a u nichž není nezbytně nutné nalézt absolutně nejlepší možné řešení, ale stačí takové, které dostatečně dobře splňuje určité vlastnosti.

Evoluční algoritmy využívají k hodnocení kvality jedinců tzv. *fitness funkci* (Weicker, 2003). *Fitness* jedince je obvykle jedno číslo, které shrnuje míru žádoucích vlastností vůči těm nežádoucím, případně podobu výsledku s ideálním stavem. Toto číslo se pak dá mezi jednotlivými jedinci snadno porovnávat a vybírat podle něj nejúspěšnější výsledky. V některých případech může být volba fitness funkce jednoznačná, kupříkladu u měřitelných fyzikálních veličin jako obsah plochy nebo vysílaný výkon.

Výběr správné fitness funkce v uměleckém oboru je však velmi obtížný (McCormack, 2005). Samozřejmě je možné měřit a vyčíslit velké množství různých vlastností jednotlivých zvuků nebo celých melodií, ale tato čísla samotná ještě přímo nevedou na hodnocení skladeb a výběr jedinců pro další zpracování. I tento vztah je možné studovat a odhadnout, ale za všech možných okolností bude vý-



Obrázek 1.2: Schéma iterativního procesu generování a hodnocení jedinců evolučním algoritmem (Weicker, 2003)

sledná fitness funkce přímo závislá na očekáváních a subjektivním vnímání příslušných osob. U hudby neexistují jednoznačné, obecně platné odpovědi na otázky kvality. Mohli bychom fitness funkci a celý výběr optimalizovat na základě určitých očekávání podle, řekněme, jazzové tvorby známých hudebníků, ale tím se zákonitě vzdálíme od veškeré jiné možné tvorby.

Alternativou k fitness funkci, která by se snažila samostatně skladby hodnotit, může však být také člověk. V takovém případě se celý postup výrazně zpomalí, neboť všechny skladby z každé generace si musí uživatel nejprve přehrát a ohodnotit, a pak teprve může program vytvořit generaci novou. Na druhou stranu se tím subjektivní, estetické vjemy přenechají člověku, kterému na nich záleží a má svoji představu o kýženém výsledku a o otázce kvality hudby.

Po výběru vhodných jedinců z aktuální generace se na nich staví další generování. Proces mutace a křížení jedinců však výrazně závisí na konkrétní implementaci a reprezentaci skladby, které se musí plně přizpůsobit.

1.3 Zvolená metoda

1.3.1 Datová struktura

Program popisovaný v této práci využívá pro reprezentaci skladby stromovou strukturu (viz sekce 1.2.2), kterou doprovází vedlejší datová struktura: *knihovna hudebních motivů*.

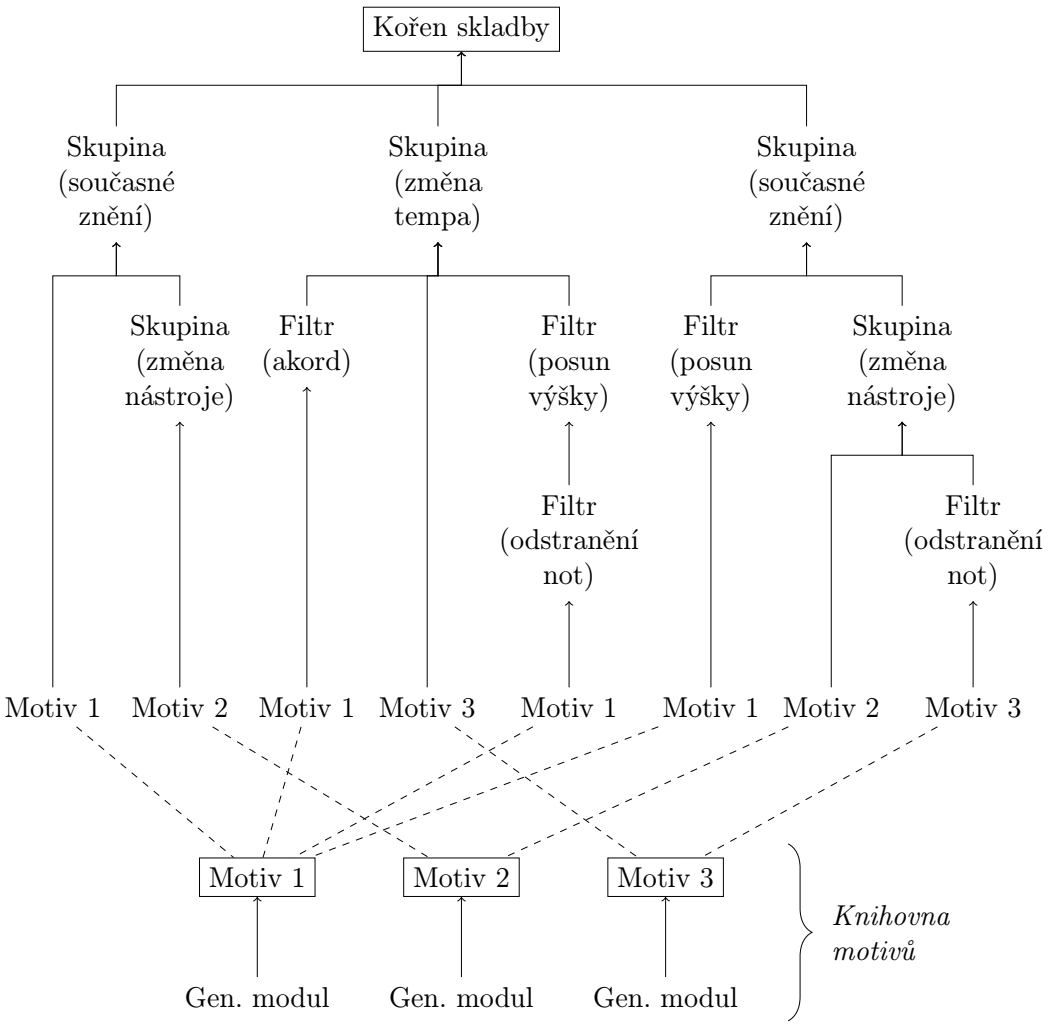
Před vytvořením stromu se nejprve vygeneruje určité množství individuálních hudebních motivů, na nichž bude strom postaven. Konkrétní motivy v dané skladbě se vybírají z této knihovny a jeden motiv se v ní může vyskytovat na několika místech. Velikost knihovny motivů ovlivní míru opakovaní nebo podobnosti částí skladby, od rozmanité sbírky použitých melodií až po jediný motiv, jenž je opakovaně používán v několika podobách. Obecně vzato knihovna limituje chaos a nesouzvučnost, jenž vznikají, jakmile se v každé části skladby nechá nezávisle generovat nová melodie.

Strom (viz obrázek 1.3) definuje strukturu skladby. Vnitřní, *skupinové uzly* mohou obsahovat několik potomků, jimž dodávají společné vlastnosti, např. společný hudební nástroj nebo tempo pro veškerou hudbu, která v potomcích vzniká. Hudba z potomků těchto uzlů navíc může být přehrávána postupně nebo současně.

Samotné generování hudby však zajišťují *zásuvné moduly*. Jejich úkol je vytvářet základní hudební motivy nebo upravovat melodie z jiných modulů. Svůj výsledek následně pošlou ve stromě o úroveň výš, kde se dále zpracovává. Melodie se postupně upravují jak na úrovni výšky, délky a hlasitosti tónů, tak na úrovni hudebních nástrojů, tempa a času, kdy se přehrají. Nakonec se celá skladba poskládá v kořeni stromu.

Generující moduly samy od sebe vytváří určitou melodii. Ta může teoreticky být libovolně komplikovaná, od pouhé rostoucí sekvence tónů, po úplně jiný generátor kratšího celku stavící výrazně na hudební teorii. Instance generujících modulů tvoří jednotlivé hudební motivy v jejich knihovně, z níž mohou být vybrány a použity ve skladbě vícekrát.

Hudební motivy však nemusí být vždy napříč celou skladbou používány naprosto totožně. K tomu přichází druhý typ zásuvných modulů, které se ve stromě



Obrázek 1.3: Příklad reprezentace skladby v paměti pomocí použité stromové struktury s moduly a knihovnou motivů.
Čas ve skladbě plyne zleva doprava. Data proudí po směru šipek zespoda nahoru: od knihovních motivů v listech tvořených generujícími moduly, přes modifikace a uspořádání filtrujícími moduly a skupinovými uzly, po kořenový uzel, ve kterém vznikne celkový výsledek.

používají. *Filtrovací moduly* berou jako vstup melodii vytvořenou podstromem a určitým způsobem ji přetvoří, např. odstraní některé tóny, jiné prodlouží, změní výšku tónů apod. Tyto filtry mohou být volně přidávány do stromu nad motivy z příslušné knihovny, což umožní celé aplikaci spolehlivě ve skladbě generovat odvozené, ovšem nikoli shodné, melodie.

Jak bylo již dříve řečeno, generování hudby je silně řízeno konkrétními pravidly a mírou aplikace teorie, a změna těchto pravidel v případě změny kýženého výsledku je netriviální. Zásuvné moduly, které se do programu přidávají z externích souborů, umožňují využívat jednotný postup generování struktury a následné evoluce skladeb. Ovšem to, jakým konkrétním způsobem se samotná melodie skutečně tvoří lze jednoduše nahradit.

1.3.2 Evoluce

Stromy skladeb se nevygenerují rovnou finální, ale podrobí se vývoji pomocí evolučního algoritmu. V sekci 1.2.3 jsme popsali komplikace, které s sebou nese automatická fitness funkce. Z toho důvodu program popisovaný v této práci žádnou takovou funkci neimplementuje. Místo toho nechává veškeré hodnocení vygenerovaných skladeb na uživateli, který se může samovolně rozhodnout se zaměřit na různé vlastnosti — subjektivní vjemy jsou tak klasifikovány subjektivním vnímáním.

Vedlejším efektem takového přístupu je však zároveň nemožnost programu automaticky vyřazovat neúspěšné kombinace a mutace skladeb před jejich zařazením do nové generace. Všechny nové stromy se tudíž v dalším kroku aplikace vyskytnou.

Po ohodnocení a výběru úspěšnějších jedinců probíhá mutace a křížení. Mutace způsobí malé, náhodné, samovolné změny ve stromě, např. odstranění některých uzlů, přidání jiných nebo změna jejich parametrů. Křížení současně prochází dva stromy a náhodně používá prvky z jednoho či druhého pro vytvoření nové skladby.

1.4 Podobné práce

Pojem *syntéza hudby* nevede na jediný problém s jasným cílem — jde o celou řadu otázek, kde se volně prolínají obory vědecké s těmi uměleckými a kde mnohdy neexistuje žádný objektivně správný výsledek. Jednotlivé práce se často zaměřují na určitou konkrétní oblast zájmu zblízka, ovšem s tím, že se nejedná o celkové řešení dané problematiky.

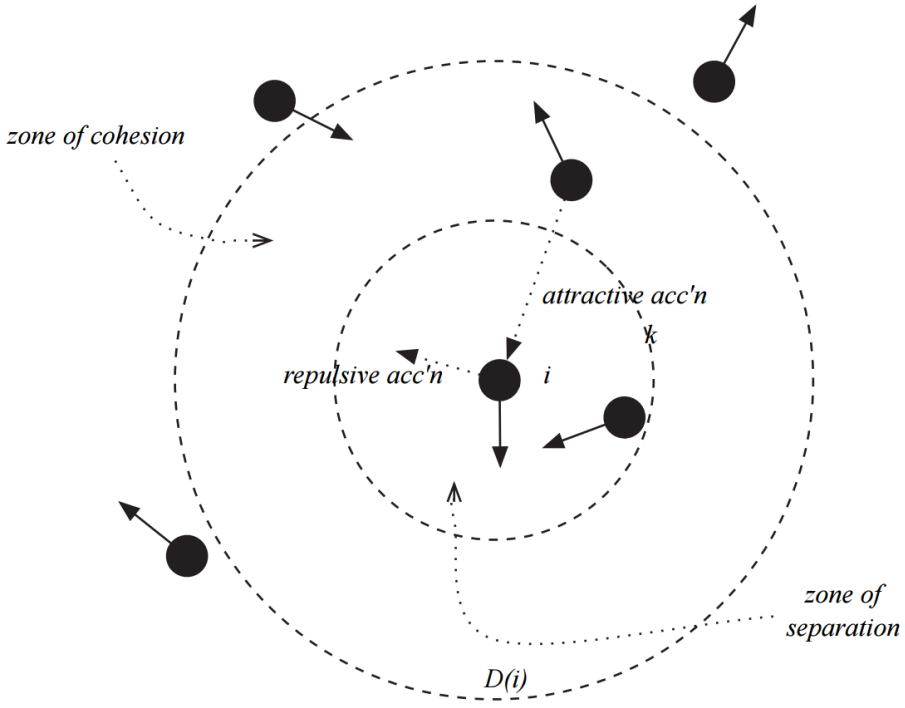
Počítačová tvorba hudby začíná u jednotlivých zvuků, pokračuje generováním skupin tónů, až nakonec může zahrnout celou strukturu skladby, v níž se autoři snaží více či méně přesně pochopit a napodobit tvůrčí myšlení reálných hudebníků. Společné rysy mnoha prací spočívají v nutnosti na požádání generovat různé výsledky, které ovšem vždy splňují žádané vlastnosti. Náhoda vynucená nedeterminismem je tak často doprovázena evolučním přístupem, který získaný výstup dále zpracovává a upravuje.

1.4.1 Syntéza zvuků a hudby

McDermott, Griffith a O'Neill (2007) se zabývají syntézou zvuků, které by napodobovaly vlastnosti a znění zvuků získaných kupříkladu z různých nahrávek. Samotný tón nebo jiný zvuk, jenž chceme syntetizovat, se dá popsat klidně více než 200 atributy, ovšem doprovodné experimenty se omezovaly na 40 z nich, které dostatečně přesně popisují aspekty zvuku týkající se jeho zabarvení, lidského vnímání a těch čistě statistických. Výrazným problémem při napodobování cílového zvuku pak tkví v nelinearitě chování výsledku v závislosti na změnách těchto parametrů, tj. při určitých hodnotách všech ostatních parametrů nemusí změna jednoho z nich způsobit žádný rozlišitelný rozdíl, nebo se naopak může malou změnou výsledný tón proměnit neobvykle výrazně. Výsledné hledání na základě jednoduchého gradientního algoritmu, který by se pouze postupně snažil přibližovat hodnoty svých parametrů, tak velmi snadno selže. Místo toho se zde tím pádem uplatní genetický algoritmus řízený fitness funkcí popisující vzdálenost od cílových parametrů v jedné z několika možných metrik.

Volné improvizaci inspirované jazzovou hudbou se pak věnuje Blackwell (2007) pomocí metod založených na tzv. swarms („rojích“). V prostoru jsou rozpro-

střena „zrnka“ reprezentující velmi krátké hudební ukázky. Tímto prostorem pak dynamicky prolétá virtuální roj částic a postupně spouští zvuky, na něž v prostoru naráží. Roj je následně přitahován určitými zvuky dodanými do prostoru jako atraktory. Jakmile takovýto atraktor roj objeví, začne létat poblíž, čímž dynamicky vytváří melodii podobnou dané zvukové ukázce, ale nikoli totožnou. Podobně jako v jazzové hudbě není důležitá celková skladba a konvenční pojetí rytmu, melodie a harmonie, ale spíše její plynutí, nepřipravovaná, postupně se rodící a krátkodobá struktura. Tohoto výsledku je dosaženo interakcí roje pouze se svým blízkým okolím na základě principů odloučení, napodobování, soudružnosti, zvídavosti a společenské komunikace, vypozorovaných z hejn, rojů a společenství zvířat v přírodě.



Obrázek 1.4: Zóny v okolí částice roje, v nichž se částice vzájemně přitahují nebo odpuzují (Blackwell, 2007)

Ramirez a kol. (2007) také vychází z jazzové tvorby, ale místo volné improvizace na základě blízkého okolí bez žádné celkové struktury se snaží symbolicky popsat myšlenkový pochod, znalosti a projev jazzového hudebníka. Nad takovýmto symbolickým popisem následně spouští evoluční algoritmus, z něhož získávají výpočetní model pro samotné generování nové hudby. Nízkoúrovňový po-

pis nahrávky vychází z výpočtu okamžité energie, základní frekvence a rozdělení nahrávky na jednotlivé noty po aplikaci diskrétní Fourierovy transformace. Vyšší úroveň pravidel v modelu dané nahrávky se pak získává analýzou sousedících skupin not a odhadem, jakým způsobem se obvykle navazuje na rozpracované hudební motivy. Po natrénování programu různými jazzovými nahrávkami se z těchto dat zkonstruuje pomocí sekvenčně pokrývajícího algoritmu sada pravidel, která je popisují co nejpřesněji. Tato pravidla jsou následně podrobena mutaci a křížícím operacím, a nakonec z nich je zrekonstruována skladba nová.

Různorodých přístupů ke generování hudby je ovšem velké množství. Jedna možnost vychází z celulárních automatů (Burraston a kol., 2004). Tato metoda využívá určité typy celulárních automatů jako Conway's Game of Life (Gardner, 1970) nebo Demon Cyclic Space (Dewdney, 1989). Na začátku běhu je matice automatu nastavena náhodně, dále se vyvíjí podle pravidel vybraného automatu. Jednotlivé živé buňky automatu odpovídají trojici not popsanou svými souřadnicemi, které se mohou lišit ve své délce, výšce a časování. Program jednotlivé buňky projde a živé spustí v daném pořadí po jednotlivých sloupcích seshora dolů.

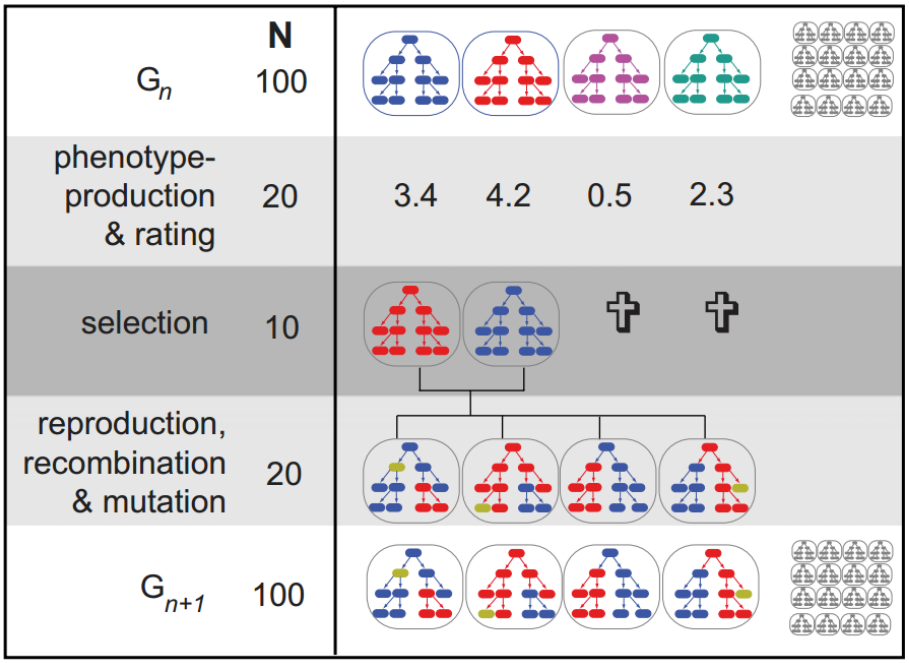
Program zvaný GenDash (Waschka II, 1999) se zaměřuje na tvorbu orchestrální hudby nevyužívající elektronické nástroje. Autor v něm přichází s nápadem založit výběr vhodných kandidátů pro další generace na lidském „mentorovi“ a náhodě místo fitness funkce, která bývá výpočetně náročná, neefektivní a těžko se adaptuje na změny v hudebním stylu. Pro přechod do nové generace se využívá jediný křížící bod, a některé aspekty skladeb (nebo celé skladby) se mohou stát „recesivními vlastnostmi“, které se dočasně skryjí a zapomenou, ovšem v budoucích generacích se mohou opět objevit.

1.4.2 DarwinTunes

Na uživatelském hodnocení generovaných melodií staví také projekt DarwinTunes (MacCallum a kol., 2012). Melodie jsou reprezentovány stromy s digitálními genomy, z nichž každý kóduje informace o pozici not, instrumentaci a parametrech chování. Některé parametry, jako tempo či rytmus, však zůstavají shodné pro všechny současně vyvíjené skladby. Celý strom pak lze projít a aplikací pra-

videl z každého genomu vytvořit krátká hudební smyčka. Tyto stromy si umí náhodně proházet své genomy nebo vytvořit nový, náhodný materiál.

Vytvořené hudební smyčky pak byly umístěny na internetu[1], kde široká veřejnost po určitou dobu mohla hodnotit estetičnost jednotlivých skladeb pro výběr vhodných kandidátů do nových generací. V každé generaci se hodnotila pouze pětina všech skladeb, z nichž se polovina odstranila a zbylé stromy vytvořily generaci novou, tudíž generace obsahovaly různě staré skladby.



Obrázek 1.5: Zpracování jedné generace skladeb v projektu DarwinTunes. (MacCallum a kol., 2012)
Uživatelé hodnotí v každé generaci pětinu všech dostupných skladeb. Hůře hodnocená polovina z nich zanikne, zatímco lepší polovina si náhodně prohodí své genomy, přidají nějaký nový genetický materiál a výsledek zařadí zpět do celkové populace.

1.4.3 Shrnutí

Většina těchto děl se zaměřuje buď na generování velice krátkých úseků o jednotkách až desítkách not, nebo na napodobování jazzové improvizace, u níž je spíše než celková stavba skladby důležitá hudební návaznost v blízkém okolí.

Tato práce se primárně inspiruje posledními dvěma zmíněnými pracemi — GenDash a DarwinTunes — které místo jednorázové tvorby fitness funkce pro vý-

[1] `http://darwintunes.org/`

běr ideálních kandidátů do dalších generací využívají pro umělecké zaměření výsledku vstup a hodnocení od člověka.

Místo všeobecně náhodného výstupu ovšem přidává do generovaných melodií určitou strukturu spolu s filtrující moduly, které blíže popisují způsob, jakým se od sebe různé hudební motivy odvozují. Výsledek poté umožňuje např. opakování stejných či podobných melodií napříč skladbou, která tak působí méně náhodně. V neposlední řadě se klade důraz na modularitu programu, díky níž je relativně snadné volně zaměňovat a kombinovat typy a metodiky generování motivů.

2. Technický základ

Tato práce a doprovázející aplikace staví na generování hudby dle standardu MIDI. Vzhledem k tomu je užitečné jeho fungování alespoň obecně pochopit jak pro uživatele, který si bude v programu hudbu generovat, tak pro programátora, který zkoumá, jakým způsobem v nejnižší vrstvě generování probíhá. Tato kapitola nejprve popisuje historii standardu MIDI, jak jí byl jeho vývoj ovlivněn a jak je následně celý systém uspořádán. Poté se kapitola věnuje virtualizaci tohoto standardu v rámci softwaru a jeho aplikaci v knihovně programovacího jazyka Java.

2.1 MIDI

Podobně jako u grafických formátů digitálních souborů, jenž je možné rozdělit na rastrové a vektorové, se dají na dvě hlavní skupiny v závislosti na metodě vzniku dotyčných dat dělit i soubory hudební. Zatímco analogový zvukový signál se po nahrání při převodu do digitální formy obvykle nejprve diskretizuje za pomoci metod vzorkování a kvantování, zvuk generovaný v počítači bez takovéhoto reálného zdroje může být přirozené ukládat pomocí instrukcí a mechanismů, jenž k němu vedou.

Nicméně mechanismus popisu, jakým způsobem se dá vytvořit a zreprodukovat hudební dílo, není nijak přímočarý. Pro tyto účely vznikl rozsáhlý technický standard zvaný *Musical Instrument Digital Interface*, aneb *MIDI* (The MIDI Association).

2.1.1 Zprávy

V počátcích MIDI vznikalo jako hardwarové i softwarové rozhraní mezi elektronickými hudebními nástroji a dalšími podpůrnými přístroji, před tím, než se po celém světě rozšířily obecně programovatelné počítače. Fyzické přístroje, jako například syntetizéry nebo sekvencery, se propojovaly pomocí drátů, které přenášely data daná příslušným protokolem.

Tato data jsou rozdělena do jednotlivých MIDI *zpráv*, skládající se ze *statu-*

sového bytu, jenž popisuje typ zprávy, a *datových bytů*, jejichž význam se mění podle kontextu a konkrétní potřeby. Jednotlivé zprávy pak mohou kódovat různé příkazy, například začínající či končící tón o dané výšce a hlasitosti nebo změnu vybraného hudebního nástroje. Mnohé zprávy také ve spodních 4 bitech svého statusového bytu kódují číslo jednoho ze 16 *kanálů*, kterého se zpráva týká. Jednotlivé kanály umožňují paralelní proud různých dat, kupříkladu tóny rozdílných hudebních nástrojů, přičemž přístroje mohou zpracovávat zprávy z libovolného výběru kanálů.

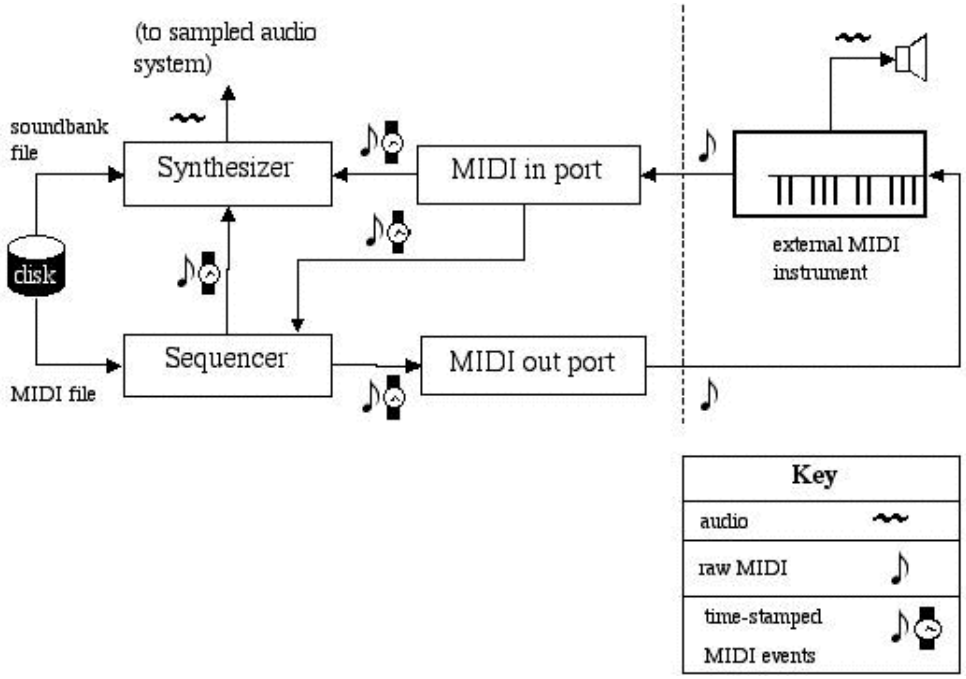
Základní předpoklad pro správné fungování tohoto protokolu pak spočívá v tom, že data se *streamují*, aneb že zprávy se v náležitých zařízeních interpretují rovnou v době jejich obdržení. Z toho důvodu je nutné veškeré načasování zpráv zařídit před jejich odesláním a očekává se, že do cílového zařízení dorazí včas. Toho lze dosáhnout kupříkladu tak, že hudebník hraje naživo, zpráva pro každou notu se odešle v době stisku dotyčné klávesy a její zvuk se syntetizuje a spustí okamžitě, jakmile dotyčný signál doputuje propojovacím drátem do příslušného zařízení.

2.1.2 Sekvenční data

Alternativním přístupem ke streamovaným MIDI zprávám jsou data *sekvenční*. Zprávy se v nich obalují do *událostí* obsahující informaci o čase, ve kterém má dojít k jejich interpretaci. Navíc se do seznamu platných dat protokolu přidávají i tzv. *meta zprávy*, které umožňují vkládat mezi ostatní data i informace pro syntézu jednotlivých zvuků jinak nedůležité, jako nastavení tempa nebo vedlejší textové informace typu název skladby či text písně. Veškerá takováto data pak je možné uložit do souboru a kdykoli opět přehrát se správným časováním.

Standardní MIDI soubor zahrnuje jednu *sekvenci*, která obsahuje alespoň jednu *stopu*. V rámci stopy se ukládají jednotlivé události s informací o čase a vlastní zprávou, napříč 16 kanály. Významnou část souboru pak zaplňují události pro jednotlivé noty, přičemž různé hudební nástroje se mohou ukládat buď v rámci různých kanálů, nebo v různých stopách. Sekvenci pak čte hardwarový či softwarový *sekvencer*, ve správném čase se v něm generují jednotlivé zprávy a ty se následně odesílají do výstupního zařízení nebo portu k finálnímu zpraco-

vání. Oproti běžným vzorkovaným hudebním formátům jsou MIDI soubory obvykle výrazně menší a umožňují větší flexibilitu pro následné zásahy do skladby, ovšem zároveň jsou omezeny na zvuky, které je možné syntetizovat v počítači a pro jejich přehrávání je nutné mít správně nastavený *syntetizér* s příslušnými hudebními nástroji.



Obrázek 2.1: Diagram možného zapojení MIDI zařízení a toku dat (Oracle)

2.1.3 Hudební nástroje

Vedle obecného komunikačního protokolu MIDI existuje i standard *General MIDI*, který určuje dodatečnou specifikaci pro syntetizéry. Kromě určitých požadavků na schopnosti syntetizérů zpracovávat všechny kanály nebo několik not hrajících současně a popisu některých dalších zpráv definuje i standardní sadu 128 hudebních nástrojů a 47 bicích nástrojů rezervovaných pro 10. kanál. Konkrétní zvuk nástrojů stále závisí na použitém syntetizéru, ovšem podle specifikace General MIDI by měly odpovídat definovaným vlastnostem.

Hudební nástroje jsou ve standardu MIDI ukládány v hierarchii *bank* a *programů*. Specifikace dovoluje až 128 různých očíslovaných bank, z nichž každá obsahuje až 128 opět očíslovaných programů. Každý program odpovídá jednomu konkrétnímu hudební nástroji a algoritmu jeho syntézy, tudíž v běžící aplikaci

se dá nástroj jednoznačně identifikovat pomocí této dvojice čísel. Přepínání nástroje na daném kanálu se obdobně dělí na samostatnou změnu jeho banky a programu.

2.1.4 Standard v současnosti

Přestože původně byl standard MIDI zamýšlený jako rozhraní mezi individuálními fyzickými přístroji, později mohla být jednotlivá zařízení nahrazena počítačovými programy, a to do takové míry, že tvorba, čtení nebo ukládání hudebních dat i jejich uspořádávání do sekvencí, syntéza a reprodukce se mohou nacházet v rámci jediného nástroje. Nicméně, přestože je celý proces virtualizován, používané terminologie, význam a tok dat zůstávají analogické k původnímu způsobu fungování.

Standard MIDI tak nedefinuje pouze datový formát, ve kterém se mohou ukládat hudební soubory ve formě instrukcí pro jejich reprodukci, ale popisují celý způsob práce s takovýmito daty, zejména spolupráci mezi různými součástmi potřebnými pro tvorbu a reprodukci hudby. Konkrétní implementaci jednotlivých zařízení, ať už hardwarovou nebo softwarovou, nijak nedefinuje, pouze pro všechny dodává jednotný jazyk a metodiku, zatímco možnost ukládat výsledek do jednoznačně reprodukovatelných souborů je takřka vedlejším produktem.

2.2 Java Sound API

MIDI je velmi rozšířený hudební standard, a tak existuje pro práci s ním velké množství knihoven pro různé programovací jazyky. V této práci jsme jako vývojovou platformu vybrali jazyk *Java*. Nejen, že je Java multiplatformní, tzn. její programy mohou bez samostatného překladu do různých souborů běžet na mnoha operačních systémech nebo procesorech, a pro tuto aplikaci není zcela zásadní vysoká rychlost běhu, ale navíc má Java velkou podporu pro hudební aplikace. Součástí základních knihoven Javy je i rozhraní *Java Sound API*, jehož část je právě věnovaná standardu MIDI.

Toto rozhraní je sice čistě softwarové, nicméně svojí organizací napodobuje původní fyzickou strukturu zapojení jednotlivých zařízení, pro něž byl standard

MIDI vyvíjen. Hierarchie dat v sekvencích, stopách, událostech a zprávách se statusovými a datovými byty je zrcadlena hierarchií dostupných tříd. Virtuální zařízení, která umí přijímat nebo vysílat MIDI data, jsou reprezentována objekty implementujícími rozhraní `MidiDevice`. Významné postavení pak má třída `Sequencer`, jež zachytává a ve správném čase vysílá jednotlivé události nebo zpracovává celé sekvence, a `Synthesizer`, jenž vytváří z těchto dat samotný audio výstup. Java Sound API pak také rovnou obsahuje 128 základních hudebních nástrojů ze specifikace General MIDI.

3. Uživatelská dokumentace

Tato kapitola popisuje, jakým způsobem lze popisovaný program spustit a následně ovládat. Podrobněji se ukazuje jak vytvořit, nahrát nebo uložit nový projekt, a poté jak přehrávat, prozkoumávat nebo hodnotit vygenerované skladby, jak vytvořit novou generaci skladeb a jak uložit do souboru vybranou skladbu.

3.1 Instalace

Program pro svůj běh v první řadě vyžaduje nainstalovanou platformu Java SE (standardní edice) verze 8 či vyšší, podporované operačními systémy Windows, Mac OS, Linux a Solaris. Instrukce pro zjištění aktuální nainstalované verze Javy a odkazy na její stažení jsou k dispozici na oficiálních internetových stránkách[1].

Jakmile je správná verze platformy Java nainstalována, program reprezentovaný souborem *MidiGen.jar* se spustí standardní metodou pro daný operační systém, například dvojklikem.

Pro správný běh aplikace je ovšem nutné dodat i soubory obsahující jednotlivé moduly. Program tyto soubory hledá v podadresáři *modules* umístěném ve stejné složce jako spouštěná aplikace. S touto prací jsou dodány soubory s moduly *modifier.jar* a *simpleGen.jar*, které umožňují ihned spustit generování. Funkční struktura souborů tedy může vypadat například takto:

Dokumenty/MidiGen/MidiGen.jar
: spouštěný soubor,

Dokumenty/MidiGen/modules/modifier.jar
: obsahuje filtrovací moduly pro úpravu existujících motivů,

Dokumenty/MidiGen/modules/simpleGen.jar
: obsahuje moduly pro generování základních motivů.

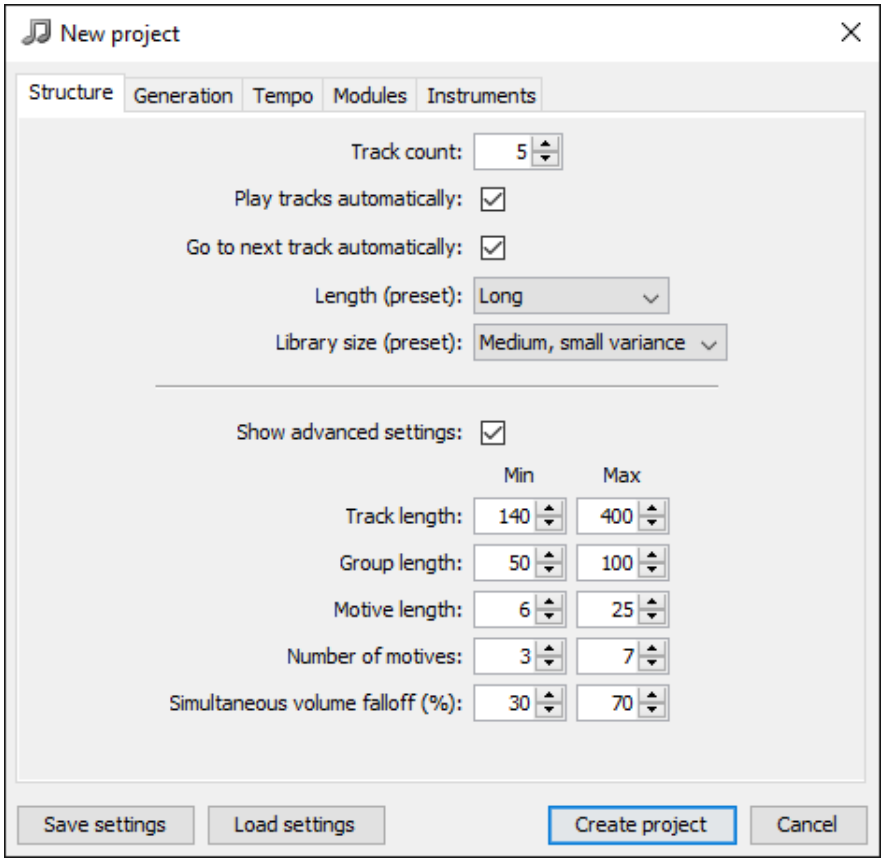
Uživatelské rozhraní programu je v anglickém jazyce, tudíž je snadno šiřitelný po celém světě.

[1] `http://www.oracle.com/technetwork/java/javase/downloads`

3.2 Vytvoření projektu

Konkrétní sbírka několika skladeb, jejich generace, moduly, které k jejich vygenerování byly použity a nastavení, kterým bylo generování ovládáno, jsou všechny zahrnuty do společného projektu. Než je možné s programem jakkoli pracovat, nejprve je nutné tento projekt založit nebo nahrát ze souboru.

Po spuštění programu se v jeho okně objeví dvě tlačítka. *New project* zobrazí dialog, ve kterém lze nastavit možnosti nového projektu a poté ho vytvořit. *Open project* umožní vybrat soubor z disku, ze kterého se načte již vytvořený projekt. Tyto možnosti jsou k dispozici i z menu na horní liště, odkud je také u již otevřeného projektu možné celý projekt uložit do souboru.



Obrázek 3.1: Záložka *Structure* dialogu pro vytváření nového projektu

Po otevření dialogu pro vytvoření nového projektu lze uložit stávající nastavení do souboru nebo je z něj naopak nahrát pomocí tlačítek *Save settings* a *Load settings* ve spodní části okna. Díky tomu je možné si například jednou podrobně nastavit používané hudební nástroje nebo bližší nastavení jednotlivých modulů a následně tato nastavení opakovaně používat pro nové projekty.

Dialog samotný se skládá z několika záložek. První tři obsahují především číselné parametry, které lze nastavit pomocí přednastavených hodnot („presetů“) z rozbalovacích seznamů. Zaškrtávací políčko *Show advanced settings* zobrazí nebo schová číselné parametry, pomocí nichž je možné hodnoty nastavit vlastní.

Záložka *Structure* (viz obrázek 3.1) obsahuje základní parametry struktury skladby a chování aplikace:

Track count počet skladeb v rámci jedné generace,

Play tracks automatically
zda má začít přehrávání skladby ihned po jejím přepnutí,

Go to next track automatically
zda se má přejít na další skladbu ihned po ohodnocení,

Track length rozsah délek celé skladby (v časových jednotkách MIDI — „ticks“),

Group length rozsah délek, které by přibližně měly mít jednotlivé skupinové uzly ve stromě,

Motive length rozsah délek, kterých by měly dosahovat generující moduly při tvorbě hudebních motivů,

Number of motives
rozsah počtu knihovních motivů, které bude skladba využívat,

Simultaneous volume falloff
procentuální pokles hlasitosti pro současně přehrávané melodie.

Záložka *Generation* dovoluje nastavovat parametry spjaté s generováním, mutací a křížením stromů:

Percentage of tracks undergoing cross-breeding / mutation only
procentuální podíl skladeb, které při přechodu do nové generace vzniknou křížením dvojic stromů, resp. pouhou mutací jednoho stromu,

Mutation chance
pravděpodobnost, že daný uzel bude podroben samovolné mutaci,

Mutation intensity
parametr určující velikost změn provedených mutací; vyšší číslo znamená větší změny,

Diversity of library motive popularity
: rozdílnost popularity jednotlivých hudebních motivů v knihovně; nízké číslo způsobí rovnoměrnější zastoupení všech motivů ve skladbě, u vyšších čísel pak budou některé motivy mnohem častěji vybírané než jiné
(pro pokročilé uživatele: jedná se o parametr λ exponenciálního rozdělení, dle něhož se vybírá váha pro výběr náhodného motivu),

Percentage of library motives kept in a new generation
: procentuální podíl knihovních motivů, které budou zachovány při křížení dvou stromů; čísla vyšší než 50 % povedou na exponenciální růst počtu motivů v jednotlivých generacích,

Chance of mirroring motives across instruments
: pravděpodobnost, že současně znějící melodie přehrávané více nástroji budou nutně sdílet stejný hudební motiv,

Chance of shifting simultaneous groups in time
: pravděpodobnost, že současně přehrávané melodie budou vůči sobě časově posunuty o několik „ticků“,

Chance of changing instruments in new generations
: pravděpodobnost, že při křížení dvou skladeb budou hudební nástroje hlavního z nich nahrazeny těmi z druhé skladby,

Chance of favoring new motives in place of the removed ones
: pravděpodobnost, že při nahrazování zahozených hudebních motivů během křížení dostanou přednost motivy vygenerované zcela nově.

Záložka *Tempo* obsahuje parametry rychlosti skladby. Tempo je udávané v jednotkách BPM (údery za minutu) a pro danou skladbu se vybírá z normálního rozdělení omezené minimální a maximální povolenou hodnotou:

Mean
: průměrná hodnota tempa,

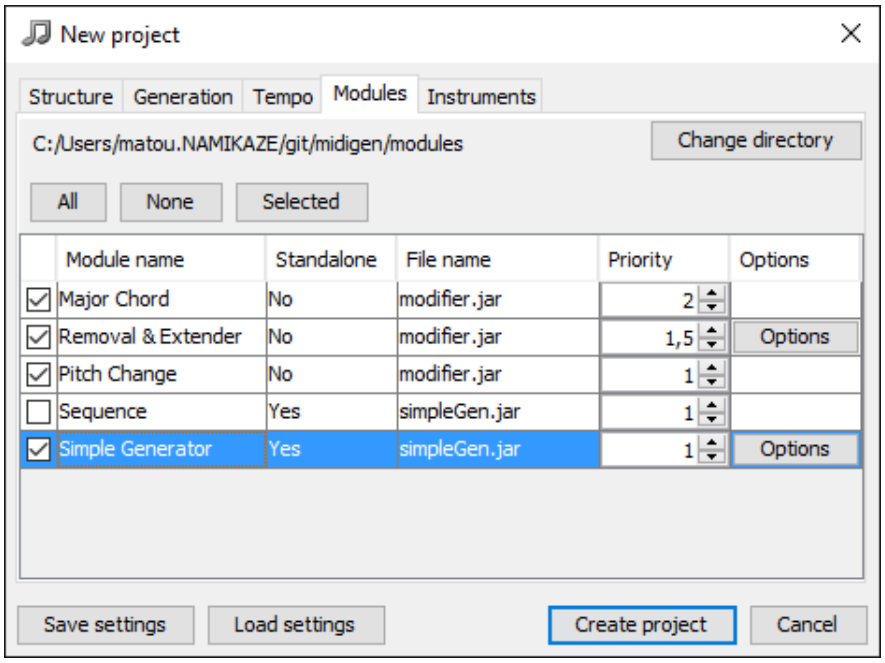
Standard deviation
: směrodatná odchylka tempa,

Limits
: minimální a maximální povolená hodnota tempa,

Range of change probability
: rozpětí pravděpodobnosti, že skupinový uzel stromu skladby změní tempo,

Range of change magnitude
: rozpětí absolutní hodnoty změny tempa ve skupinovém uzlu.

Záložka *Modules* (viz obrázek 3.2) zobrazí seznam všech nalezených modulů. První sloupec tabulky umožňuje výběr těch modulů, které chceme ve skladbách daného projektu využívat. Tlačítka *All* a *None* hromadně vyberou nebo odstraní výběr všech motivů naráz, zatímco *Selected* zaškrtne všechny vybrané řádky (výběr několika řádků najednou lze provést standardní metodou pro daný operační systém, např. ve Windows pomocí kliknutí na jeden řádek a držení klávesy Shift při výběru jiného). Sloupec *Priority* definuje relativní četnosti jednotlivých modulů vůči sobě, např. modul s prioritou 5 bude použit pětkrát častěji než modul s prioritou 1. Poslední sloupec obsahuje tlačítko pro podrobnější nastavení konkrétního modulu, pokud je k dispozici.



Obrázek 3.2: Záložka *Modules* dialogu pro vytváření nového projektu

Součástí dodané aplikace jsou následující moduly:

Sequence sekvenční generátor, který vytvoří jednoduchou rostoucí či klesající posloupnost tónů,

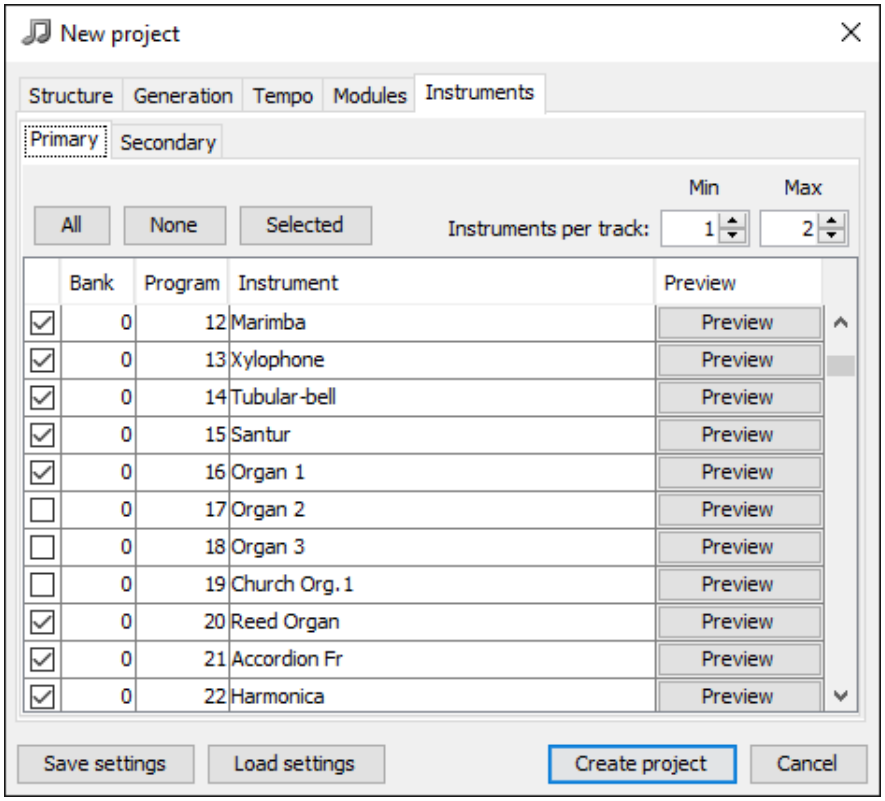
Simple Generator melodický generátor, který vytvoří posloupnost několika tónů, z nichž každý je odvozen od předchozího v rámci určitého maximálního rozpětí rozdílů výšek,

Pitch Change filtr, který změní výšky všech tónů jiného podstromu,

Removal & Extender
: filtr, který vymaže nějaké noty vygenerované jiným podstromem a případně příslušně prodlouží tóny předchozí,

Major Chord
: filtr, který veškeré noty převede na durový kvintakord s původním tónem jako základním (tj. ke každému tónu současně přidá i velkou tercii a čistou kvintu).

Poslední záložka — *Instruments* (viz obrázek 3.3) — obsahuje seznam všech hudebních nástrojů, které program nalezl. Nástroje jsou rozděleny na primární (podzáložka *Primary*) a sekundární (podzáložka *Secondary*); sekundární nástroje nemohou být použity pro hlavní melodii a musí být doprovázeny nějakým primárním nástrojem. Výběr nástrojů se provádí stejně jako volba modulů v předchozí záložce, poslední sloupec pak obsahuje tlačítko, které tento nástroj pro ukázku přehraje na jednoduché stupnici tónů.



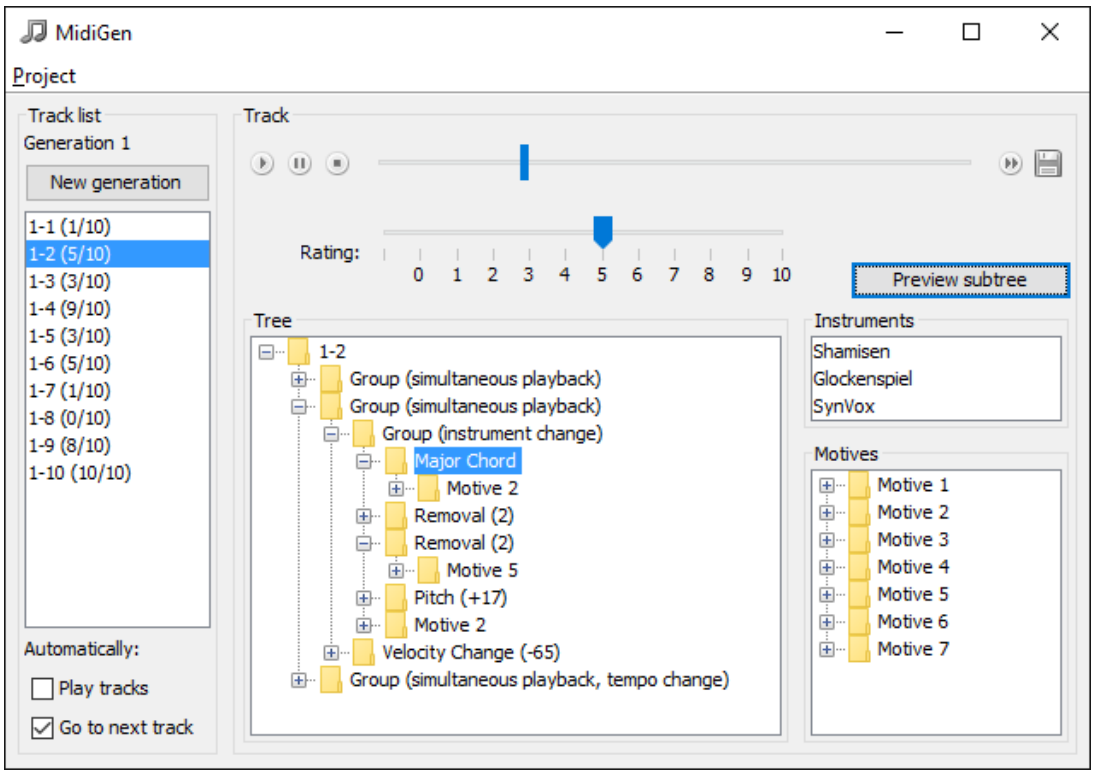
Obrázek 3.3: Záložka *Instruments* dialogu pro vytváření nového projektu

3.3 Ovládání projektu

Po vytvoření nového projektu stisknutím tlačítka *Create project* a potvrzení aktuálního nastavení se vygeneruje první generace skladeb a následně se zobrazí okno s jejich seznamem. Nahrání již existujícího projektu ze souboru zobrazí projekt v odpovídajícím stavu.

V levém sloupci jsou vypsány všechny skladby dané generace, mezi kterými lze volně přepínat, spolu s jejich dosavadním uživatelským hodnocením. Ve zbytku okna jsou zobrazeny ovládací prvky pro přehrávání, hodnocení nebo ukládání vybrané skladby, pod nimi pak bližší informace o jejím stromě, použitých hudebních nástrojích a motivech.

Před dokončením jedné generace a vytvořením nové se musí všechny stávající skladby ohodnotit — neohodnocené skladby jsou označeny v jejich seznamu pomocí *(?/10)*. Jakmile takovou skladbu vybereme kliknutím na její položku v seznamu, tlačítky v horní části okna ji můžeme přehrát, pauzovat nebo zastavit. Aktuálně přehrávaná pozice lze za běhu nebo u pozastavené skladby posouvat pomocí indikátoru průběhu vedle nich.



Obrázek 3.4: Okno spuštěného projektu

Skladbu následně ohodnotíme spodním posuvníkem pomocí čísla od 0 do 10. Toto číslo nastavuje relativní četnost výběru této skladby ve fázi vytváření nové generace — skladba ohodnocená 0 nebude vybrána nikdy, zatímco skladba s číslem 8 bude vybírána dvakrát častěji než ta ohodnocená číslem 4.

Pokud by nás zajímalo podrobněji, jakým způsobem tato skladba vznikla, pod hlavními ovládacími prvky jsou vyobrazeny strom, který skladbu reprezentuje v paměti, použité hudební nástroje a knihovna motivů (viz sekce 1.3.1). Po vybrání libovolného uzlu stromu můžeme přehrát pouze tuto část skladby pomocí tlačítka *Preview subtree* (pro vlastnosti, které tento podstrom nedefinuje, jako nástroje nebo tempo, se použije implicitní nastavení dané kořenem celé skladby). Obdobně můžeme přehrát i ukázku jednotlivých nástrojů (*Preview instrument*) nebo knihovních motivů, z nichž se dílo skládá (*Preview motive*).

Vybranou skladbu také můžeme uložit na disk do souboru MIDI (viz sekce 2.1) pomocí tlačítka s disketou napravo od indikátoru průběhu skladby. Jakmile se budeme chtít přesunout na hodnocení další skladby, buďto vybereme novou skladbu ze seznamu na levé straně okna, nebo klikneme na tlačítko vedle diskety *Next track*. Pokud je zaškrtnuta možnost *Play automatically*, při výběru nové skladby se okamžitě spustí.

Po ohodnocení všech skladeb dané generace můžeme tlačítkem *New generation* fázi ukončit, shrnout výsledky a vygenerovat podle nich novou generaci. Seznam skladeb se nahradí a přehrávání či hodnocení skladeb můžeme provést stejně jako předtím.

V kterékoli chvíli je možné celý projekt uložit do souboru pomocí položky v horním menu *Project* → *Save*. Po jeho opětovném nahrání získáme zpět použité nastavení projektu, aktuální generaci skladeb a již provedené hodnocení.

4. Programátorská dokumentace

V této kapitole se nejprve popisují použité technologie a platformy, na nichž je aplikace postavena, a poté instrukce k překladu a k sestavení programu z jeho zdrojového kódu. Následně je struktura kódu spolu se způsobem interakce jednotlivých jeho částí postupně popsána a znázorněna třídními diagramy.

4.1 Použité technologie

Aplikace je napsána v programovacím jazyce *Java SE 8* s použitím standardního API jazyka, tudíž je přenositelný a spustitelný na všech platformách, které tuto verzi Javy podporují: Windows, Mac OS, Linux a Solaris. Program nevyžaduje žádné externí knihovny.

Výrazně využívané balíčky standardní knihovny jsou `javax.swing` pro vykreslování uživatelského rozhraní a `javax.sound.midi` pro tvorbu, přehrávání a ukládání MIDI dat (viz sekce 2.2).

4.2 Sestavení programu

K překladu a sestavení programu je využit *Apache Ant*. Přiložené sestavovací skripty (build scripts) definují následující cíle (targets):

cleanFull odstraní všechny přeložené soubory programu,

build přeloží a sestaví jádro programu,

doc vygeneruje do adresáře *doc* automatickou programátorskou dokumentaci z dokumentačních komentářů,

jar seskupí přeložené soubory do spustitelného souboru *MidiGen.jar*,

plugins přeloží a sestaví přiložené moduly do podadresáře *modules*,

pluginAPI přeloží a sestaví do souboru *moduleAPI.jar* veškeré třídy a rozhraní nutné k vytváření nových modulů:

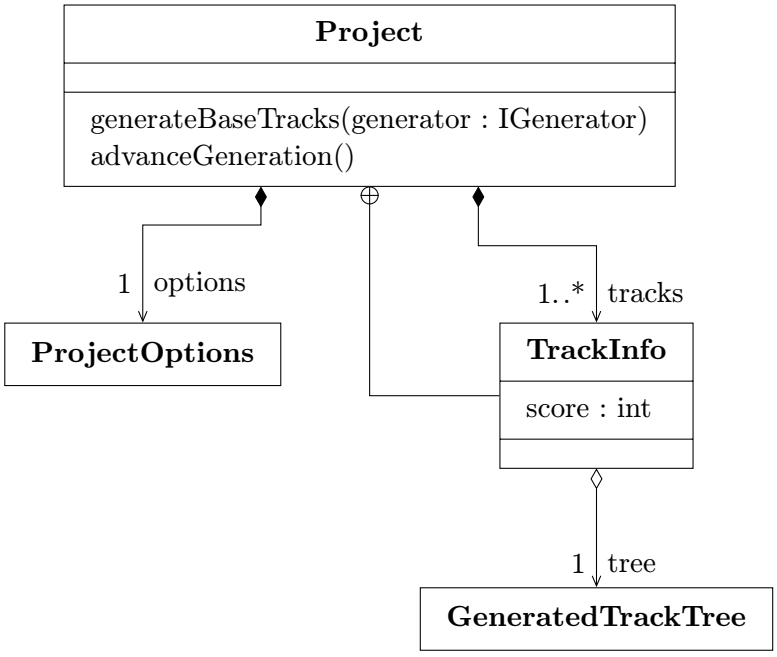
- celý balíček `helikarm.midigen.module.api`,
- celý balíček `helikarm.midigen.module.tracktree.note`,
- rozhraní `IGenerationModule`, `ITrackTreeNode` a `ITrackTreeGroupNode` z balíčku `helikarm.midigen.tracktree`.

4.3 Struktura programu

Vstupní branou do programu je třída `MidiGen`, jenž nejprve v příslušném podadresáři pomocí `ModuleHandler`u dohledá veškeré dodané zásuvné moduly a následně spustí uživatelské rozhraní. Vytvořený projekt je celý reprezentován třídou `Project`, která se pomocí serializace umí načíst ze souboru nebo se do něj uložit. Třídy `Generator` a `CrossBreeder` zajišťují obecnou práci se stromy skladeb při jeho generování a křížení.

4.3.1 Projekt

`Project` obsahuje jednak veškerá nastavení definovaná na začátku uživatelem, jednak seznam všech vygenerovaných stromů skladeb v aktuální generaci. Tato třída zajišťuje obecnou práci se skupinami skladeb, zejména jejich hromadné vytváření nebo spouštění postupu do nové generace. Viz obrázek 4.1.

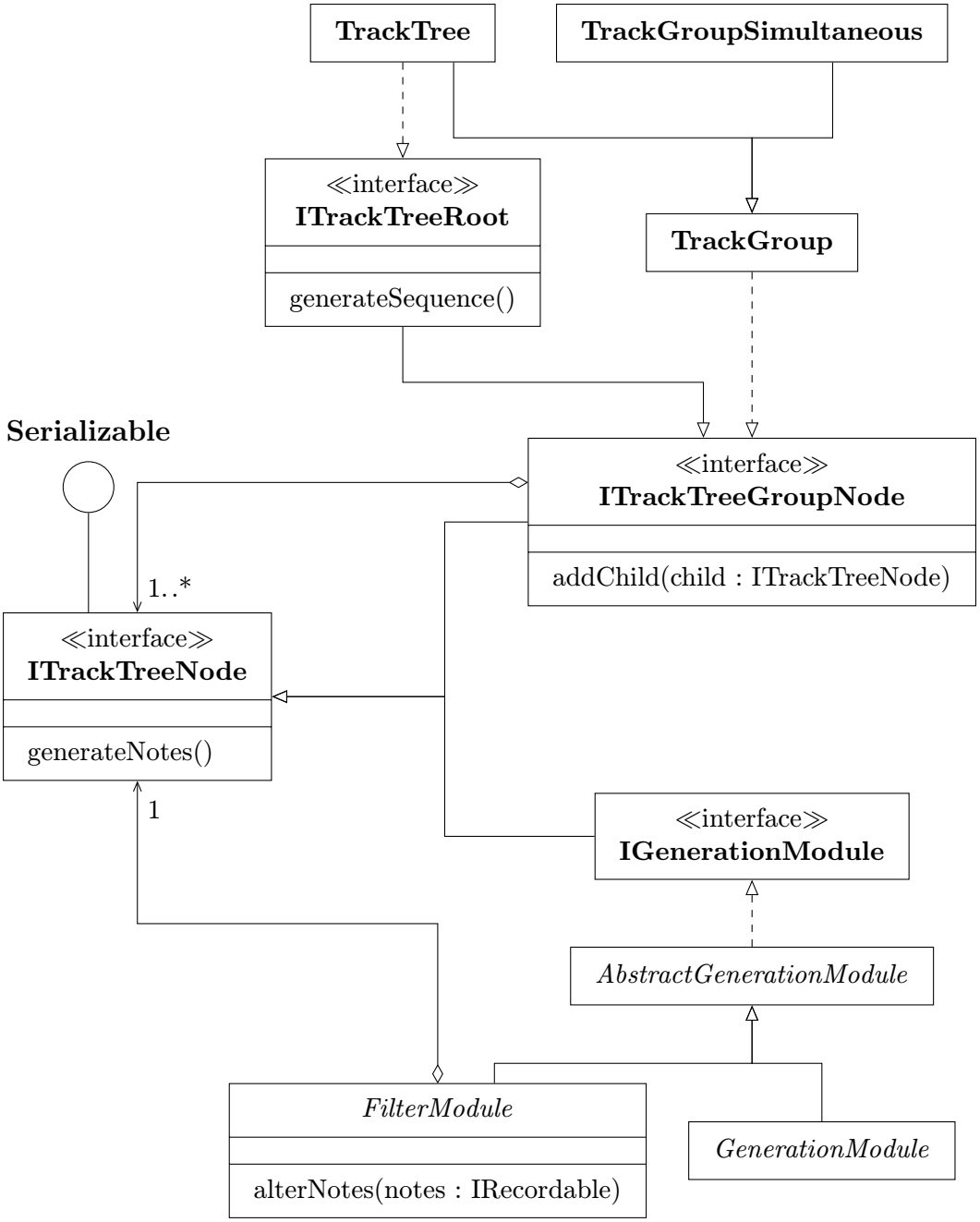


Obrázek 4.1: Třídní diagram projektu

4.3.2 Strom skladby

Skladbu v paměti představuje strom, jehož vnitřní vrcholy definují hierarchickou strukturu skladby, zatímco v jeho listech se nachází instance jednotlivých modulů (viz sekce 1.2.1).

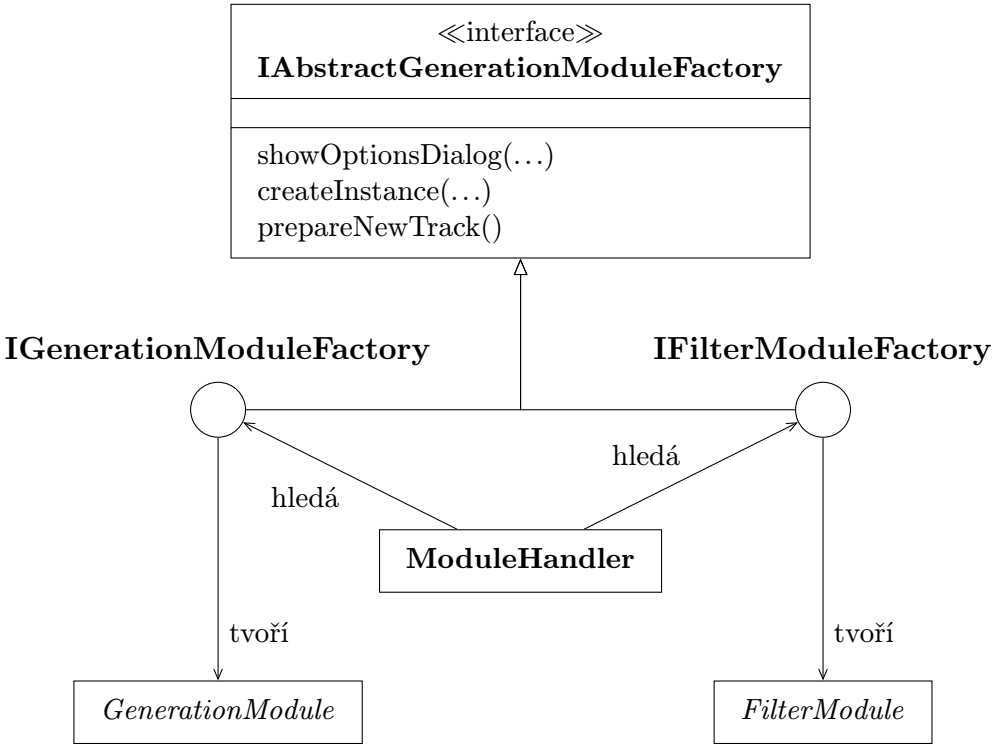
Základní struktura stromu následuje návrhový vzor *composite* (Gamma a kol., 1995). Veškeré uzly jsou implementací serializovatelného rozhraní `ITrackTreeNode`, od něhož dědí dvě hlavní větve hierarchie inheritance: skupinové uzly (`ITrackTreeGroupNode`), které vlastní další uzly stromu jako potomky, a moduly (`IGenerationModule`), který vytváří samotnou melodii. Zvláštním typem modulu je `FilterModule`, jenž obaluje jiný uzel stromu jako *decorator*. Viz obrázek 4.2.



Obrázek 4.2: Třídní diagram stromu hierarchicky reprezentujícího skladbu

4.3.3 Zásuvné moduly

Při inicializaci programu `ModuleHandler` projde všechny .jar soubory ve složce *modules* a v nich dohledá všechny platné třídy reprezentující zásuvné moduly. Aby byl zásuvný modul platný, musí obsahovat *factory* třídy (Gamma a kol., 1995), které budou na požádání dodávat instance příslušných modulů ve stromě. Rozhraní pro platné factory třídy jsou dvojí: `IGenerationModuleFactory` pro moduly přímo vytvářející melodie, a `IFilterModuleFactory` pro *decorator* moduly, které upravují melodie z jiného podstromu. Filtrovací moduly se nijak nestarají o získání tohoto podstromu, pouze nadefinují funkci `alterNotes`, které se v parametru dodá příslušná vygenerovaná melodie. Viz obrázek 4.3.

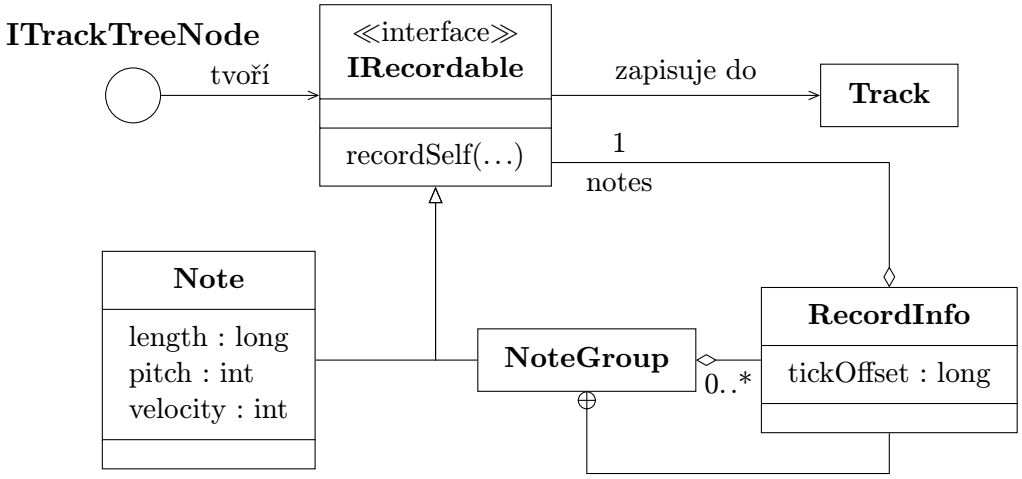


Obrázek 4.3: Třídní diagram zásuvných modulů a jejich uzlů stromu

4.3.4 Generování MIDI sekvence

Strom skladby představuje hierarchický způsob, jakým lze vytvořit hudební celek. Než se z něj však získá samotná MIDI sekvence, nejprve se musí převést na reprezentaci jednotlivých not. Noty tvoří jiný, jednodušší strom dle vzoru *composite* založený na rozhraní `IRecordable`. Objekty typu `Note` představují jedinou notu, která se umí nahrát do MIDI stopy (`Track`), zatímco `NoteGroup`

několik takových objektů obaluje do skupiny se společným kanálem, tempem a posunem v čase. Viz obrázek 4.4.



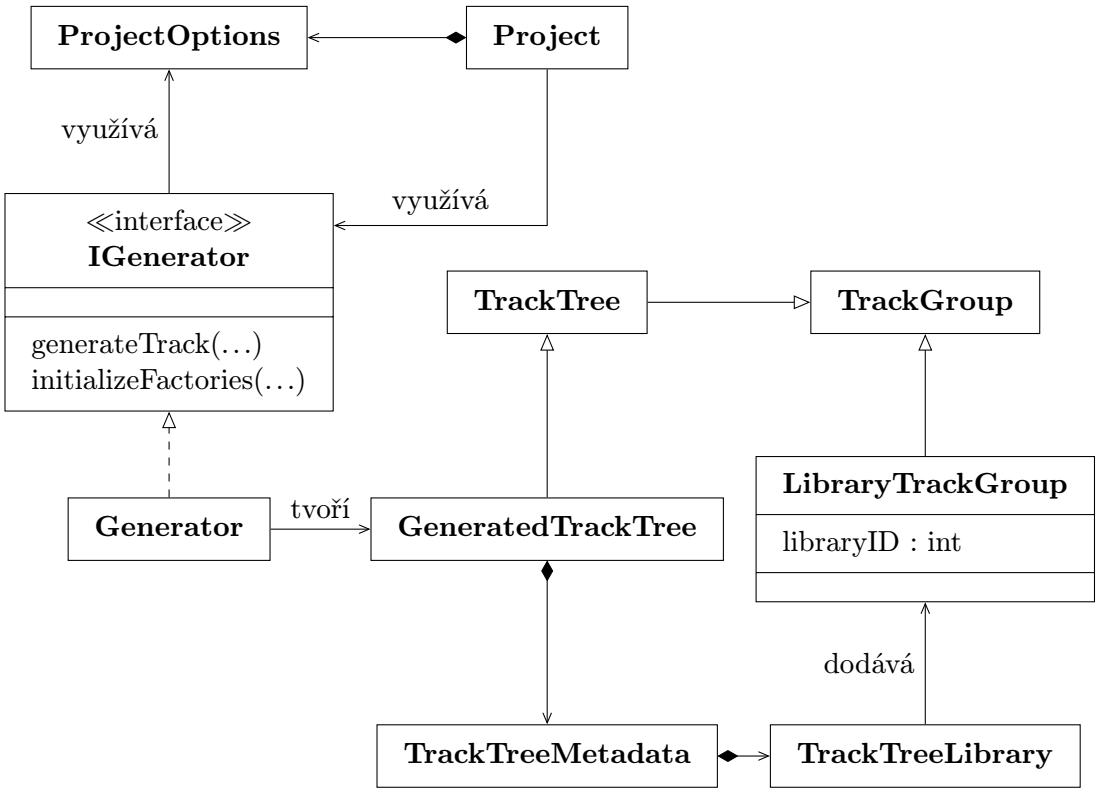
Obrázek 4.4: Třídní diagram generátoru MIDI sekvencí ze stromu skladby

4.3.5 Generování stromu

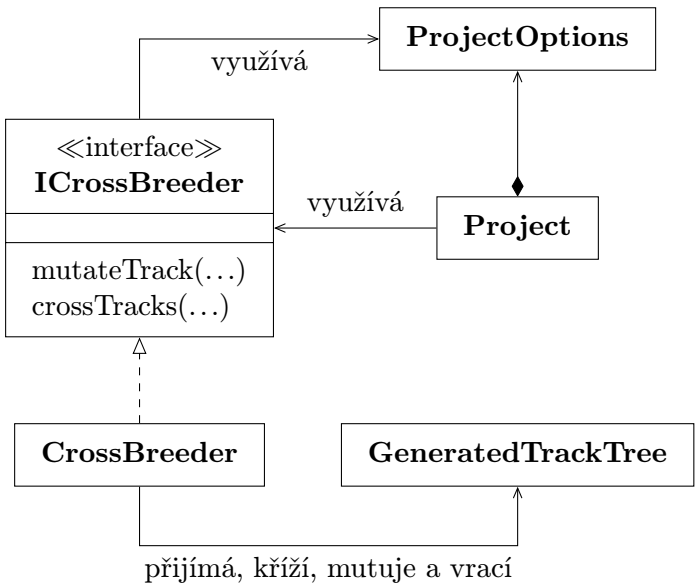
O vytváření celého stromu se stará třída `Generator`, která na požádání funkcí `generateTrack` podle daného nastavení projektu vytvoří strom skladby první generace. Tento `GeneratedTrackTree` obsahuje vedle samotné stromové struktury také metadata, v níž jsou uloženy zejména knihovna motivů a použité hudební nástroje. Knihovna `TrackTreeLibrary` udržuje seznam všech hudebních motivů spolu s pravděpodobnostmi jejich výběru a umožňuje vybírat motiv buďto náhodně nebo dle identifikačního čísla. Viz obrázek 4.5.

4.3.6 Křížení stromu

Křížení a mutaci stromů vyvolává třída `CrossBreeder`. Celý postup je rozdělen do několika podprogramů postupně volaných z metody `crossTracks` (více viz sekce 1.2.3). V `createRoot` se nejprve z obou vstupních stromů extrahují metadata včetně knihovny motivů a použitých hudebních nástrojů, patřičně se zkombinují a vyrobí se kořen nového stromu s těmito metadaty. Poté se v `crossNodes` již popsaným způsobem prochází současně oba stromy a podle náhodných rozhodnutí se do nového stromu postupně přidávají jednotlivé uzly. Nakonec se v `mutateTree` nový strom nechá samovolně zmutovat. Viz obrázek 4.6.



Obrázek 4.5: Třídní diagram generátoru stromu skladby



Obrázek 4.6: Třídní diagram křížení a mutace stromu skladby

4.4 Evoluční algoritmus

Jednou ze základních funkcí programu je evoluční algoritmus, který umožňuje stromy skladeb postupně mutovat a křížit (viz sekce 1.3.2). Než k těmto krokům dojde, uživatel si musí přehrát dosud vygenerované skladby a ohodnotit je. Hodnocení probíhá očíslováním jednotlivých skladeb mezi 0 a 10. Tato skóre se použijí jako váhy pravděpodobnosti jejich výběru do fáze vývoje, v níž se dvojice stromů nejprve zkříží do nového jedince a ten se následně zmutuje. Tento postup se opakuje, dokud nezískáme dostatečně velkou novou generaci skladeb, které se opět nechají uživateli přehrát a hodnotit.

Mutace stromu dovoluje následující úpravy:

- odstranění instance knihovního motivu nebo celého podstromu, který je obsahuje,
- přidání nového podstromu do skupinového uzlu, obdobně jako v prvotní fázi generování,
- mutace instance filtrujícího modulu tak, jak je v něm naimplementována.

Před samotným křížením dvou stromů se projdou a zkombinují vedlejší informace, jenž je doprovází, zejména použité tempo, hudební nástroje a knihovna motivů. Tempo se vybere podle jedné skladby a případně se lehce pozmění. Nástroje pro daný strom jsou namapovány na určité MIDI kanály (viz kapitola 2), a pro nový strom se provede náhodný výběr z obou původních. Knihovna motivů se kombinuje obdobně, ale podstupuje více kroků zajišťujících, že nově vznikající strom bude z těch původních obsahovat pouze některé, předem vybrané motivy (v rámci mutace se ovšem mohou objevit i motivy zcela nové):

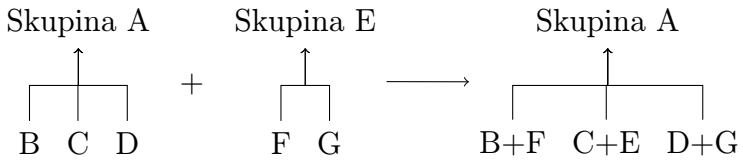
- Z obou původních knihoven se získá náhodný výběr motivů, které se budou používat i ve stromě novém. Instance modulů v nich obsažené se mohou nejprve nechat zmutovat svojí dodanou implementací.

- Do seznamu motivů používaných budoucím stromem se přidá několik nových motivů, obdobně jako v prvotní fázi generování.

- Motivy původních stromů, které nebyly vybrány do nové generace, se přemapují na nějaký motiv nové knihovny, přičemž větší šanci má přemapování na zcela nové motivy.

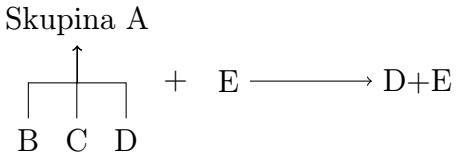
Nakonec se provede samotné křížení dvou stromů skladeb. Oba stromy se prochází současně do hloubky, ovšem jeden z nich má výstavní postavení řídícího stromu. V každém kroku křížení se drží jeden podstrom z každé skladby a na základě typu jejich kořenů se provede určitá operace, která vybere nebo vygeneruje odpovídající uzel v novém stromě a posune křížení dále.

- Pokud zkoumáme dvojici skupinových uzlů, jako na samém začátku algoritmu, projdeme postupně všechny potomky uzlu řídícího stromu, pro každý z nich vybereme odpovídajícího potomka z druhého uzlu (případně vybereme celý druhý podstrom) a přesuneme křížení na tuto dvojici. Výběr potomka z druhého stromu je řízen indexem z normálního rozdělení se střední hodnotou danou pořadím aktuálního uzlu prvního stromu v rámci všech svých sourozenců. Díky tomu se častěji navzájem kříží podstromy ze začátku skupiny apod., nicméně to není zaručené a může dojít ke křížení i na větší vzdálenost ve stromech.



Obrázek 4.7: Možný výsledek křížení dvou skupinových uzlů

- U jednoho skupinového uzlu jednoduše vybereme náhodně nějakého potomka a překřížíme ho s druhým uzlem.

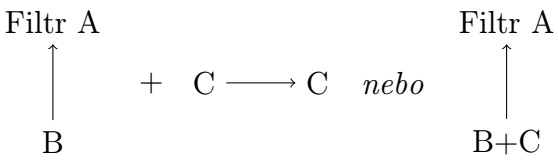


Obrázek 4.8: Možný výsledek křížení skupinového uzlu s neskupinovým

- Ve všech ostatních případech máme z principu na výběr ze dvou jednoduchých uzlů, z nichž použijeme právě jeden, vybraný z rovnoměrného rozdělení.

- Jakmile vybereme filtrovací modul, následující krok postaví protější nevybraný uzel proti potomkovi tohoto filtru. Díky tomu je možné ve výsledku

získat například melodii z jednoho stromu proměněnou filtrem z toho druhého.



Obrázek 4.9: Možné výsledky křížení filtrovacího modulu s jiným uzlem

- Žádný knihovní motiv nezkopírujeme přímo, ale místo něj do nového stromu vložíme motiv daný mapováním z předchozí fáze – některé motivy mohou být až na mutace stejné, jiné motivy se jednotně napříč celou skladbou nahrazují jiným. Díky tomu struktura skladby zůstává zachována, ovšem samotná melodie se změní.

Po dokončení fáze křížení tak získáme nový strom, který vzájemně kombinuje obecnou strukturu, hudební motivy, nástroje a rytmus původního výběru.

5. Testování aplikace

Po vytvoření aplikace jsme ji dali k dispozici pěti lidem s různou znalostí hudební teorie k otestování. Získaná zpětná vazba také pomohla doladit několik vlastností programu.

5.1 Ovládání programu

Aplikaci zkoušelo používat pět různých lidí, od těch, kteří se tvorbě hudby věnují výrazně, až po takové, kteří ji jen běžně poslouchají. Významná část zpětné vazby od testujících uživatelů byla věnována ovladatelnosti programu a grafickému rozhraní. Mnohé připomínky z této oblasti jsme nakonec zahrnuli do finální podoby aplikace.

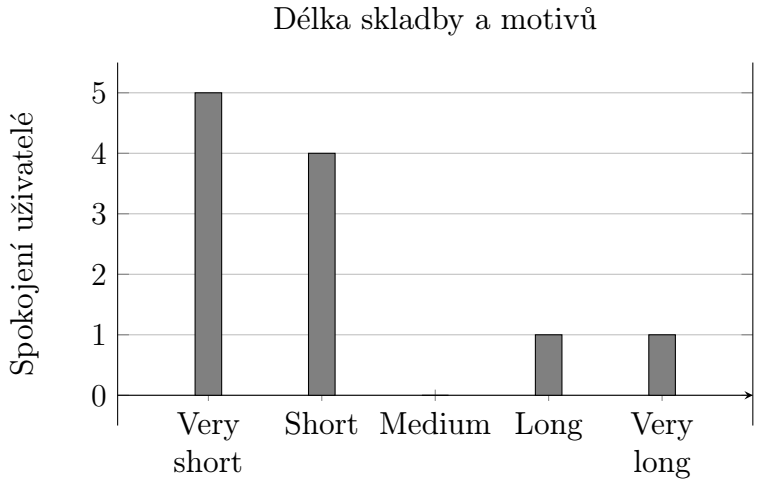
Mimo jiné se jednalo o přednastavené skupiny parametrů při vytváření nového projektu bez nutnosti nastavovat zvlášť všechny ručně nebo o možnost ukládat a načítat nastavení nového projektu z externího souboru pro snazší opakování a testování různých parametrů, vedle mnoha dalších menších vylepšení.

Nepochybně největší důsledek pro ovládání programu, který vyšel ze zpětné vazby od uživatelů, je vizualizace stromu a knihovny motivů jednotlivých skladeb. Díky němu je možné nejen procházet strukturu, jenž kombinuje a přetváří hudební motivy vytvořené zásuvnými moduly do celé skladby, ale také přehrávat samostatně jednotlivé podstromy, což výrazně zvýšilo uživatelské pochopení principu fungování celého generátoru a náhled do chování evolučního algoritmu.

5.2 Generovaná hudba

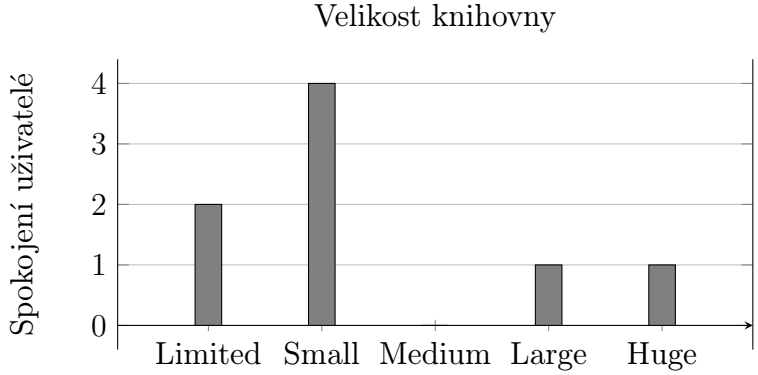
Kromě ovládání programu testující uživatelé hodnotili také kvalitu vygenerovaných skladeb a reakce programu na změny parametrů projektu.

Skupina parametrů, na níž se téměř všichni shodli, byla délka skladby a motivů (viz obrázek 5.1). Všem uživatelům se líbily především kratší skladby (přednastavené hodnoty *Short* a *Very short*), protože delší skladby měly častěji tendenci kombinovat motivy, které se k sobě hodily méně a skladby působily více náhodně.



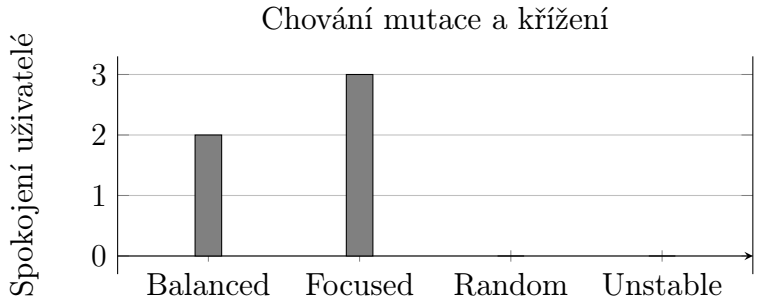
Obrázek 5.1: Největší spokojenost uživatelů podle délky skladeb a motivů. Uživatelé, kteří byli podobně spokojeni s více nastaveními, jsou započítáni pro každé z nich.

Většina dalších parametrů však takto jednoznačné hodnocení neměla (viz obrázek 5.2). Zatímco 3 uživatelé preferovali jedině menší knihovny motivů, u níž chválili znatelnější opakování a odvozování motivů, ostatní naopak raději volili mohutnější knihovny, které zvýšily rozmanitost dostupných melodií.



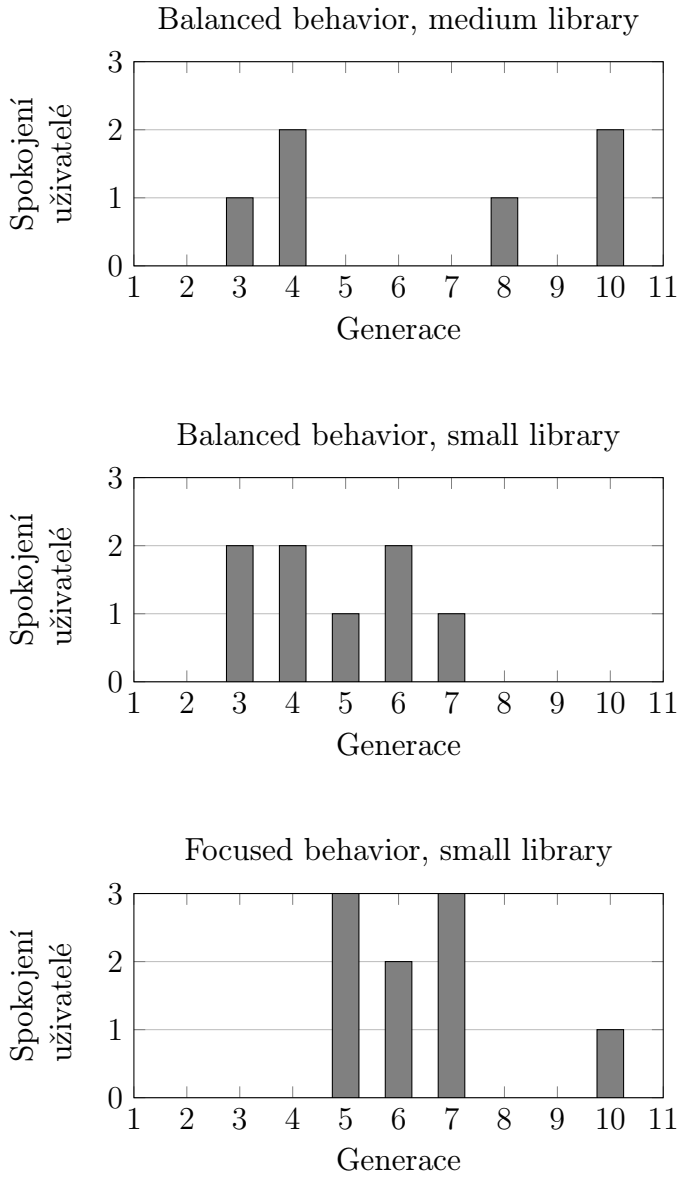
Obrázek 5.2: Největší spokojenost uživatelů podle velikosti knihovny. Uživatelé, kteří byli podobně spokojeni s více nastaveními, jsou započítáni pro každé z nich.

Ti, kteří si vybírali menší knihovnu, obvykle pro chování mutace a křížení volili spíše nastavení odvozené od přednastaveného *Behavior: Focused*, které dává výraznou přednost některým motivům knihovny před jinými a při křížení nepřidává do nových skladeb žádné nové motivy. Na druhou stranu ti, kteří preferovali větší výběr motivů, odvíjeli chování spíše od nastavení *Balanced*, které dává všem motivům ve skladbě rovnocennější postavení a umožňuje větší změny do dalších generací. Viz obrázek 5.3).



Obrázek 5.3: Největší spokojenost uživatelů podle chování mutace a křížení.

Počet skladeb v jedné generaci, který si nechali lidé generovat, se ze začátku pohyboval od 5 do 10, ovšem ve finálních projektech obvykle konvergoval k vyšší z těchto hranic. Počet prozkoumávaných generací do okamžiku, kdy byli uživatelé s výsledkem nejspokojenější, velmi závisel na zvolených nastaveních, ovšem nakonec se obvykle pohyboval mezi 5. a 7. generací (viz obrázek 5.4).



Obrázek 5.4: Největší spokojenost uživatelů se skladbami podle počtu generací. Porovnání spokojenosti ve třech kombinacích velikosti knihovny motivů s chováním křížení a mutace. Uživatelé, kteří byli spokojeni v rozsahu generací jsou započítáni pro každou z nich.

Závěr

Hudba je jedna z nejvýznamnějších tvůrčích činností naší civilizace. Doprovází ji již po tisíce let, vyvíjí se s ní, odráží mnohé rysy a pocity jejích tvůrců. Přestože lze tuto formu umění studovat z mnoha různých úhlů pohledu, otázky kvality, vnímání nebo odezvy posluchačů zůstávají výrazně subjektivní a exaktními vědami těžko uchopitelné.

Žádná umělecká tvorba ovšem nevzniká sama od sebe, bez ovlivnění z jiných zdrojů. Při skládání nové hudby může být těžké přijít s novými nápady, aniž bychom je přímo přebírali z existujících skladeb. Jednou z možností, odkud začít, je generování hudby počítačem, který dokáže produkovat výsledky mnohem rychleji a není nutně zatížen znalostí ostatní tvorby. Výsledná hudba může být využita rovnou k relaxaci, jako podkres k práci nebo právě pro tvůrčí podnět a další zpracování člověkem.

V této práci jsme vytvořili program generující skladby pomocí stromové struktury, která využívá základní společné melodie z knihovny hudebních motivů, jenž strom doprovází, a postupně je přetváří a kombinuje do finální podoby. Generování a úprava jednotlivých melodií je řízena zásuvnými moduly, které lze do programu volně přidávat pro úpravu celé metody použité ke skládání hudby. Vyvinuli jsme také evoluční algoritmus, jenž upravuje a skládá prvky různých skladeb dohromady, čímž lze kombinovat motivy, způsob jejich úpravy a celkovou strukturu z různých skladeb z předešlých generací.

Vše je řízeno sadou parametrů, díky nimž si každý uživatel může přizpůsobit styl vygenerovaných skladeb podle vlastních preferencí. Aplikace má uživatelské rozhraní, které dovoluje tato nastavení, celé projekty i vytvořené skladby ukládat do souborů. Dále zobrazuje hierarchickou strukturu stromů pro jednotlivé skladby a umožňuje samostatné přehrávání hudebních motivů z doprovodné knihovny nebo hudby generované jednotlivými podstromy. Byla vyvinuta uživatelská dokumentace, která popisuje, jakým způsobem lze program ovládat, a několika lidem byla aplikace dodána k otestování. Zpětná vazba od těchto uživatelů dále pomohla doladit aplikaci k jejich spokojenosti a byly prozkoumány jejich preference nastavení projektů.

Samozřejmě by tento způsob tvorby hudby šel nadále rozšiřovat další funkcionalitou. Nové zásuvné moduly by mohly aplikovat různou hudební teorii pro sofistikovanější tvorbu základních hudebních motivů nebo přidat jiné způsoby zpracování těch stávajících. Celá struktura stromu by mohla být blíže upravována jinými moduly, které by umožňovaly pokročilejší koordinaci více hudebních nástrojů nebo explicitní rozdělení skladby na refrény, sloky a další sekce skladby. Případně by během křížení mohl program podrobněji zkoumat obě skladby, hledat společné vlastnosti a zajistit jejich postup do další generace.

Nicméně takovýto pohled na hudební skladby přináší silný nástroj napodobující možný způsob myšlení lidského skladatele a díky uživatelskému hodnocení i rozšiřitelnosti moduly lze snadno přizpůsobit tvorbu hudby dle svých představ.

Seznam použité literatury

Blackwell, T. (2007). Swarm granulation. In Romero, J. a Machado, P., editors, *The Art of Artificial Evolution: A Handbook on Evolutionary Art and Music*, pages 103–122. Springer Berlin Heidelberg. doi: 10.1007/978-3-540-72877-1˙5.

Burraston, D., Edmonds, E., Livingstone, D. a Miranda, E. R. (2004). Cellular automata in midi based computer music.

Dewdney, A. K. (1989). A cellular universe of debris, droplets, defects, and demons. *Scientific American*, **August**, 88–91.

Gamma, E., Helm, R., Johnson, R. a Vlissides, J. (1995). *Design Patterns: Elements of Reusable Object-oriented Software*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA. ISBN 0-201-63361-2.

Gardner, M. (1970). Mathematical games: The fantastic combinations of john conway's new solitaire game "life". *Scientific American*, **223**, 120–123.

Hirata, K., Tojo, S. a Hamanaka, M. (2014). Algebraic Mozart by tree synthesis.

MacCallum, R. M., Mauch, M., Burt, A. a Leroi, A. M. (2012). Evolution of music by public choice. *PROCEEDINGS OF THE NATIONAL ACADEMY OF SCIENCES OF THE UNITED STATES OF AMERICA*, **109**, 12081–12086. doi: 10.1073/pnas.1203182109. URL `http://dx.doi.org/10.1073/pnas.1203182109`.

McCormack, J. (1996). Grammar based music composition. *Complex systems*, **96**, 321–336.

McCormack, J. (2005). *Open Problems in Evolutionary Music and Art*, pages 428–436. Springer Berlin Heidelberg, Berlin, Heidelberg. ISBN 978-3-540-32003-6. doi: 10.1007/978-3-540-32003-6˙43. URL `http://dx.doi.org/10.1007/978-3-540-32003-6_43`.

McDermott, J., Griffith, N. a O'Neill, M. (2007). Evolutionary computation applied to sound synthesis. In Romero, J. a Machado, P., editors, *The Art of Artificial Evolution: A Handbook on Evolutionary Art and Music*, pages 81–101. Springer Berlin Heidelberg. doi: 10.1007/978-3-540-72877-1˙4.

Oracle. Trail: Sound (The Java™ Tutorials). URL `https://docs.oracle.com/javase/tutorial/sound/`. [Online; přístup 26. července 2016].

Ramirez, R., Hazan, A., Marine, J. a Serra, X. (2007). Evolutionary computing for expressive music performance. In Romero, J. a Machado, P., editors, *The Art of Artificial Evolution: A Handbook on Evolutionary Art and Music*, pages 123–142. Springer Berlin Heidelberg. doi: 10.1007/978-3-540-72877-1˙6.

The MIDI Association. The Complete MIDI 1.0 Detailed Specification. URL https://www.midi.org/specifications/item/the-midi-1-0-specification. [Online; přístup 26. července 2016].

Waschka II, R. (1999). Avoiding the fitness "bottleneck": Using genetic algorithms to compose orchestral music. In *Proceedings of the 1999 International Computer Music Conference, ICMC 1999, Beijing, China, October 22-27, 1999*. URL http://hdl.handle.net/2027/spo.bbp2372.1999.376.

Weicker, K. (2003). *Evolutionary algorithms and dynamic optimization problems*. Der Andere Verlag.

Seznam obrázků

Přílohy

Obsah přiloženého CD-ROM

Přiložené CD-ROM obsahuje následující adresáře a obsah:

application přeložený a spustitelný soubor s aplikací pro generování hudby a podadresářem s dodávanými moduly,

documentation programátorská dokumentace programu vygenerovaná z dokumentačních komentářů,

source zdrojové soubory aplikace v jazyce Java SE 8 a sestavovací skripty pro program Apache Ant pro její překlad,

thesis text této práce.

Aplikace využívá ikony Silk[1] vytvořené Markem Jamesem a vydané pod licencí *Creative Commons Attribution 2.5 Generic*[2].

[1] `http://www.famfamfam.com/lab/icons/silk/`
[2] `https://creativecommons.org/licenses/by/2.5/`