Katedra informatiky
Přírodovědecká fakulta
Univerzita Palackého v Olomouci

DIPLOMOVÁ PRÁCE

Metody počítačového generování hudby

2020
Vedoucí práce: Mgr. Petr Osička, Ph.D.

Bc. Petr Valigura
Studijní obor: Aplikovaná informatika, prezenční forma

Bibliografické údaje

Autor:	Bc. Petr Valigura
Název práce:	Metody počítačového generování hudby
Typ práce:	diplomová práce
Pracoviště:	Katedra informatiky, Přírodovědecká fakulta, Univerzita Palackého v Olomouci
Rok obhajoby:	2020
Studijní obor:	Aplikovaná informatika, prezenční forma
Vedoucí práce:	Mgr. Petr Osička, Ph.D.
Počet stran:	61
Přílohy:	1 CD/DVD
Jazyk práce:	český

Bibliograhic info

Author:	Bc. Petr Valigura
Title:	Methods of Music Generation
Thesis type:	master thesis
Department:	Department of Computer Science, Faculty of Science, Palacký University Olomouc
Year of defense:	2020
Study field:	Applied Computer Science, full-time form
Supervisor:	Mgr. Petr Osička, Ph.D.
Page count:	61
Supplements:	1 CD/DVD
Thesis language:	Czech

Anotace

Práce pojednává o metodách počítačového generování hudby. Jsou v ní obsaženy základy hudební teorie, prozkoumání existujících i nových řešení, jejich hodnocení, implementace a vylepšení. Součástí práce je také webová aplikace, která umožňuje vytvářet základní hudební nápady. Aplikace funguje jako platforma pro skladatele, kteří hudbě příliš nerozumí nebo hledají nápad. Práce též obsahuje systém pro uživatelské hodnocení metod.

Synopsis

Thesis deals about computer methods of music generation. It contains basics of music theory, research of existing and new solutions, their evaluation, implementations and improvements. Part of this thesis is also a web application for basic composition of music ideas. The application servers as platform for composers, that don't know a lot about music, or they search for new idea. The thesis also contains a system for user evaluations of the methods.

Klíčová slova: webová aplikace; hudba; JavaScript; neuronové sítě; genetické algoritmy, skládání

Keywords: web application; music; JavaScript; neural network, genetic algorithms, composition

Děkuji vedoucímu této práce Mgr. Petru Osičkovi, Ph.D. za jeho pomoc, rady a nápady, které mi při vytváření této práce velmi pomohly. Také moc děkuji své rodině za podporu nejen při psání práce, ale i při studiu.

Místopřísežně prohlašuji, že jsem celou práci včetně příloh vypracoval samostatně a za použití pouze zdrojů citovaných v textu práce a uvedených v seznamu literatury.

datum odevzdání práce podpis autora

Obsah

Seznam obrázků

Seznam zdrojových kódů

1 Úvod

Hudba provází lidstvo od nepaměti – ať už ve formě popěvků afrických kultur, bojových chorálů, lidových písní nebo těch z žebříčků hitparád. Hudba funguje velmi dobře samostatně, je využívána v divadelních představeních, filmech, počítačových hrách, videích, v aplikacích anebo ve školách jako pomocník při učení. Dokáže navozovat či dokreslovat různé emoce, funguje jako zaměstnání pro mnoho lidí, ať už se jedná o zpěváky, skladatele, producenty, zvukaře, DJ nebo prodavače CD. Hudba má tedy velký význam pro naši společnost. Přes tyto nesporné vlivy, byl donedávna jediným možným způsobem, jak vytvořit nějakou skladbu, práce a talent člověka – skladatele. V posledních letech však přichází způsob nový – počítač. Ten zastoupil lidi již v celé řadě oblastí. Dokáže to ale u tak kreativního úkolu, jakým je hudba? Tuto otázku jsem se rozhodl prozkoumat.

1.1 Historie generování hudby

Nápad generovat hudbu [1] nebo automatizovat aspoň část kreativní kontroly je velmi starý. Příkladem může být *Musikalisches Würfelspiel* (hudební hra s kostkami) z 18. století, konkrétně populární Mozartova verze, kde jsou malé hudební útržky náhodně přeuspořádány házením kostky, které poté vytvoří celou skladbu. Nejedná se však o nejstarší příklad. Historicky starší je například jiná hra s kostkami – *Der allezeit fertige Menuetten und Polonaisencomponist* (Johann Philipp Kirnberger). Nebyli však jediní. Mezi lety 1757 až 1812, minimálně dvacet hudebních her s kostkami bylo publikováno [2]. Navíc existovaly i způsoby, které kostky nevyužívaly. Důvodem vzniku pak bylo hlavně umožnění i méně zkušeným skládat hudbu. Některé způsoby, jak využít náhody při skládání, můžeme vysledovat i v letech pozdějších. Příkladem může být *Atlas Eclipticalis* Johna Cage. Jeho princip spočíval v tom, že složil píseň pomocí průsvitného papíru umístěného náhodně tak, aby bylo možné poznačit si hvězdy na obloze. Ty pak představovaly noty. Velkou revoluci v generování hudby však přinesly až počítače. Ještě před jejich praktickým využitím, v době počátečních návrhů pro obecné počítací zařízení, byly vedeny úvahy nad využitím strojů a generování hudby [3]. Jedno z prvních skutečných využití počítače v hudbě přinesli však autoři Hiller Jr a Isaacson, v roce 1957, počítačem ILLIAC I [4]. Od té doby prošlo generování hudby velkým vývojem. Navzdory tomu se však stále jedná o špatně definovaný problém a není jasné, co všechno, a jak dobře, dokáže počítač vygenerovat, jaký je vztah mezi jednotlivými systémy, co je přesně cílem u úkolu kompozice a celá řada dalších otázek [1].

1.2 O problému

Skládání nebo komponování hudby [5] počítačem přináší hned několik problémů. Vytvořit dobrou hudbu není jednoduchá věc ani pro člověka. Ten musí často umět hrát na hudební nástroj, znát základy hudební teorie nebo mít talent. V ideálním

případě zvládá všechny tyto tři věci. Ale ani to mu nezaručuje složení dobré skladby.

1.2.1 Hodnocení výsledků

Dalším problémem je určení toho, co můžeme pokládat za dobrou skladbu [6]. Ačkoliv primárním tématem této práce není ohodnocení výsledků generování, v rámci zkoumání metod samotných, je, dle mého, vhodné vědět, jak kvalitní skladby dokážou metody vygenerovat. V případě některých je to dokonce nutnost (viz 4.5 a 4.6). Hudba je určena primárně člověku, tedy zřejmým způsobem hodnocení je právě lidský posluchač. Existují však i jiné způsoby, a to použití hudební teorie nebo umělé inteligence. Co zvolíme závisí i na cíli metody – jestli chceme generovat skladbu podobnou určitému žánru či dokonce existující písni, dále na prostředcích ať už časových či výpočetních nebo skladbě, která by se měla posluchači co nejvíce líbit a dalších kritériích.

Hodnocení lidským posluchačem se na první pohled může zdát jako nejlepší. Uživatel si poslechne píseň a ohodnotí ji. Problémem je zde však doba hodnocení. Uživatel totiž, po nějakém čase, může být poslechem hudby, který nemůžeme urychlit, unaven a samotný proces zabere jednomu subjektu dost času [7]. Takže ačkoliv člověk dokáže dobře ohodnotit kvalitu skladby (pomineme-li různý hudební vkus lidí) není vhodný pro širší hodnocení. V takovém případě by bylo nutné zvýšit počet hodnotitelů, případně zvolit jiný typ hodnocení.

Při tomto úkolu by také šla využít pravidla hudební teorie (viz 4.5). Jejich použitím by šel vytvořit systém nebo fitness funkce, které by se daly použít pro rychlé a automatické hodnocení, jenž by netrpělo unaveností. Problém pravidel je však v jejich získávání a omezeném množství. Lze se spolehnout jen na část stylů, které byly popsány hudebními teoretiky, jenž jsou často příliš obecné a chybí jim konkrétní charakter. Takové ohodnocení, dle mého, však nereflektuje přímý stav tak, jak hudbu vnímá člověk. A i u malého množství pravidel je problém vybalancovat velkou závislost na daném stylu, autorovi nebo skladbě a právě velkou obecnost.

Třetím způsobem, který částečně řeší slabé stránky těch předešlých, jsou metody strojového učení (viz 4.6). Naučení na „správných“ skladbách můžeme totiž využít nejen při generování hudby, ale i jejím hodnocení. Přináší, podobně, jako u metody pravidel hudební teorie, rychlost hodnocení, neúnavnost a zároveň není třeba pracovat se složitými pravidly hudební teorie, které jsou často složité nebo vůbec neexistují. Jejich problémem je však úzká závislost na učených skladbách, což by mohlo způsobit problém. Kdyby daná skladba byla dobrá například v žánru jazz (byla by vygenerovaná z jazzových písní, například pomocí Markových řetězců (viz 4.3)), mohly by hodnotící metody, které se na jazz nezaměřují, skladbu ohodnotit špatně. V podstatě je tak nemožné využít je při různých hudebních stylech, respektive bylo by třeba mít celou řadu hodnotících modelů pro různé žánry.

Ve své práci jsem se rozhodl využít hodnocení uživatelů. Důvodem je to, že se metodami nezaměřuji jen na konkrétní hudební styl, často písně tvořím

bez předlohy a části využívané u hodnotících metod založených na pravidlech hudební teorie a strojového učení již využívám v jiných metodách generování skladeb. Tím, že je používám v metodách, by mohlo dojít ke zkreslení, kde by metoda založená na hudební teorii dostala větší hodnocení jen z toho důvodu, že splňuje pravidla, která byla použita při samotném generování. Pro zbavení problémů s hodnocením uživatelů – únavou a různým hudebním vkusem, jsem se rozhodl využít hodnocení velkého množství uživatelů. Těm, abych předešel špatně vybraným výsledkům, poskytnu místo skupinky generovaných skladeb (jelikož by mohly, tím, že je vyberu já, ovlivnit reálné hodnocení uživatelů) celou aplikaci. Uživatel si tak bude generovat skladeb kolik chce, tedy nedojde tak snadno k vytížení času a případné únavě z poslechu. Také to lépe zachytí obecný hudební vkus uživatelů, který se může lišit. Díky tomu bude, dle mého, kvalitněji zachycen charakter metod. Navíc to mohu využít na všechny metody s předpokládanou stejnou úspěšností.

1.2.2 Předmět generování

Třetí hlavní problém, který zde budu řešit, je rozhodnutí co generovat [1]. Generování hudby je velmi rozsáhlý problém. Spadá zde generování rytmů, melodií, harmonií, ale dokonce i akustické charakteristiky zvuku. Na těchto možnostech můžeme navíc stavět další tím, že je ovlivníme emocemi, příběhem, uživatelskou interakcí a omezením toho, jak složité na hraní mají být. Často tak lze narazit na práce zaměřující se pouze na jednu věc, kterou generovat.

V rámci zachování zadání, přistoupím k tomuto problému tak, že se zaměřím na generování toho, co intuitivně vnímám jako hudbu, tak jak je obecně prezentována při hudební výchově na základních školách. Hudbou tak budu rozumět hlavně melodii s rytmem (případně harmonií), aby bylo možné si výslednou skladbu zazpívat, případně zahrát na hudební nástroj. Díky tomu bude odpovídat i hudbě jakou známe z hudebních notací. Tuto melodii budou skládat hlavní zástupci metod, sloužící pro generování bez modifikací, které by mohly narušit charakter a srovnání metod.

1.2.3 Výběr reprezentace

Posledním z velkých problémů je určení, jaká bude reprezentace hudby (viz 3), ať už pro generování nebo učení. Nabízí se dvě základní možnosti pracovat s hudbou jako posloupností úrovní analogového zvukového signálu (vzorků), jednoduše poslouchatelnou hudbou typicky uloženou v souborech s příponou WAV nebo MP3 anebo využít hudbu v nějaké notaci – tedy ve formátu ve kterém není přímo hudba uložena, ale je tam pouze popis toho, co a jak hrát. Z důvodu jednodušší práce, analyzování, velikosti a lepšího získání datasetu jsem se rozhodl pro možnost druhou, s možností převedení výsledku právě do formátu WAV.

1.3 O aplikaci

Na problém generování hudby se nechci dívat pouze teoreticky, ale i implementovat hlavní zástupce metod a pokud možno, i přivést vlastní inovaci. Vytvořím tak webovou aplikaci, která nejenže bude metody sdružovat a používat, ale umožní hudbu skládat i bez nich. Aplikace tak bude představovat platformu pro skladatele nehledě na úrovni jejich znalosti z hudební teorie, tudíž si budou moci buď skladbu složit od začátku sami nebo využít jednu z metod a skladbu si poté upravit. Některé metody navíc budou moci stavět na jejich hudebních myšlenkách a nějakým způsobem ji i dokončit. Výsledek bude možné vygenerovat v souboru MIDI nebo WAV, takže ho budou moci hned poslouchat nebo si ho převést do jejich oblíbeného DAW[1].

Díky tomu, že se jedná o aplikaci webovou, bude více přístupnější lidem, kteří v rámci používání budou moci jejich vygenerované skladby hodnotit a tím bude mít aplikace, na rozdíl od jiných prací, silnou hodnotící metodu, která nebude trpět špatným výběrem skladeb, ale naopak by se zde mělo projevit dlouhodobé používání metod. Navíc nebudou tolik závislé na vkusu uživatelů. Aplikace v rámci této funkcionality bude v současné době jedinečná – nenašel jsem podobné řešení. Jazykové prostředí je voleno poněkud nestandardně anglicko-české, kde část editoru je anglicky z důvodu zavedeného označování funkcí, pro které jsem v českém jazyce nenalezl vhodný překlad. Dále také kvůli technologiím, které pracují s anglickými verzemi slov a hudební teorií (viz 2). Česky je pak samotná sekce metod, pro zachování užšího vztahu s textem této práce a používání stejné terminologie u metod.

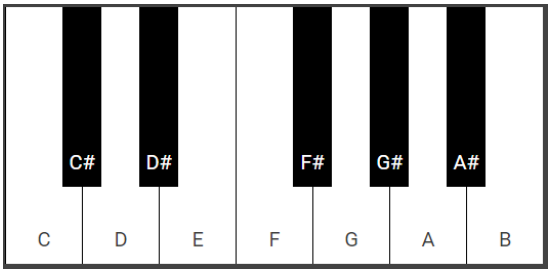
[1]Digital Audio Workstation - Software určen pro nahrávání, editaci a produkování hudby

2 Hudební teorie

Pro pochopení principů použitých v metodách a částečně i pro uživatelské používání aplikace (ačkoliv zde ve většině případů stačí velmi malá znalost) je klíčová znalost hudební teorie. V této části uvedu nejdůležitější základní pojmy, bez kterých metodám nelze porozumět.

2.1 Výběr hudební teorie

V této sekci uvedu přehled znalostí hudební teorie, který je nutný pro porozumění zbytku textu. Obecně však budu vycházet z české (obecně středoevropské) a ze západní hudební teorie (používané například v Anglii, Americe), které jsou velmi podobné [8]. Důvodem je její rozšířenost ať už u znalostí lidí, použité literatury, populárních hudebních nástrojů nebo knihoven a technologiích použitých v této práci (viz 5). Pro snadnější pochopení popíši některé části na klaviatuře piana na obrázku 1, která dobře odpovídá mnou vybranou hudební teorii. Navíc je obecně známá a je využita i v samotné aplikaci, jako způsob pro zadávání not.

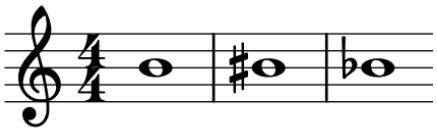


Obrázek 1: Klaviatura piana.

Klaviatura piana se skládá ze skupin 12 kláves [9]. Počet skupin závisí na velikosti samotného piana a liší se dle výšky tónu (vlevo je tón nejnižší a vpravo naopak nejvyšší). Sedm z těchto dvanácti kláves je bílých a odpovídající tónům C, D, E, F, G, A, B. V české hudební teorii se místo B používá H, ale v této práci zvolím variantu s B. Důvodem je využití zahraničních knihoven a častějšího použití tohoto značení v různých hudebních editorech – jedná se však o stejnou notu. Zbylých pět kláves (černé barvy) odpovídá tónům C♯ (D♭), D♯ (D♭), F♯ (G♭), G♯ (A♭), A♯ (B♭) - V rámci hudební teorie používané v Česku, narazíme na označení B♭ jako „bé“. Kvůli konfliktu se zahraničním B však toto značení používat nebudu. Znak ♯ znamená zvýšení o polovinu tónu, takzvaný půltón, tedy například C♯ značí, že výška tónu C byla zvětšena o půltón. Naopak ♭ značí jeho snížení o půl tónu (značení v hudební notaci na obrázku 2).

V rámci piana je možné si všimnout, že například C♯ a D♭ jsou stejné tóny, a tak s nimi budu také pracovat. V hudební teorii jsou však stanovená pravidla, která v této práci nejsou podstatná, specifikující značení v konkrétní skladbě. Klávesa je vyšší o půltón než její nejbližší levý soused a menší o půltón než

soused pravý. Tyto informace implikují různé názvy stejných tónů – například E♯ a F. V této práci budu využívat označení bez ♯ či ♭ a jinak variantu s křížkem.

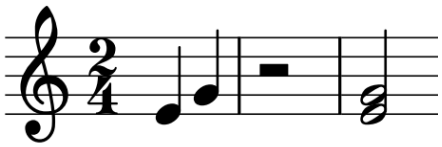


Obrázek 2: Přirozená, zvýšená, snížená nota.

2.2 Intervaly

Pojem interval [10] se používá při popisu vzdálenosti dvou not, která se měří jako rozdíl ve výšce tónu. Když nezní současně, ale postupně po sobě, hovoříme o něm jako o melodickém, jinak je to interval harmonický (značení v hudební notaci na obrázku 3).

Obecně číslo vzdálenosti v diatonické stupnici (více v následující sekci) značíme tvary vycházející z latiny, to jest: **prima, sekunda, tercie, kvarta, kvinta, sexta, septima a oktáva** (anglicky pak unison, second, third, fourth, fifth, sixth, seventh, eighth). Vyšší intervaly než 8 nás obecně nezajímají, jelikož se vztahy opakují.



Obrázek 3: Interval melodický a harmonický.

Číslo vzdálenosti může být stejné, ačkoliv je možné, že se bude lišit v takzvané kvalitě [12]. Pro kvalitu je klíčové spočítat přesné číslo půltónů a tónů v daném intervalu. Například u noty C a D je interval sekunda, stejně, tak u noty E a F, nicméně počet půltónů se liší – C na D je celý tón, tedy dva půltóny, z E na F je to pouze jeden půltón. Proto je nutné zavést značení této kvality. Já zde použiji anglické výrazy z důvodu použitých technologií a literatury:

- **Seconds (sekunda)** - major (dva půltóny), minor (jeden půltón)
- **Thirds (tercie)** - major (čtyři půltóny), minors (tři půltóny)
- **Fourths (kvarta)** - perfect (pět půltónů), augmented (šest půltónů)
- **Fifths (kvinta)** - perfect (sedm půltónů), diminished (šest půltónů), augmented (osm půltónů)
- **Sixths (sexta)** - major (devět půltónů), minor (osm půltónů)
- **Sevenths (septima)** - major (jedenáct půltón), minor (10 půltónů)

2.3 Stupnice

Jak bylo dříve zmíněno, máme dvanáct not a to v takzvané oktávě. Stupnice [10] je pak sada not vybraná z těchto dvanácti. Skladatelé často vytváří melodie a akordy pro harmonizaci používajíc právě noty z nějaké stupnice. Ty mohou mít různý počet not. V této práci využiji zejména takzvané diatonické stupnice, ty jsou heptatonické (sedmitónové), s intervalem celého tónu nebo půltónu mezi dvěma sousedními notami (celý tón se tu tak vyskytuje pětkrát a půltón dvakrát). V praxi narážíme nejčastěji na dvě diatonické stupnice: durovou (anglicky *major*, její třetí interval - third/tercie je právě major) a mollovou (anglicky *minor*, její třetí interval - third/tercie je minor) [12]. Pro intuitivnější chápání, je dle mého, vhodné využít klaviaturu piana (z obrázku 1) a konkrétní příklad.

Vezměme si stupnici C dur, ta začíná na klávese C a její další tóny jsou ostatní bílé klávesy, tedy C, D, E, F, G, A, B, C. Při analýze jednotlivých intervalů si můžeme všimnout, že je zde určité střídání tónů a půltónů a to: celý, celý, půlový, celý, celý, celý, půlový[2]. Tyto kroky tak definují tuto (durovou) stupnici. Pokud bychom začali například na klávese A a dodrželi tento vzor dostaneme A, B, C♯, D, E, F♯, G♯, A tedy A dur. V praxi se využívá označení tónů jednotlivých stupňů, ty se značí pomocí římských čísel a názvů:

I. Tónika, II. Supertónika, III. Medianta, IV. Subdominanta, V. Dominanta, VI. Superdominanta, VII. Subtónika.

Dané názvy jsou také využity při značení akordů stojící na příslušném stupni (viz 2.4).

Co když začnu od tónu A a zahraji po něm následující bílé klávesy? Určitě se bude jednat o tóny vyskytující se v C dur, intervaly však budou posunuty. Dostaneme celý, půlový, celý, celý, půlový, celý a celý tón, což je právě vzor mollové stupnice. V tomto případě, konkrétně tóny A, B, C, D, E, F, G, A je stupnice a moll. Vztah mezi C dur a a moll nazýváme relativností. Tedy a moll, je relativní stupnice k C dur. Každá durová stupnice má relativní mollovou stupnici, která začíná 6 notou (nebo stupněm) durové stupnice. Podobný posun je možné aplikovat i na další stupně - viz 2.7.

Mezi další stupnice patří například pentatoniky (stupnice obsahující pět not) nebo stupnice chromatické (ty obsahují not dvanáct). V práci je také, v menší míře, využiji. Dále využiji ještě dvě formy stupnice mollové, tj. její melodickou a harmonickou formu. Ty mění tu přirozenou v některých intervalech, díky čemuž zní obecně lépe.

2.4 Akordy

Kombinací dvou nebo více not dostaneme akord [10]. Ten máme možnost zahrát dohromady nebo postupně tzv. *arpeggio*. Obecně akordy tvoříme pomocí výběru not ze stupnice, které mezi sebou tvoří tercii. Často používaným typem akordu jsou pak triády, ty jsou brány jako základ harmonického systému. Složky triády

[2] Anglicky se celý krok značí W a půlový H.

se označují jako *root*, *third*, *fifth* a jsou založené na intervalech, tedy jsou určující v případě prohození pořadí not. Pořadí not v akordu je možné zpřeházet, takže například akord C dur (anglicky C major) se v základní pozici skládá z C, E, G (root je nejhlubší nota - je tedy v root pozici). Je však možné využít takzvané inverze a vytvořit a použít variantu E, C, G (third je nejhlubší tón - je to první inverze) nebo G, C, E (G je nejhlubší tón - je to druhá inverze).

Vezmeme-li výše uvedené durové a mollové stupnice a dále pravidlo o vytvoření akordu výběrem noty pomocí intervalu tercie, dostaneme 7 základních akordů v každé stupnici, které využívají její noty. Vzniklé triády (respektive intervaly v nich) se mohou lišit. Můžeme je tak rozdělit na 4 druhy. V rámci použité literatury a zachování označení použitého u intervalů, zde uvedu anglické názvy:

- **Major** (skládá se z major third, a perfect fifth)
- **Minor** (skládá se z major third, a perfect fifth)
- **Augmented** (skládá se z major third, a augmented fifth)
- **Diminished** (skládá se z minor third, a diminished fifth)

Obecně nejčastěji narazíme na „major a minor“ triády. Naopak setkat se s „augmented“ nebo „diminished“ triádou je složitější, jelikož díky své disharmonii zní hůře.

Jak jsem již zmínil, římská čísla se nepoužívají jen u intervalů, ale právě i u akordů, kde stupeň odpovídá root notě ve stupnici. Toto značení se často používá z důvodu nezávislosti na konkrétní stupnici a toho, že se opakují ve velkém množství písní. Například v jazzu je akordová progrese II - V - I brána jako úplný základ mainstreamu tohoto žánru a nachází se ve velkém množství jazzových písní a standardů [13]. I v jiných stylech se některé progrese vyskytují často, písně se pak liší třeba jen melodií. Při skládání je tak možné využít libovolnou populární akordovou posloupnost a stejně vytvořit vlastní píseň. Obecně se pro označení mollových (minor) akordů používají malá římská čísla a naopak u durových (major) velká.

Triády nejsou jediné možné akordy. V rockových písních můžeme narazit na takzvané *power akordy*, které za notou označující root mají číslo 5 (skládají se z root noty a kvinty, chybí jim tak charakter tercie), v jazzu na čtyřnotové akordy (obsahující triádu a septimu) končící číslem 7 a i u nich je celá řada typů. V rámci tohoto textu jsou však nepodstatné. Posledním, zde použitým typem akordů, jsou *sus* („suspended“) akordy [11]. Jedná se o akordy, ve kterých je tercie nahrazena typicky kvartou nebo sekundou. To, jaký akord kde patří, má korelaci s tóninami a funkční harmonií.

2.5 Tónina

U skladeb je obecně možné určit její tóninu [10], což je jakási sada tónů obsahující informace o skladbě. Názvy tónin odpovídají názvům stupnic – například C dur,

d moll aj., což implikuje využití not ze stupnice v analyzované písni. Určit to, o jakou stupnici se jedná, není vždy triviální. Například C dur a a moll obsahují stejné noty a navíc během skladby může dojít ke změně tóniny – k tzv. modulaci. Při určování tóniny je třeba se zaměřit na vztahy všech not k jedné, k té centrální (k tónice). Není zvláštní, že na tónice začínáme i končíme a celou dobu nás k tomu posloupnost akordů a not vede. Znalostí tónin můžeme omezit množství not na ty, které zní relativně dobře. Funkce pak určuje vztah akordu nebo stupně k tonálnímu centru. Přesnější určení je nad rámec tohoto textu.

2.6 Transpozice

Vezměme si libovolnou skladbu. Pokud v ní každou z not posunu o stejné číslo půltónu jedním směrem, dojde k posunu dříve zmíněné centrální noty, tedy ke změně tóniny. Na druhou stranu však vztahy not zůstanou zachovány tak, jako u stupnic, kde nás tolik nezajímají samotné noty, ale spíše vzor intervalů. I tady nám bude znít skladba velmi podobně. Lidé, kteří nemají referenční tón nebo absolutní hudební sluch ani nemusí poznat rozdíl. Noty budou tedy o nějakou konstantu posuny. Toho využívají zpěváci, kteří by, kvůli rozsahu svého hlasu, nebyli schopni skladbu bez úpravy zazpívat. Já toho naopak využiji při rozšíření datasetů[3] (viz 4.6.6).

2.7 Tonalita a modalita

V předešlých odstavcích jsem popisoval hudbu tak, jak je nám v mainstreamovém světě nejčastěji prezentována. Zaměřil jsem se na stránku tonality [12], což je nějaký organizovaný vztah tónů v hudbě kolem tonálního centra, kde ho ostatní tóny podporují nebo k němu vedou. Je tak úzce svázán s pojmem tónina. S takzvanou kadencí, která souvisí s ukončováním hudební myšlenky tak, aby dále produkovala otázku nebo naopak skladbu ukončovala. Dala by se tak přirovnat k ukončení věty v přirozeném jazyce. Závěrečným akordem kadence by měla být tónika. Nejjednodušší kadencí je postup: tónika – subdominanta – dominanta – tónika. Je pak možné určovat druhy kadencí. Modalita [10] na druhou stranu referuje na volbu not, mezi nimiž existuje daný vztah. Zde jsme blíže u termínu stupnice. Obecně je možné si módy představit jako posun stupnice o nějaký stupeň (tak jak byl definován výše), podobně jako jsme posouvali durovou stupnici, abychom získali mollovou. Ve skutečnosti přirozená mollová stupnice je 6 mód tzv. Aeoliian. Celkový přehled: **Ionian (I), Dorian (II), Phrygrian (III), Lydian (IV), Mixolydian (V), Aelioan (VI), Locrian (VII)**.

Vztah mezi těmito dvěma „druhy“ nemusí být vždy zcela jasný, často se prolínají. V tonalitě lze zkonstruovat velké množství módů.

V této práci budu přistupovat jednoduše k tonalitě jako k systému 12 příbuzných durových a mollových stupnic, které slouží jako základ pro vytváření triád (případně s přidáním septimy) kolem centrální tóniky. Tyto akordy pak

[3]Sada dat určená ke strojovému učení.

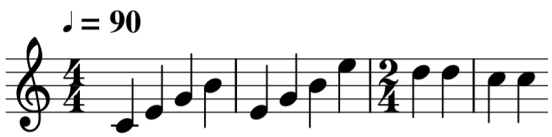
Ionian	I	W–W–H–W–W–W–H	C–D–E–F–G–A–B–C
Dorian	II	W–H–W–W–W–H–W	D–E–F–G–A–B–C–D
Phrygrian	III	H–W–W–W–H–W–W	E–F–G–A–B–C–D–E
Lydian	IV	W–W–W–H–W–W–H	F–G–A–B–C–D–E–F
Mixolydian	V	W–W–H–W–W–H–W	G–A–B–C–D–E–F–G
Aelioan	VI	W–H–W–W–H–W–W	A–B–C–D–E–F–G–A
Locrian	VII	H–W–W–H–W–W–W	B–C–D–E–F–G–A–B

Tabulka 1: Příklad modality na stupnici C dur

spolu budou definovat posloupnosti k definici centrálního tónu. Tonalita je tedy blíže tomu, co obecně vnímáme jako běžnou harmonii. Modalita má naopak blíž k melodii.

2.8 Rytmus a časové předznamenání

Mimo to, jaký tón hrajeme a v jakém pořadí, záleží i na tom, jak dlouho ho hrajeme [14]. Délka tónu může být zvlášť v počítači různá. V hudební teorii je zvykem k tomu přistupovat s určitými pravidly. Celá skladba je rozdělena na takty (anglicky *bar* nebo *measure*) a v každém taktu může být pouze tolik not, aby se jejich doba trvání rovnala „hodnotě“ taktu. Tuto hodnotu udává časové předznamenání. Jedná se o 2 čísla, kde první udává počet dob v taktu a druhé délku trvání jedné doby, které je vyjádřeno v hodnotě trvání noty. Noty jsou totiž rozděleny dle trvání na **celé, půlové (polovina celé), čtvrťové (polovina půlové), osminové (polovina čtvrťové) a šestnáctinové (polovina osminové)**. Podobným dělením dvěma se dá postupovat i dál, ale v praxi to není moc časté. Pokud naopak chceme zachytit pouze čas ve kterém se nehraje, vkládáme takzvané pomlky (anglicky *rest*). Časové předznamenání se může v průběhu písně měnit. To, jak dlouho skladba bude tedy nakonec reálně trvat, záleží na tom, kolik trvá jedna doba (udává se v počet dob za sekundu anglicky *beats per seconds*, zkratkou BPM) a na časovém předznamenání. Je možné vyjádřit i jiné trvání tónu, od výše zmíněných, pomocí speciálních značek. Např. tečky, která prodlouží trvání noty o její polovinu, čarou pro spojení noty či symbolem trioly, kdy zahrajeme tři noty v jedné době a další. Díky tomu můžeme získat i trvání, které je odlišné od standardních not. Tyto údaje bývají uvedeny například v notaci 5.



Obrázek 4: Příklad časových údajů v notaci.

3 Reprezentace

To co běžně pokládáme za hudbu může mít v digitálních zařízeních celou řadu reprezentací a formátů. V této části popíši ty nejdůležitější a představím mnou vybranou reprezentaci a důvody jejího výběru.

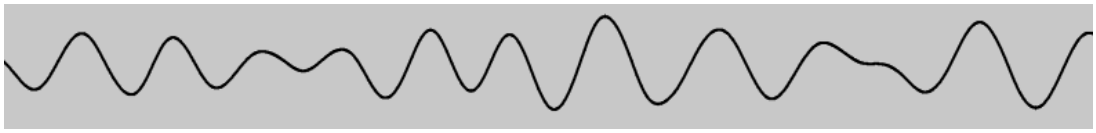
3.1 Noty

Se základní reprezentací hudby, tedy s notami, se setkáváme již na základních školách [9]. Jejím cílem je ukázat, jak se daná skladba hraje. Sama o sobě nelze přehrát. Základem not je grafické označení tónů, které se mají hrát (případně zpívat). Různé formy not lidé využívají již tisíce let. Dnešní noty poskytují informace nejen o tom jaké tóny hrát (určeno dle vertikální pozice v notové osnově) a jak dlouho je hrát (dle vyplnění a stylu nožičky), ale také s jakou dynamikou, v jakém tempu (kolik BPM zabere třeba čtvrťová nota), na jaký nástroj atd. Noty jsou převážně používány muzikanty. Ať už v partituře orchestru nebo ve zpěvníku s názvy akordů napsanými nad takty. Hodnota grafické notace záleží na klíči, kde mezi nejpoužívanější patří houslový, případně basový klíč, které se většinou kombinují. Pianista pak často hraje tóny v basovém klíči levou rukou a v houslovém pravou rukou. U klíčů je napsáno časové předznamenání a tónina. V práci bylo ukázáno již několik příkladů, například na obrázku číslo 5.

Pro použití generování hudby, dle mého, však nejsou vhodné. U uživatele by byla vyžadována jistá odbornost, aby porozuměl výsledku. Výsledek nelze jednoduše přehrát a v případě použití not z obrazů (u strojového učení) by bylo třeba použít některou z metod analýzy a zpracování obrazu. Také by bylo nevyhovující uchování velkého množství těchto obrazů.

3.2 Zvuková vlna

Reprezentace vychází ze zvuku z fyzikálního hlediska [16], který je digitalizovaný vzorkováním a kvantizací. V počítači je tak reprezentován jako posloupnost hodnot amplitud v čase. Jeho hlavní výhodou je to, že je možné ho rovnou přehrát, rozšířenost v multimediálních systémech a využití jen metod pro zpracování audia. Obtížné je však propojení s hudební teorií. Příkladem může být problém, který je v ostatních (zde uvedených) reprezentacích velmi triviální – určení tónu, který hraje. Zvuková vlna vyžaduje analýzu jednotlivých částí, která je, zvlášť v případě polyfonních zvuků, obtížná, a ne stoprocentně spolehlivá [18]. Některé metody by zde navíc byly těžce využitelné a velikost dat zbytečně velká. Proto jsem se rozhodl tento způsob reprezentace u metod generování nevyužít. Vhodný je však v případě finálního generování zvuku, kde využívám jednu z jeho výhod a to tu, že výsledek lze rovnou přehrát a uživatel tak neskončí jen se souborem říkající, jak se má jeho výtvor hrát.



Obrázek 5: Zvuková vlna ve formě sinusoidy.

3.3 ABC notace

V případě předešlých byla největším problémem analýza a velikost, či složitá interpretace člověkem, díky čemuž se, v mém případě, staly velmi nevhodnými. ABC notace [15] je hudební notace založená na ASCII, je tak možné s ní pracovat v textových editorech. V základní verzi nalezneme, jak je vidět na obrázku 6, informace o názvu, časovém předznamenání, délce a tónině. Největší výhodou je však přístupnost pro běžné lidi. Obsahuje totiž značení not tak, jak je známe ze sekce základní hudební teorie – například tón G se značí písmenem G. Díky tomu, že se z praktického hlediska jedná jen o text jak se co má hrát, zabírá malou velikost a dá se jednoduše analyzovat. V případě neuronových sítí zde máme rovnou reprezentaci, která se dá využít při učení. Jedná se tak (v mém případě) o rozumnou volbu. Nicméně tato notace není tolik rozšířená. Neumožňuje označit vše potřebné. V případě, že bych s ní chtěl lépe pracovat, by bylo nutné si ji stejně načíst, nějakým způsobem pozměněnou do paměti, a navíc její možnost přehrávání nebo využití v DAW aplikací, by bylo velmi slabé a často bychom ji museli konvertovat do jiných reprezentací. Proto jsem se rozhodl ji nevyužít.

```
X:2873
T:Folk song
C:anon.
O:France
M:3/4
L:1/4
K:C
ccc|BBB|ABc|G3|
FFF|EEE|DDD|C3|
CDE|CDE|CDE|F3|
DEF|DEF|DEF|G3|
```

Obrázek 6: Příklad části souboru s ABC notací.

3.4 MIDI

MIDI neboli Musical Instrument Digital Interface [19] je standard, který popisuje komunikační protokoly, digitální rozhraní a konektory, které propojují různé druhy hudebních nástrojů, počítače a další audio zařízení. Používá se pro přehrávání, upravování, ale i nahrávání hudby. Komunikace probíhá ve formě tzv.

MIDI zprávách. MIDI soubory tak, spíše než zvukové vlny, které by se rovnou přehrávaly, obsahují popis toho, co hrát. Tato data nejsou velmi objemná, dají se velmi dobře analyzovat, využívají aspoň základy hudební teorie a můžeme tak pohodlně pracovat se samotnými noty a časovými předznamenáními. MIDI slouží dobře k základnímu tvoření nápadů. Můžeme vytvořit soubor obsahující několik MIDI stop, kde můžeme zakomponovat více nástrojů, díky čemuž můžeme jednoduše sestavit hudební nápad – vzít melodii z klavíru, harmonické akordy z kytary a přidat rytmické bicí. MIDI je jednoduché na používání. Uživatel nemusí umět hrát na nástroje ani znát hudební teorii. Stačí využít MIDI editor a jen si „naklikat“, případně změnit píseň k obrazu svému. MIDI se dají relativně jednoduše přehrát.

Kvůli těmto důvodům jsem se rozhodl využít technologie MIDI napříč celou aplikací, a to nejen u metod, ale i u editoru, který umožní uživateli zpracovat hudební myšlenku, ať už jeho vlastní nebo nějakou z vygenerované metody. Z nastudování metod generování hudby mně také vyšlo, že reprezentace pomocí MIDI bude nejvíce reprezentativní a dokáži s její pomocí implementovat největší množství metod.

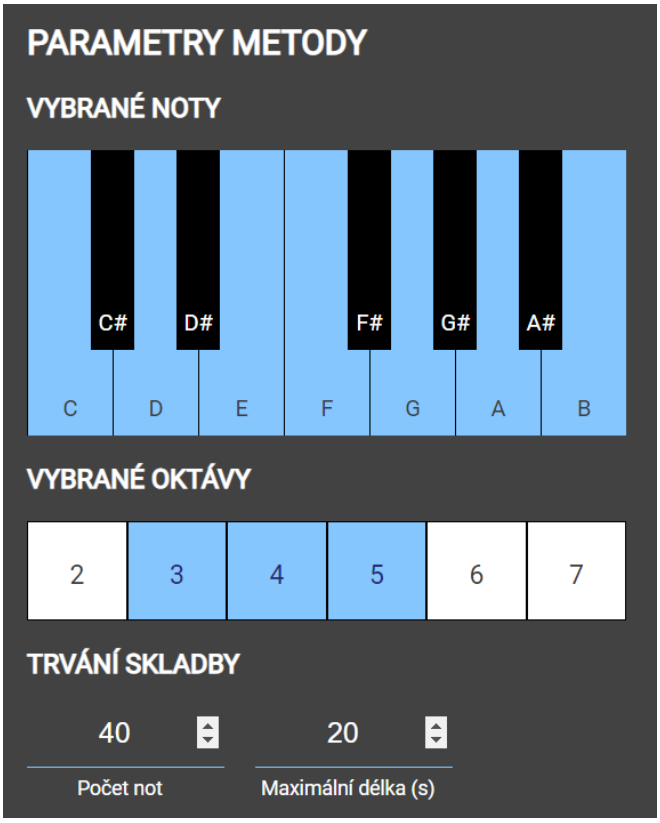
4 Metody

4.1 Random

Tato metoda, kterou jsem nazval Random a navrhl i vytvořil ji sám, vychází z naivní představy a z otázky co se stane, když necháme počítači, konkrétně jeho generování pseudonáhodných čísel, volnost ve všem. Tedy v tom, jakou notu zvolíme, jakou oktávu, v jaký čas ji do celého díla zasadíme a jak dlouho bude trvat. Z tohoto pohledu je to nejjednodušší algoritmus použitý v této práci. Obecně by se tak dal přirovnat k člověku, který neví nic o hudbě a hraje na piano, které neslyší, což je v reálném světě velmi nestandardní situace. Metoda tak spíše slouží k představení pro uživatele, jak se samotným prostředím pracovat. S ostatními metodami se pracuje (z uživatelova hlediska) podobným stylem, liší se jen dle vstupů.

4.1.1 Vstupy

V rámci zlepšení výsledků, jsem umožnil uživateli používat jím vybranou množinu not, kterou zvolí pomocí komponenty piana. Díky tomu budou použity jen noty z určité stupnice, a tím by mělo dojít ke zlepšení výsledků na úkor náhodnosti, která by však byla, bez tohoto omezení, příliš vysoká. Dalšími možnostmi vstupu jsou oktávy, počet not i délka celé skladby. Jejich cílem je lepší distribuce not v rámci celé skladby. Všechny parametry jsou zobrazeny na obrázku 7.



Obrázek 7: Vstupy metody Random.

4.1.2 Hodnocení

Jednoduchost algoritmu je vykoupena jeho slabými výsledky. Metoda bohužel, jak by se dalo očekávat, neposkytla, z mého pohledu, žádný dobrý výsledek (při vstupech, které odpovídají běžně dlouhé, například dvouminutové skladbě s možnými modulacemi). Často se ve skladbách vyskytovalo ticho a délka trvání not zapříčinila to, že i když metoda vygenerovala posloupnost například tří not, které tvořily intervaly s nějakou používanější kadencí, byly od sebe tak daleko, že jsem neměl pocit nějakého propojení. Jeho výsledky jsou v porovnání s ostatními metodami daleko nejhorší. Metoda tedy v žádném případě není schopna generovat celé skladby.

Problémy vycházející z hudební teorie (viz sekce 2) jsou špatné časové vlastnosti not, nereflektování statistických dat již vygenerované části a to, že by se zde určité noty měly vyskytovat častěji než jiné. Obdobná situace platí i o intervalech (některé noty by se měly častěji vyskytovat za jinými). Díky tomu je z hlediska hudební teorie problém určit tóninu, kadence nedává smysl a celkově skladba působí zmateně.

Nicméně použitím vhodných parametrů, zejména výběru not (například pentatonika obsahující primu, sekundu, tercii, kvintu a sextu z libovolné durové stupnice) a velmi krátkých posloupností (maximálně 10 sekund) dala i tato metoda

výsledky, které hodnotím jako poslouchatelné a na kterých by se dala postavit celá skladba. Dobrá byla také jednoduchost jejího používání.

Z důvodu popsaných výše a jejich vážných nedostatků při delších skladbách, bych ji tak označil hodnocením 1 z 5.

4.2 Formální gramatiky

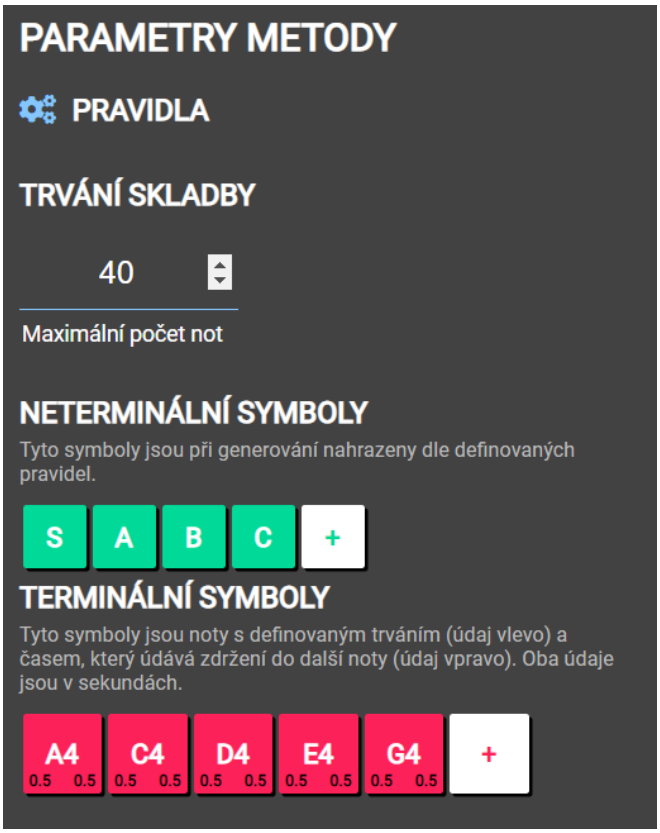
Předchozí metoda trpěla příliš velkou náhodností, ať už se jednalo o umístění not, o jejich délku nebo v jakém pořadí se vyskytují. Tento problém implikuje potřebu větší a lepší kontroly. Tu můžeme dát uživateli pomocí formálních gramatik [20]. Tedy nějakého generativního aparátu, kterým je čtveřice $G = \langle N, \Sigma, P, S \rangle$, kde N je abecedou neterminálních symbolů, Σ je abecedou terminálních symbolů, P je množinou odvozovacích pravidel a $S \in N$ je tzv. *počátečním* respektive *startovním* neterminálem. Gramatiky nám běžně generují řetězce formálních jazyků (tedy nějakou množinu řetězců, které mají určitou společnou vlastnost). Jak je ale využít v hudbě?

4.2.1 Použití v hudbě

Použití gramatik v hudbě je jedna z nejstarších metod generování skladeb [1]. Vychází se zde z myšlenky, kdy se díváme na skladbu jako na posloupnost not nebo akordů. Tento pohled se hodí i u jiných metod. Nabízí se tak velmi jednoduchý způsob. Vzít bezkontextovou gramatiku (při zavedení určitých podmínek je možné použít i jiné gramatiky), jejíž terminální symboly budou představovat noty. Vygenerujeme tak řetězec not, který pak jednoduše převedeme do formátu, který lze přehrát. V takovém případě, ale narazíme na problém monofonního zvuku, kde navíc všechny noty zní stejně dlouho a v písni nejsou žádné pomlky. Problém monofonního zvuku by se dal částečně řešit tím, že by se generovaly přímo akordy, a tak by docházelo ke generování stupňů akordů v nějaké tónině. Ostatní metody, v této práci, však negenerují jen akordové posloupnosti. Díky tomu by zde mohl vzniknout problém v jejich srovnání tak, že by se zvolila pravidla pro generování často používané posloupnosti (viz 2) např. II - V - I, která by se uživatelům více líbila, ale už by pořádně nesplňovala má pravidla kladená na výsledek.

Propracovanější metodou by tak bylo využití gramatik i na jiné aspekty než akordy nebo noty, ale například i na rytmus. Obecně bychom tak měli více gramatik pro jednu skladbu. Jiný, zajímavější způsob využil Jon McCormack [21], který vytvořil virtuálního hráče se stavem a instrukcemi. Vygenerovaný řetězec byl pak sekvenčně čten zleva doprava (podobně jako Turingův stroj čte z pásky). Různé symboly pak měnily stav hráče nebo se dle nich provedl nějaký úkon. Například: tečka řekne hráči ať hraje aktuální notu, + zvětší aktuální notu o půltón, naopak – ji o půltón zmenší. McCormack také využil L-systémy. Ty představují kompaktní způsob reprezentace komplexního vzorce, který má nějaký stupeň repetice, což hudba často má.

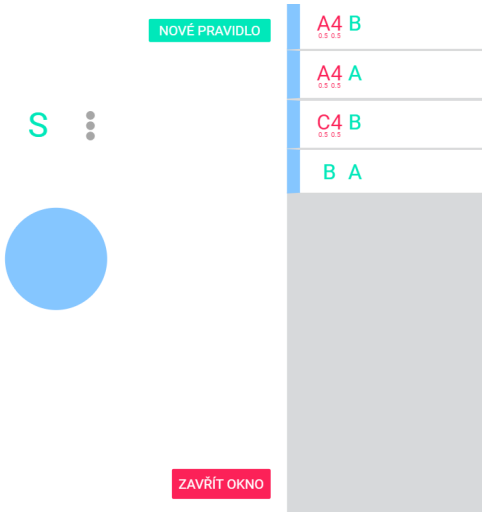
V této práci jsem se rozhodl L-systémy nevyužít, jelikož fungují spíše jako úsporný způsob reprezentace skladby, jejíž části se opakují a jejich použití pro generování různých skladeb při stejné gramatice je dle mého více omezující. Metoda by pak byla více rozdílná od ostatních, v této práci použitých A i proto, aby byla uživatelsky co nejvíce přístupná. Také se mi zdálo nevhodné použití stejných symbolů jako McCormack. Jelikož nechávám uživatele vytvářet vlastní gramatiky, tak si myslím, že by zhoršily jednoduchost použití (ta bude i tak pro neznalého uživatele složitá). Rozhodl jsem se tak k použití gramatik. Takových, kde terminálními symboly jsou noty. Díky tomu jsou jednoduše interpretovatelné uživatelem a převeditelné do formy, kterou lze přehrát nebo zobrazit. Problémy délky noty, její umístění a monofonního zvuku vyřeším tak, že k symbolu bude možné přidat jeho trvání (tak, jak je to i v případě not), ale také prodlevu do další noty, která, když bude nulová, bude umožňovat zahrát akord blokově a nejen jako arpeggio. Je navíc možné zvýšit pravděpodobnost některé noty tím, že ji vložíme víckrát. Díky tomu se více přiblížíme tonálnímu centru tak, jak ho můžeme znát z tonality a při správném využití intervalů i k existenci tóniny (viz 2), kterou můžeme dobře identifikovat. Příklad výchozí gramatiky na obrázku 8.



Obrázek 8: Příklad parametrů metody gramatiky.

4.2.2 Vstupy

Tato metoda je z hlediska uživatelského rozhraní a vstupů nesložitější. Uživatel zde má možnost navolit si terminální a neterminální symboly. Neterminály nesmí být duplicitní a delší než jeden znak, například tedy „X“. Terminály pak musí odpovídat notám s jejich trváním a zdržením do další noty – například „A5 – 0,1 – 0,0“. S těmito symboly pak pracujeme v editoru pravidel (na obrázku 9), kdy se přepínáme do různých stavů (ty odpovídají pravidlům na levé straně) a k nim přiřazujeme výstupy (část pravidla na pravá straně). Mimo to máme možnost zvolit maximální počet not. Pro využití tohoto omezení jsem se rozhodl kvůli zákazu příliš dlouhých (nekonečných) skladeb. Přívlastek „maximální“ je zde proto, že při různých pravidlech nebude vždy moci dosáhnout tohoto limitu.



Obrázek 9: Editor pravidel.

4.2.3 Hodnocení

Jak už bylo řečeno, metoda úzce souvisí se samotnými gramatikami. Tedy její výsledky se pohybují, dle mého, od neposlouchatelných až po příjemné, jednoduché melodie. Obsahuje však řadu problémů. Prvním z nich je potřeba znalosti uživatele zkonstruovat gramatiku, což může být pro lidi bez základů formálních jazyků složité. Tento problém se snažím kompenzovat prací se symboly, které by měly být co nejvíce zřejmé, což je rozdíl od těch u McCormacka. Další kompenzací je grafický editor, který by měl řadu věcí udělat intuitivnější. Druhým problémem je znalost hudební teorie. Pravidla a symboly lze sice zvolit náhodně, ale výsledky jsou pak obecně horší než v metodě Random. Dalším problémem je nerespektování delších posloupností not. Posledním velkým problémem je objemnost gramatiky, kterou je potřeba vytvořit pro aspoň trochu zajímavé výsledky.

Metoda není vhodná pro drobné vyzkoušení a pro kvalitnější výsledek je třeba strávit delší dobu vytvářením gramatiky, která, pokud je navržená špatně, může

generovat písně, které zní pokaždé totožně. Metodu, kvůli mé znalosti základů z hudební teorie a formálních gramatik, hodnotím 1.5 z 5. Bez těchto vědomostí bych ji však ohodnotil 1 z 5.

4.3 Markovovy řetězce

Předešlá metoda není v praxi moc použitelná. Mezi její problémy patří nutnost znalosti formálních gramatik a složitost vytvoření obsáhlejších pravidel. Tyto problémy jsou však řešitelné. Stačilo by, aby pravidla nevytvářeli uživatelé, ale počítač. Odpadly by pak i starosti s potřebou znát hudební teorii nebo formální jazyky. Také bychom se zbavili složitostí s výskytem pravděpodobností různých not. Použití počítače však neřeší poslední hlavní problém - ignorování posloupnosti vedoucí k vybírané notě (metoda řeší pouze přechod z levé strany pravidla na hodnotu na pravé straně). Konstrukce akordů a intervalů by tak byla složitější. Tato potřeba, k problému přistoupit více statisticky, mě přivedla k Markovovým řetězcům.

4.3.1 Řešení

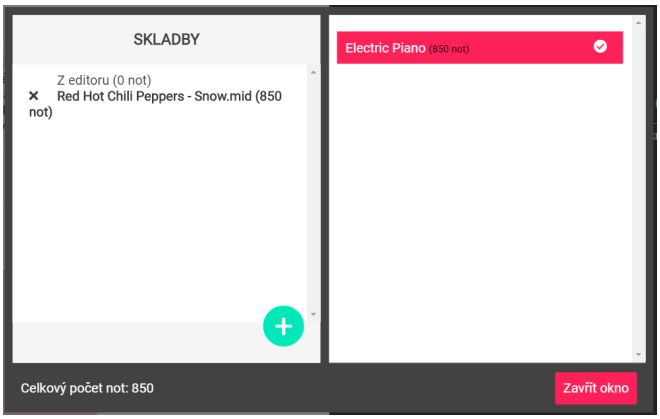
I skladatel se často dostane do situace, kdy se rozhoduje, jaká bude další nota, tedy jaká je pravděpodobnost, že další notou bude například C5, tedy $P(C5)$? Aby nota dávala v rámci své pozice a času výskytu smysl, je potřeba se podívat na notu předchozí. Kdyby to byla nota G5, zajímá nás $P(C5|G5)$. Potřebujeme nějaký sekvenční model, na jehož základě budeme rozhodovat o notách.

Markovovy řetězce [22] jsou jedny z nejjednodušších modelů pro popis chování jevů, které probíhají v čase. V mém případě je použiji pro určení dalších not. Metoda bude mít takzvané stavy, těch bude konečné číslo a budou vycházet z analyzované skladby. Budeme zde tak pracovat systémem náhodných veličin (náhodný proces), kde hodnoty budou nabývat pouze diskrétních hodnot. Mezi stavy se bude dát přecházet. Přechody budou ovlivněny pravděpodobností, jenž získáme taktéž z analyzované skladby. Pokud by pravděpodobnost závisela jen na aktuálním stavu, hovoříme o takzvaných Markovových řetězcích prvního řádu. Již z dřívějších úvah však vyplynulo, že vytvářet skladby na základě jen jednoho intervalu není velmi vhodné. Přibližme si tuto myšlenku na akordech. Je obecně složité nalézt píseň jen o dvou akordech, protože většina používaných kadencí má délku alespoň 3. Využijeme tedy drobnou modifikaci – Markovovy řetězce vyššího řádu. Ty, jednoduše řečeno, při výběru nového stavu počítají s předchozími N stavy.

Pro reprezentaci stavu můžeme zvolit řadu způsobů. Stavem může být tón, interval, délka, akord nebo jakási kombinace všeho. V tomto případě, kvůli potřebě harmonické hudby s různým trváním, zvolím kombinaci tónu, délky a prodlevy do další noty, tedy jakýsi vektor velikosti 3. Problémem této trojice je však její různorodost napříč písní. Například dva tóny C, které by se lišily jen o 0.01 sekundy, by metoda vyhodnotila jako rozdílné stavy, ačkoliv člověk by je vnímal téměř totožně. Z tohoto důvodu jsem umožnil ve vstupu zaokrouhlování.

4.3.2 Vstupy

Metoda, ve srovnání s předešlými, zaujme na první pohled malým počtem vstupů. Nejdůležitější je zde výběr stop, který aktivujeme po kliknutí na „Použité skladby“. V něm je k dispozici, jak je vidět na obrázku 10, výběr stop z editoru nebo importovaných MIDI skladeb. Uživatel tak metodu může použít jak pro vlastní skladby, tak i pro skladby jiných autorů. Mimo výběr not je zde požadovaná velikost výsledku. Ta se udává v maximálním počtu not. Pro využití tohoto omezení jsem rozhodl kvůli zákazu příliš dlouhých skladeb. Přívlastek „maximální“ je zde proto, že při špatném navolení analyzovaných stop a řádu, píseň nebude vždy moci dosáhnout tohoto limitu. Dále zaokrouhlení pro vyřešení problému různorodosti. Posledním vstupem je zde již zmiňovaný řád, ten odpovídá řádu Markovových řetězců z definice. Vstupy je možné vidět na obrázku 11.



Obrázek 10: Vstup použitých skladeb.

Obrázek 11: Vstupy metody Markovových řetězců.

4.3.3 Hodnocení

Na rozdíl od klasických formálních gramatik je tato metoda velmi přívětivá k uživatelům. Nevyžaduje totiž žádné znalosti hudební teorie. Je dostatečně rychlá, a navíc generuje i polyfonní hudbu, kterou obecně, při správně navolených parametrech, hodnotím jako poslouchatelnou. Často je dokonce možné určit tóninu. Není také výjimkou narazit na delší posloupnost intervalů, které v rámci skladby dávají smysl a vedou k nějaké používanější kadenci.

Uživatelské problémy této metody jsou ty, že skladby často obsahují krátké, ale rozpoznatelné části, které se přímo vyskytují v originále. To však může být vhodná vlastnost, pokud uživatel chce jen variaci nějaké skladby. Dále, ačkoliv existuje jistá korelace mezi začátkem skladby a jejího zbytku (v podobě drobných hudebních útržků), není zde větší vztah a obecně nedochází k nějakému rozvinutí skladby. Navíc zde velmi záleží na zvolené hudbě a parametrech. Ty, pokud jsou zvoleny nevhodně, mohou velmi rychle vyústit v neposlouchatelnou hudbu. Rušivý je také občasný výskyt některých not. Ty si však uživatel může odstranit v editoru. Mimo uživatelské problémy je to také složitější implementace samotné metody (v porovnání s předešlými dvěma).

Tato metoda je první, kterou pokládám, dle poměru jednoduchosti a výsledku, za prakticky použitelnou pro delší skladby (nad 1 minutu). Stále však není ideální kvůli výše zmíněným výtkám. Celkově ji hodnotím 2 z 5.

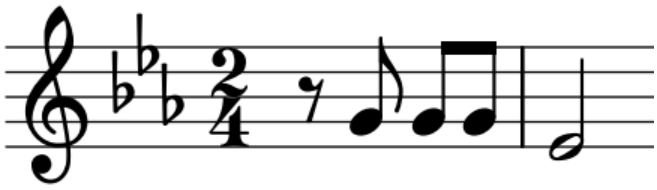
4.4 Random plus

Znalost hudební teorie (v rozsahu sekce 2) byla nápomocná už v metodě formálních gramatik, kde bylo možné si zvolit noty vhodných intervalů a trvání. Stále se však jednalo pouze o jednoduché melodie, jejichž pravidla byla složitá a celá znalost hudební teorie byla na uživateli. Práci s některými částmi hudební teorie by však mohl zvládnout i počítač. Ideou za touto metodou je algoritmická práce s hudební teorií, kompozičními technikami, harmonizací a dalšími. Jelikož jsem nenašel řešení, které by se tímto vším zabývalo, rozhodl jsem se navrhnout zcela vlastní metodu. Jako základní kámen mého řešení jsem zvolil motiv.

4.4.1 Skládání dle motivu

Skladatelé nejsou vždy odkázáni jen na talent, mohou také využít jisté techniky kompozice. Vezměme si například *Prelude no. 1 in C major* z *The Well-Tempered Clavier* od Johanna Sebastiana Bacha. Při poslechu této skladby nebo pohledu na její noty si můžeme všimnout jakési repetice krátké melodie. Tato melodie se během celého díla lehce mění, díky čemuž sice dokážeme identifikovat, že se jedná pořád o stejnou skladbu, ale posluchače nenudí a celé dílo nám obecně zní dobře. Nápadem získaným z této skladby je tak složit krátkou melodii (jeden až dva takty), tu určitým způsobem upravovat a vytvořit na jejím základě celou skladbu. Touto krátkou melodií budeme rozumět tak zvaný motiv [5]. Jako motiv zde budeme brát krátkou rytmickou posloupnost not, které jsou charakteristické

právě svými intervaly nebo rytmem a jsou přítomné napříč celou skladbou. Jejich využití by však nemělo být zcela náhodné, ale mělo by být provázáno logikou, sjednocením a koherencí. Dobrý příklad motivu je známý z úvodu Beethovenovy *Symfonie č. 5 c moll op. 67 „Osudová“*. Jak je vidět na obrázku číslo 12, máme zde klesající „major“ tercii - interval z G na E♭, to je právě onen charakteristický interval, poté máme 3 osminové noty a 1 půlovou, to je zase charakteristický rytmus, který Beethoven pak používá i nadále. Motivem mohou být také jen intervaly nebo jen rytmus.



Obrázek 12: Motiv „Beethovenovy Symfonie č. 5 c moll op. 6“.

4.4.2 Generátor motivu

Prvním krokem při skládání hudby pomocí motivu je samotný motiv vymyslet. Tento proces je schůdný i pro amatéra. Ten může, po nějaké době strávené například s pianem, přijít na náhodnou posloupnost not, která mu zní dobře. V rámci prozkoumání dalšího skládání hudby počítačem jsem se však rozhodl pro vytvoření generátoru motivu. Generátor motivu je postaven nejen na tonalitě, ale i modalitě. Motiv je generován na základě tónu a tóniny. Určuje se zde tak první rozpoložení skladby, tedy jestli bude spíše veselejší nebo smutnější. Je určena její stupnice, což je důležité pro další krok, čímž je modální změna všech not. Získáme pak Ionian, Dorian, Phrygrian, Lydian, Mixolydian, Aelioan nebo Locrian verzi daného motivu se spojením s následným tonálním přístupem. Zmenšíme tím šanci na výsledky, které zní stejně (předpokládá se, že uživatel nebude rozlišovat mezi tóninami). Posledním, hlavním krokem je pak zvolení vhodného rytmu. Ten bude nabývat jen hodnot běžných v hudební teorii, hodnoty by neměly být zcela náhodné jako v metodě Random. Díky tomu by skladba měla získat lepší charakter. Kvůli genetické metodě (viz 4.5) jsou zde hodnoty výběru not ovlivněny hodnotami, ty v této metodě zůstávají náhodné.

4.4.3 Skladba

Skladba bude tedy tvořena motivy, jejich neměnná repetice by však pro posluchače nebyla dostačující. Jak bylo ukázáno již v případě Bacha, je vhodné motiv nějak upravit. Budu zde pracovat s tóninou určenou dle prvního motivu a motivy, které budou postupně vkládány do skladby, budou variacemi motivů předchozích. Variace získáme těmito úpravami (body odpovídají jednotlivým podmínkám v souboru `Motif.js`):

- **Posun v rámci stupnice** – noty se posunou vpravo nebo vlevo, dodrží se však intervaly stupnice, modulace.
- **Otočení pořadí not**
- **Zpřeházení délek trvání not**
- **Spojení stejných not** – pokud budu mít dvě stejné noty za sebou, spojím je do jedné.
- **Rozdělení not na dvě** – opak předchozího bodu.
- **Posun nehledě na stupnici** – noty se posunou o daný počet půltónů bez respektování stupnice.
- **Převrácení skladby dle prostřední noty**
- **Vynechání jedné noty** – nota bude odstraněna.
- **Zamíchání not**
- **Přidání náhodné noty ze stupnice**
- **Nahrazení not náhodnými notami ze stupnice**
- **Přidání pomlk**
- **Změna délky trvání některých not**

Tyto úpravy mohou skladbu jednoduše zkazit, proto je třeba provádět úpravy jenom někdy. To, kdy budou provedeny, určuje náhodnost a šance. Takže, aby se akce provedla, musí být náhodné číslo větší než hodnota šance dané úpravy. Při vhodných číslech tak můžeme zmenšit například šanci na rozdělení noty na dvě nebo posun nerespektující stupnici, což jsou úpravy, které by byly ve větší míře škodlivé. Naopak v malé míře mohou dobře narušit stereotyp. Hodnoty těchto šancí jsou mnou voleny na základě analýzy skladeb a znalosti z hudební teorie. Možnost je změnit a tím se přiblížit lepšímu charakteru nebo konkrétnímu stylu, dávám uživateli v genetické metodě (viz 4.5).

4.4.4 Harmonizace

Samotná skladba vygenerovaná na základě motivu dává lepší výsledky než metoda Random. Jejím problémem je však její jednotvárnost a monofonní zvuk. V rámci zlepšení jejich výsledků a prozkoumání další oblasti generování hudby jsem se rozhodl vytvořit vlastní algoritmus pro harmonizaci hudby. Jeho principem je pak hlavně dodání akordů, které budou fungovat v rámci tónin i tónů samotných. To znamená, že dle aktuálního kontextu (tedy noty a předchozích akordů) budu volit akord ve vhodné inverzi pro získání vhodných kadencí k podpoře tonálního centra. Samotný algoritmus jsem nazval *Music Sense* a nachází

se pak v mé hudební knihovně v souboru `scales.js`. Hlavním vstupem je zde zvolená stupnice (jejíž odpovídající tóninu budeme volit při harmonizaci) a samotná melodie. Pro rozhodnutí, jaký další akord se hodí použít dle aktuálního akordu, využiji jistou formu diagramu. Diagram bude značit možné stupně akordů (kvůli nezávislosti na konkrétní tónině) a jejich možné výsledky. Správné hodnoty pak lze získat pomocí analýzy skladeb nebo využití znalostí hudební teorie [23]. Konkrétně diagramy použité v této práci vychází hlavně z oné hudební teorie, a navíc jsou tam doplněny části, které obecně generovanou skladbu ještě obzvláštní. Příkladem doplnění je využití V akordu z durové tóniny odpovídající následující notě v posloupnosti pro větší důraz. Dále zde také vkládám ne moc často využívaný stupeň „iii“, *sus* akordy a přidávám septimy.

Kroky v zjednodušených bodech (celý kód v souboru `scales.js`):

1. Nastavení parametrů tóniny (akordů, intervalů a diagramů).
2. Přiřazení možných akordů k notám (akord se přiřadí k notě, jestliže existuje v dané tónině a leží v něm ona nota).
3. Vložení prvního akordu (dávám zde přednost tónice z důvodu tonality, jinak volím náhodně).
4. Filtrace možných akordů dle aktuálního stavu (posledního použitého akordu) a diagramu.
5. Výběr filtrovaných akordů (je zde omezení na počet akordů, kvůli obecné popularitě v písni využívat méně typů akordů, například v POP music jsou často pouze 4 různé akordy).
6. Jestliže nebyl vybrán žádný akord, zkusíme aplikovat výše zmíněný V akord z durové stupnice odpovídající následující notě v posloupnosti, případně „medianty“ anebo „sus akordy“.
7. Posledním hlavním bodem je případné přidání septimy k akordu.

4.4.5 Vstupy

Jedním z cílů této metody byla také jednoduchost, aby ji dokázal používat i uživatel, který nemá žádné znalosti hudební teorie nebo soubory, které by šly analyzovat. Metoda je tak svým používání dokonce jednodušší než Markovovy řetězce. Jak je vidět na obrázku číslo 13 jediným potřebným vstupem je číslo kolikrát se má motiv opakovat. Dalším, v tomto případě volitelným, vstupem je motiv samotný. Ten si uživatel může zvolit náhodný nebo použít vlastní, importovaný z editoru.

Obrázek 13: Vstupy metody Random plus.

4.4.6 Hodnocení

Tato metoda, na rozdíl od předešlých zde uvedených, je zcela navrhnuta mnou s velkým využitím znalostí hudební teorie na všech úrovních. Bylo potřeba vyrovnat poslouchatelnost, jednoduchost, univerzálnost a zároveň také dostatek charakteru výsledků. To se částečně povedlo. Random plus generuje, dle mého, v průměru nejvíce poslouchatelných skladeb z ostatních, zde již uvedených metod. Používání je dokonce nejsnadnější ze všech metod v této práci. Univerzálnost je však, v porovnání s ostatními, průměrná. Obecně totiž generuje skladby podobného druhu. Ty zpravidla obsahují zhruba 4 akordy a opakující se strukturu, která je různě modifikována. Většinou je však možné během přehrávání skladby rozpoznat výsledný motiv a skladby tak (na rozdíl od jiných metod) působí celistvě. Díky tomu však obecně postrádají silnější a zajímavější charakter. Tento popis však sedí i na řadu písní složených člověkem.

Random plus je tak spíše vhodné použít pro získání nápadů (díky generátoru motivů), pro další dodělání skladby nebo jednoduchou podkresovou hudbu. Metoda je uživatelsky velmi jednoduchá, většinou produkuje poslouchatelné výsledky a poskytuje dobrý základ pro další práci. Skladby ovšem působí jednoduše, jsou si jistým způsobem podobné a chybí jim větší charakter. Díky tomu však působí celistvě. Metodu proto hodnotím 3.2 z 5.

4.5 Genetické algoritmy

Metoda Random plus využívá silně širokou škálu hudební teorie, a to jak ve formě tonality, tak i modality. Tímto stylem dostáváme různé, relativně dobré skladby. Její univerzálnost je však vykoupena absencí silnějšího charakteru a nemožností generovat skladby určitého typu neboli žánru. Metodu by bylo možné modifikovat tak, aby generovala například jazzové písně nebo naopak rockové. Tím by ale došlo k omezení, které nemá žádná z ostatních metod. To znamená, že metoda by byla od začátku vytvořena pro konkrétní žánr. V rámci objektivnějšího hodnocení jednotlivých metod a prozkoumání více řešení, jsem tedy od této varianty upustil.

Jak jsem již zmínil, dalším problémem je to, že obsáhlost Random plus může být občas nechtěná. Uživatel může chtít pouze jednoduchou skladbu (zmenšit počet akordů a opakující se motiv bez větších změn) nebo skladbu nejsložitější. Sílu Random plus je pak třeba řídit tak, aby byla co nejvíce podobná tomu, jak si to uživatel představuje. Kvůli těmto problémům jsem se rozhodl využít a navrhnout nové řešení pomocí, v generování hudbě oblíbených, genetických algoritmů.

4.5.1 Definice

Kořeny genetických algoritmů [24] vychází z Darwinovy myšlenky (z druhé poloviny 19. století), že populace živočichů se vyvíjela po mnoho generací dle principu přirozeného výběru. Zhruba o století později, se tato myšlenka dostala do světa počítačů. V dnešní době mají genetické algoritmy široké využití a jsou využívány pro řešení problémů jak vědci, tak i praktickými specialisty. Pod pojmem evoluční algoritmus se běžně řadí pojmy jako genetické algoritmy, evoluční strategie, evoluční programování a genetické programování. Obecně však v této práci budu přistupovat k pojmu genetický algoritmus tak, jak jej navrhl Johna Holland [25].

Budeme mít nějakou populaci, jedinci se v ní budou snažit přežít a reprodukovat se. Čím úspěšnější budou, tím lepší a větší šanci budou mít na potomka. Díky tomu vlastnosti schopnějších, či lépe přizpůsobivých jedinců, zakódované v genové výbavě, se předají dalším generacím. Budeme tak pracovat s generacemi populací, kde úspěšnost jedinců bude měřena dle uživatelského hodnocení. Toto hodnocení bude kvalitativně vyjadřovat míru vlastností konkrétního jedince vzhledem k řešenému problému a současně bude i měřítkem reprodukční způsobilosti tohoto jedince. Použitím simulace náhodného procesu přirozeného výběru spolu s geneticky inspirovanými operátory křížení a mutace, nám dá možnost vytvoření nové generace, která by měla přinést lepší výsledky. Jestliže tento proces opakujeme po generacích, geny úspěšnějších se budou kombinovat a my dosáhneme požadovaného výsledku. V rámci genetiky je důležité zmínit dva pojmy: genotyp (je soubor všech genetických informací jedince, v mém případě uložena v DNA) a fenotyp (je soubor všech znaků a vlastností jedince) [26].

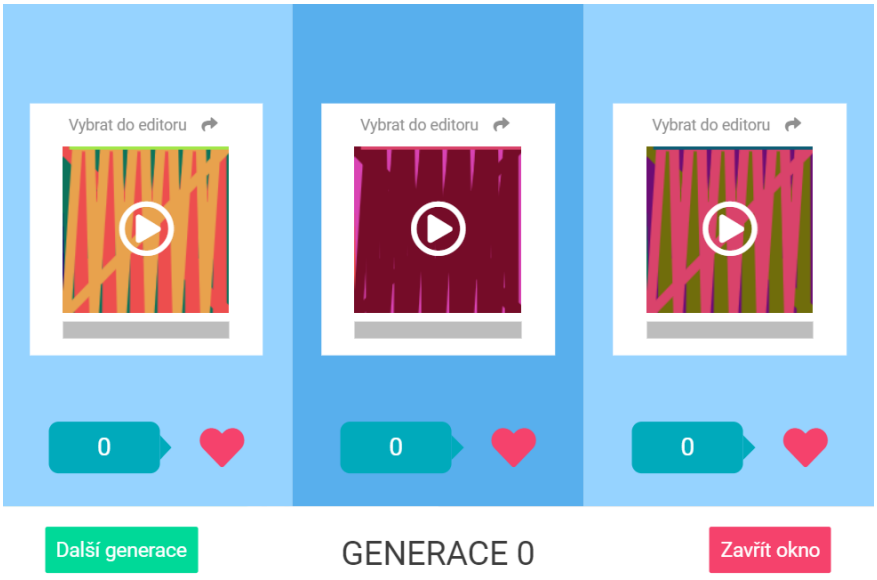
4.5.2 Použití při generování hudby

Genetické algoritmy u generování hudby je možno navázat na celou řadu funkcí. Můžeme na jejich základě vybírat noty, rytmy, motivy nebo celé skladby. Jelikož každá zde použitá metoda generuje skladbu, zvolil jsem tu poslední. Jelikož neexistuje určitý druh řešení (jako tomu například bylo u Markovových řetezců) a i když je cílem stejný výsledek, tak se konkrétní reprezentace fenotypů a genotypů liší, rozhodl jsem se genetický algoritmus navázat na metodu Random plus ze sekce 4.4. Už teď jsou jejím základem obecné úpravy motivů a harmonizace, které jsou však dané. Přidáním genetických algoritmů by její výsledky měly jít zlepšit.

4.5.3 Řešení

Celé řešení staví na návrhu metody Random Plus. Ta v mnoha případech měnila skladby dle toho, zda byly náhodné hodnoty větší než některá čísla. Tyto náhodné hodnoty ovlivňovaly všechny změny motivů (jak často bude docházet k těmto změnám), počty akordů, pocit skladby (jestli bude spíše durová – veselá nebo naopak mollová - smutnější), dále to, jestli to bude jednodušší, spíše popová skladba o malém počtu akordů nebo naopak složitá a jestli se zde budeme blížit spíše žánru jazz pomocí využití septim v akordech a sus akordů. Všechna tato čísla jsou pak uložena v genech jedinců (genotyp), dle nich dostaneme výsledek, který je možné přehrát (fenotyp). Jelikož výsledek zde částečně stojí na náhodě i silná hodnota genotypu se nemusí v jedinci projevit (ačkoliv je větší šance, že se projeví), tato hodnota, společně s dalšími, se pak ale přenese do následující generace (za předpokladu dostatečné hodnoty fitness funkce a nezměnění pomocí mutace a křížení). Obecně se tak napříč generacemi vyfiltrují chtěné vlastnosti.

Jak jsem výše zmínil, to jaké geny skladeb se budou dostávat do dalších populací (v podobě nové generace) pak určí samotná fitness funkce. Fitness funkci (tedy jak je daný jedinec úspěšný) bude provádět člověk z důvodu uvedených u problémů generování hudby v sekci 1.2.1. Bude tak moci vygenerovat hudbu konkrétnějšího druhu. Hodnotit se bude dle poslechu formou bodů, jak je vidět na obrázku 14.

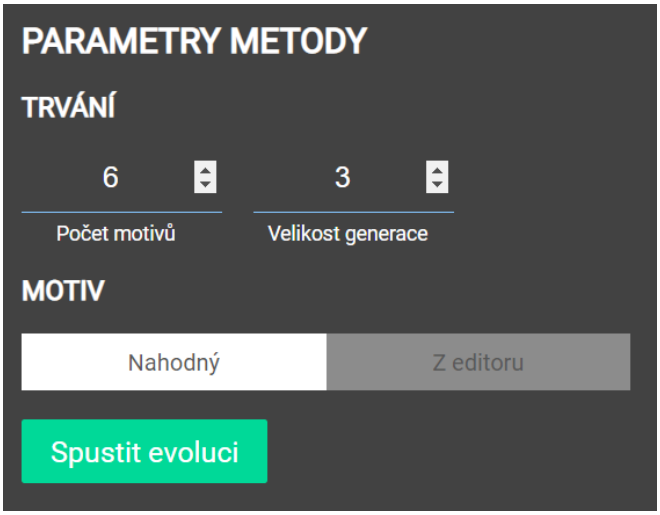


Obrázek 14: Okno pro hodnocení jedinců.

4.5.4 Vstupy

Vstupy, které je možné vidět na obrázku 15, jsou totožné s metodou Random Plus. Nově je tady však vstup pro velikost generace. Na základě velikosti generace

a otevřením okna hodnocení (tlačítkem „Spustit evoluci“) je vygenerován počet skladeb, které si uživatel přehraje a ohodnotí kliknutím na ikonu srdce a proces kliknutím na „Další generace“ opakuje, dokud není spokojen s výsledky.



Obrázek 15: Vstupy metody genetického algoritmu.

4.5.5 Hodnocení

Metoda, díky svému základu, se v několika bodech blíží Random plus. Z hlediska uživatelského používání je podobně snadná. Situaci komplikuje akorát hodnocení jednotlivých generací, které ale probíhá jednoduše a intuitivně. Problémem je však samotný proces hodnocení. Ten je třeba provádět relativně dlouhou dobu (nad 50 generací) než začne generovat chtěné typy skladeb. Navíc, díky počátečním náhodným genům, dává „nevyvinutý“ genetický algoritmus zpravidla horší výsledky než samotná metoda Random plus. Po dosažení dostatečného množství generací je však jeho funkčnost lepší než u všech předchozích metod a generuje skladby bližší určitému typu s lepším charakterem. Navíc si uživatel při hodnocení poslechne velké množství vygenerovaných skladeb a je tak větší šance, že narazí na nějakou, co mu bude vyhovovat a kterou použije.

Vzhledem k trvání doby přípravy generací a horších výsledků bez nich hodnotím tuto metodu 3.3 z 5 s přihlédnutím k její uživatelské jednoduchosti a síle při vhodných genech.

4.6 Neuronové sítě

Jako jedno z prvních řešení generování hudby se v dnešní době nabízí neuronové sítě. Důvodem je jejich statistická síla a alespoň částečná podobnost člověku. Skutečně se jedná o populární způsob, který se, v nějaké formě, poměrně často používá. Svědčí o tom i fakt, že sám Google se zaměřil na technologii, která se

přímo na umění a generování hudby zaměřuje[4]. Možností, jak využít neuronové sítě, je hned celá řada.

4.6.1 Obecný popis

Základ neuronových sítí [27] je obecně napříč jejich variacemi podobný. Vychází se zde z biologického neuronu. Ten ve velmi zjednodušené verzi funguje následovně. Skládá se ze tří hlavních částí – dendritů (vstupní přenosový kanál), buněčného jádra a axonu (výstupní přenosový kanál). Skrze neuron pak putuje signál, který se přenáší na další pomocí synapsí. Takových neuronů má lidský mozek ohromné množství, společně vytváří neuronovou síť a jejich spojení se při učení časem mění. Z toho byl vytvořen neuron umělý.

V zjednodušené, základní verzi, se umělý neboli formální neuron skládá z n vstupních hran x_i, ty modelují dendrity a určují vstupní vektor. Každá z těchto hran je charakterizována váhou w_i, která představuje, jak se daný vstup podílí na výstupu (vstupní signál může i tlumit). Dále zde probíhá výpočet vážené sumy, k ní je připočten práh (pro lepší výsledek) a výsledek se použije jako vstup aktivační funkce, ta tak rozhoduje o finálním výstupu. Aktivační (přenosová) funkce by měla být jednoduše derivovatelná, aby bylo možné jednoduše vypočítat gradient k optimalizaci učení neuronové sítě. Používá se také pro stlačení hodnot do určitého intervalu. Jednoduchým příkladem aktivační funkce je funkce sigmoid [28].

Výpočetní schopnosti samotného neuronu nejsou pro většinu problémů dostačující. Proto se neurony propojují do sítí, kde pracují jako celek s cílem vyřešit specifický problém. Sítě pak slouží k modelování určité funkce, u které zpravidla známe pouze její vstupy a výstupy. Jedním z nejjednodušších druhů je pak dopředná neuronová síť. Ta se skládá ze vstupní a výstupní vrstvy. Obecně je však potřeba přidat i takzvané skryté vrstvy, ty jsou umístěny mezi vstupní a výstupní vrstvou a používají se pro zachycení složitějších významů. Jednotlivé vrstvy jsou spolu propojené tak, že výstupy předchozí vrstvy představují vstupy vrstvy následující. Tak se pokračuje až do výstupní vrstvy, kde výsledný signál představuje odpověď sítě na vstup. Pro zachycení neuronové sítě se používá orientovaný graf. Abychom však mohli neuronové sítě používat je obecně nutné, aby prošly procesem učení.

4.6.2 Učení

Zjednodušeně řečeno, kromě struktury neuronové sítě určuje její chování i to, jaké mají váhy a prahy jednotlivé neurony [28]. Učení pak představuje proces, kdy upravujeme tyto hodnoty takovým způsobem, aby výstup odpovídal požadovaným datům. Zajímá nás hodnota udávající rozdíl získaných a požadovaných. K získání této hodnoty se používá ztrátová neboli chybová funkce. To implikuje, že cílem učení je tuto funkci minimalizovat. U učení často narazíme na algoritmy

[4] https://magenta.tensorflow.org/

využívající klesání podle gradientu. K učení je však možné použít i evoluční metody. Obecně základním principem při gradientu je rozdělení do epoch (doba za kterou se síť setkala s každým vzorkem z datasetu alespoň jednou), ty probíhají iterativně a jejich cílem je aktualizace vah. U klesání podle gradientu dojde ke spočítání parciálních derivací chybové funkce, ta určí, jak moc se jednotlivé váhy podílejí na výstupu a jak moc se mají změnit, abychom je přiblížili požadované hodnotě. Tak se po krocích postupně dostaneme až do lokálního minima.

Pro stanovení toho, jak se má hodnota rychle měnit, slouží globální parametr učení α. Zde je potřeba zvolit přiměřenou hodnotu. Pokud je α nízké, učení bude pomalé, avšak na druhou stranu, pokud bude α příliš vysoké, nebude konvergovat. Předešlý odstavec implikuje existenci „správných“ dat, podle kterých se budou upravovat váhy. To, jakým způsobem tyto data získáme záleží na druhu učení. Jedním z nejpoužívanějších typů je „Učení s učitelem“, je možné však využít i „Učení bez učitele“ a další.

„Učení s učitelem“ známé jako „Supervised learning“. Využívá sadu trénovacích dat, kde jsou obsaženy vstupy i požadované výstupy. Síť předpovídá z daných vstupů a učení pak probíhá tím, že se síť snaží minimalizovat rozdíly mezi hodnotami. Učení zpravidla končí, když je velikost chyby menší než určitý práh nebo když se rozdíl v několik epochách nezmenšil. Jestliže má síť znalost těchto zkušebních dat je možno očekávat, že v ideálním případě se dokáže vypořádat i s daty podobného typu, které již označené nejsou. Je však nutné dát si pozor, aby nedošlo k takzvanému přeučení. Síť by sice fungovala velmi dobře na trénovacích datech, ale na jiných datech již ne. Je potřeba jistého zobecnění. Přeučení lze zabránit například velkým počtem dat v datasetu nebo použitím některé z regularizačních metod [28] – například **Dropout**, kde se vynechávají poměrové části neuronů na každém trénovacím vstupu. Netvoří se tak skupiny neuronů a každý neuron se učí rozpoznávat příznaky nezávisle. Díky tomu je i model více odolný vůči ztrátě částí informace.

To, jaký způsob učení zvolíme závisí nejen na našich datech, ale i našem cíli. V dříve zmíněných dopředných neuronových sítí se při učení používá algoritmus backpropagation [27], který se používá v rámci optimalizace k minimalizaci hodnot. Můžeme jej využít u různých neuronových sítí. Různé neuronové sítě však nelze využít u všech problémů. Hovoříme o takzvaných architekturách neuronových sítí.

4.6.3 Rekurentní neuronová síť

Při poslechu písní můžeme obecně cítit jakousi celistvost. Skladbu vnímáme v rámci nějakého časového kontextu a každou poslechnutou sekundu vnímáme vzhledem k těm předešlým. Můžeme vnímat vývoj skladby ať už z hlediska začátku, prostředku a konce nebo jen konce fráze. Je zde jakási perzistence. Tradiční neuronové sítě nám toto však neumožní. Pokud bychom například měli k dispozici jen jednu notu, nebylo by možné ji dobře zanalyzovat – jestli sedí v rámci celé skladby.

Rekurentní neuronové sítě [29] tento problém řeší. Jednoduše se dají představit jako sítě, které uvnitř mají cykly, které umožňují informaci udržet. Nebo naopak několik kopií klasických neuronových sítí, kde každá předává data svému následníkovi. Toto spojení je pak vhodné pro různé sekvence – jako je text nebo právě hudba. Při používání těchto sítí se však narazilo na problém. Ačkoliv si rekurentní sítě dokážou pamatovat, jejich problémem je dlouhodobé uchování. Narážíme zde na tzv. *vanishing* a *exploding* gradient, který vzniká u metod založených na klesání gradinetu, kdy dochází k jeho „mizení“ nebo naopak „explozi“. Jinými slovy, posledních několik not by bylo v pořádku, problémem by však byly starší takty. V řadě případů se ale nemusí jednat o věc, která by tyto sítě vyřadila. V mém případě bych, ale kvůli lepší celistvosti skladby (aspoň částečně podobnou té v metodě Random plus), tyto sítě použít nemohl. Naštěstí existuje jejich varianta, která tento problém relativně dobře řeší.

4.6.4 LSTM

Long Short Term Memory [30] sítě jsou speciálním druhem RNN, které jsou vhodné, jak už název vypovídá, na učení dlouhodobě trvajících informací. Byly představeny v roce 1997 Hochreiterem a Schmidhuberem a dále byly použity a rozvinuty mnoha lidmi. Hodí se na celou řadu problémů a jsou hojně využívány.

LSTM jsou vybaveny speciálním uzávěrovým mechanismem, který ovládá přístup k vnitřnímu stavu buňky. LSTM mají schopnost přidávat nebo odstraňovat informace danému stavu buňky pomocí struktur, kterým se říká brány (často narazíme na anglické výraz *gate*). Díky tomu, že brány mohou bránit zbytku sítě, aby upravoval obsah vnitřního stavu buňky, při propagování chyby gradient jednoduše nezmizí (ve srovnání s obyčejnými rekurentními sítěmi).

4.6.5 Reprezentace

Mimo to, jaký typ a architekturu sítě zvolíme, záleží velmi i na tom, v jakém formátu ji poskytneme data. Tedy jak budou data reprezentována a jak budou data kódována. Již v předešlých částech jsme k hudbě přistupovali jako k posloupnosti not. Jako první reprezentace se tak nabízí vzít přímo sekvenci. Například posloupnost not „C3, D3, E3, F3, G3, G3, A3, A3, G3“ jednoduše namapovat na čísla „0, 1, 2, 3, 4, 5, 5, 6, 6, 5“ a ty pak využít jako vstup sítě.

Přístup, který odpovídá práci s textem na bázi písmen, je ovšem nevhodný. Mimo názvů, noty v posloupnosti hudby obsahují také různá časová trvání a mohou začít v rozdílné době. Je tedy možné, že ve skladbě bude na chvíli úplné ticho nebo naopak v jeden moment bude znít tónů hned několik. Jedním z řešení by bylo generovat pouze monofonní melodie s přesně daným trváním not. V praxi by to však této metodě uškodilo, jelikož předešlé metody toto omezení neměly a v hudbě se obecně moc často nesetkáváme s touto jednotvárností. Ačkoliv se tak jedná o nejjednodušší způsob, je až příliš omezující. Absence harmonie by se dala vyřešit tak, že bychom kromě not pracovali i s celými akordy, které bychom pak namapovali na jedno z čísel, čímž by se nemuselo nic měnit. Akordů a jejich

variací (inverzí, přidání septimy atd.) je však velké množství, což by zvětšilo dataset o jednotlivé akordy, které by byly závislé přímo na konkrétních notách a v konkrétním pořadí.

Vzniklo několik způsobů jak zmíněné problémy řešit. Například jedním přístupem [33] bylo rozdělení celé skladby na N vektorů o velikosti 89 dle nějaké frekvence. Prvních 88 bitů vektoru odpovídalo 88 klávesám piana v daný moment, kde 1 odpovídaly aktuálně stisknutým klávesám a 0 pak těm ostatním. Poslední bit představoval počet opakování vektoru. Tento přístup vyřešil řadu problémů s různým časovým rozmístění not a s jejich harmonií. Při zkoušení této metody jsem však narazil na větší velikost datasetu, které jsem se, z důvodu webové aplikace, chtěl vyhnout a také několika dalším problémům.

Rozhodl jsem se vytvořit vlastní reprezentaci, která částečně čerpá z těch předchozích. Skladbu si rozdělím na stavy (dle zvolené frekvence), v nich budu hledat začátky jednotlivých not, jejich trvání a konce. Když nota začne, vložím do výsledného řetězce zX, kde z značí, že se jedná o začátek noty a místo X je číslo odpovídající noty. V dalších stavech přidám řetězec značící čekání (například *p10*), ten se bude kumulovat dle stavu jednotlivých not, díky čemuž zvládne harmonický zvuk, různé trvání noty a výsledek nebude složen jen ze stavů existující písně. Část možné reprezentace je vidět na obrázku. 16.

```
p9 p8 z7 kz7 z16 kz16 z21 kz21 z26 kz26 z31 kz31 p8 z16
kz28 p3 z14 z21 z23 z26 p7 kz21 kz23 kz26 p1 kz14 p4 z16
kz12 kz28 p1 z23 kz23 p9 z7 p1 kz7 z17 kz17 z23 kz23 z26
kz32 p4 z14 z21 z26 z30 p1 kz14 kz21 kz26 kz30 p9 p8 z11
z18 z23 z27 kz27 z33 p1 kz18 kz23 z28 kz28 kz33 p3 z11 z
z25 z34 p1 kz25 kz34 p1 z13 kz13 z14 kz14 p1 z24 z33 p1
```

Obrázek 16: Příklad reprezentace.

4.6.6 Dataset

Jak vyplynulo z předchozích sekcí, síť se bude učit z MIDI souborů, které budou převedeny do požadovaného formátu. V rámci zvětšení množství hudby daného stylu provedu proces transpozice existujících písní. Využiji tak procesu, který například provedla Christine Payne [34] se svým datasetem. Ta ho rovněž rozdělila na části dle frekvencí, získala stavy piana, transponovala je a pak skladbu převedla do jejího formátu který podobně jako můj, zachycuje začátky a konce kláves. Navíc však umožňuje využití více nástrojů a větší možnosti s frekvencemi.

4.6.7 Implementace

Pro konkrétní implementaci jsem zvolil technologii Tensorflow (viz 5.7) kvůli její práci na desktopu i na webu, optimalizací grafickou kartou a jednodušší práci. Ta se společně s Keras API stará o konkrétní implementace algoritmů souvisejícími

se sítěmi. V tomto případě, jak je vidět v kódu 1, tak zpracuje přímo daný návrh sítě. K němu jsem přistupoval dle nutných kroků z hlediska využití datasetu, podobných sítí, experimentování a omezení síly použitého HW:

```
1     m = Sequential()
2     m.add(LSTM(
3         512,
4         input_shape=(network_input.shape[1], network_input.shape[2]),
5         recurrent_dropout=0.3,
6         return_sequences=True
7     ))
8     m.add(LSTM(512, return_sequences=True, recurrent_dropout=0.3,))
9     m.add(LSTM(512))
10    m.add(Dropout(0.2))
11    m.add(Dense(256))
12    m.add(Activation('relu'))
13    m.add(BatchNorm())
14    m.add(Dropout(0.2))
15    m.add(Dense(n_vocab))
16    m.add(Activation('softmax'))
17    m.compile(loss='categorical_crossentropy',optimizer='rmsprop')
```

Zdrojový kód 1: Část kódu pro vytvoření sítě.

- **LSTM** – tato volba vychází z předešlé části textu. Pro konkrétní počet jsem se rozhodl kvůli kompromisu výsledků a složitosti učení. Jedna se ukázala jako nedostačující, naopak čtyři nebylo možné rychle vytrénovat.
- **Dropout** – ačkoliv díky reprezentaci dochází k menšímu přeučení, stále se v datech objevují občasné stejné vzory. Pro ještě lepší výsledky využiji této, již zmíněné, regularizační metody.
- **Dense** – „standardní propojená“ vrstva pro další práci sítě.
- **Aktivace** – pro aktivaci zde využívám `Activation(relu)`, ta aplikuje „rectified linear unit activation“ funkci, která obecně funguje lépe než zmiňovaný sigmoid a řada dalších [28]. Navíc se jedná o častou volbu pro tyto sítě. Jako finální aktivaci zde volím softmax (výsledek je normalizován do pravděpodobností distribuce) kvůli konci sítě s *categorical crossentropy* [31].
- **BatchNorm** – typ vrstvy, kterou Keras nabízí. Obecně poskytuje a rychlejší učení a větší přesnost.

Síť je pak kompilována se ztrátovou funkcí (cílem ztrátové funkce je výpočet, který by se měl model během tréninku snažit minimalizovat) *categorical crossentropy* (kvůli povaze výsledku, kde výsledek bude moct patřit do jedné kategorie)

a pro zrychlení pak *RMSprop* (algoritmus pro zrychlení neuronových sítí) [32], který na tento problém funguje relativně dobře. Samotná práce se vstupy a výstupy pak probíhá velmi jednoduše. Pro celý dataset si připravím sekvence délky 101 znaků (ty jsou následně namapovány na čísla). Jednoduše řečeno, všechny znaky, kromě posledního, představují vstup do sítě a poslední pak výsledek sítě při tomto vstupu. Jednoduchým způsobem tak mohu využít zmíněné učení s učitelem.

4.6.8 Vstupy

Použití metody je snadné. Jediným vstupem je zde počet elementů (hodnot vzniklých kódováním) a je zde také potřeba načíst potřebný dataset kliknutím na tlačítko dataset. Tato akce může nějakou dobu trvat podobně, jako samotné generování.

Obrázek 17: Příklad vstupů metody neuronové sítě.

4.6.9 Hodnocení

Metoda byla zdaleka nejsložitější na návrh i implementaci. Jelikož existuje celá řada způsobů, jak přistupovat k využití neuronových sítí při generování hudby, bylo potřeba zvolit řadu věcí. Volil jsem druh sítě, její návrh, ale i samotnou reprezentaci dat, jenž by, na rozdíl od některých jiných reprezentací, pracovala i s časovými údaji not a zároveň byla velikostí co nejmenší. Další problém byl v samotném učení, které při mé reprezentaci a mém HW trvalo zhruba 30 hodin, než začalo poskytovat zajímavé výsledky. Složité bylo také napojení na webovou aplikaci, kdy bylo potřeba vyřešit nejen generování zcela nové písně pomocí naučeného modelu, ale i samotné načtení modelu a potřebných dat, které bylo neefektivní získávat při každém zapnutí aplikace. Metoda pak generuje skladby podobné skladbám, na kterých se učila (dle mého však nedochází k příliš velkému

přeučení) a uživatel tak nemá možnost vygenerovat si píseň vlastního stylu. Poslední problém, který je problémem i z uživatelského hlediska, je pomalost metody. Určitý čas může trvat, než se načtou potřebná data a dále je časově náročnější i samotná generace, která, kdybych je neprováděl pomocí web workeru (viz 6.3), by mohla zablokovat celou aplikaci. Dle mého však tato omezení nejsou tak závažná, že by metoda nebyla použitelná v praxi. Navíc poskytuje objektivně nejlepší výsledky, a to i bez uživatelovy znalosti hudební teorie. Díky využití LSTM sítí je, na rozdíl od klasických neuronových sítí, ve většině případů cítit aspoň malá návaznost a pocit celistvosti, což je vlastnost, která se nevyskytovala v žádné jiné metodě, tedy až na moji metodu Random Plus (tedy i její genetickou verzi), kde však chyběl větší charakter, který u LSTM je.

Složitost a problémy metody jsou vykoupeny jejími výsledky, a proto ji hodnotím 4 z 5.

4.7 Uživatelské hodnocení

Hodnocení hudby je často velmi subjektivní. A jak jsem zmínil v problémech generování hudby v sekci 1.2.1, nezáleží jen na subjektivním hudebním vkusu jedince, ale také na jeho momentálním stavu, tedy jestli je například unavený. Proto jsem se, mimo mé hodnocení, rozhodl dát možnost hodnotit přímo všem uživatelům. V rámci co nejpřesnějšího hodnocení jsem upustil od toho, že bych písně vybíral já, nebo že bych jednotlivcům za sebou pouštěl velké množství písní. Díky tomu, že uživatel metody hodnotí přirozeně při používání aplikace a není k hodnocení nucen přistupovat jako k nějakému úkolu, předpokládám rozumnější výsledky, které budou reflektovat různý hudební styl uživatelů, a to bez únavy, která je jedním z hlavních problémů uživatelského hodnocení. Tato hodnotící metoda by tak měla překonat všechny ostatní.

Zde zaznamenané hodnocení probíhalo anonymně po dobu jednoho měsíce (je v něm zahrnuto i mé hodnocení). Toto hodnocení je zobrazeno i u jednotlivých metod pro lepší uživatelskou představu možných výsledků, takže v aplikaci má i praktickou funkci.

Název metody	**Počet hodnocení**	**Hodnocení (1 až 5)**
Random	81	1.8
Formální gramatiky	61	1.9
Markovovy řetězce	78	2.4
Random plus	86	3.2
Genetické algoritmy	55	3.4
Neuronové sítě	69	4.1

Tabulka 2: Uživatelské hodnocení.

5 Použité technologie

V práci používám celou řadu technologií. Výběr těch správných byl klíčový, jelikož v práci řeším problémy na desktopu, webu, ale i na serveru. V některých oblastech byla na výběr celá řada možností, naopak jinde jsem narazil na různá omezení. V této části představím mnou použité technologie a důvody, které jsem měl pro jejich zvolení.

5.1 JavaScript

JavaScript [44] je dynamicky typovaný, interpretovaný, objektově orientovaný jazyk. Používá se na webu pro ovládání interaktivních částí stránek. V dnešní době jsou jeho možnosti daleko větší - využití nalezne i při komunikaci se serverem nebo na serveru samotném (viz 5.4). Podpora všech hlavních prohlížečů dělá z JavaScriptu v podstatě nutnost pro vytvoření webové aplikace, jenž jsem se rozhodl udělat.

Pro JavaScript je typické, že na rozdíl od typických objektově orientovaných jazyků, které používají třídy, JavaScript využívá prototypy. Třídy nicméně obsahuje, byly však představeny až ve verzi ECMAScript[5] 2015 [45] a slouží spíše jenom jako syntaktický cukr[6]. Jazyk ale nabízí možnost nejen objektového paradigma, ale i imperativní nebo funkcionální, které jsem v práci také zúročil.

5.2 HTML 5

HyperText Markup Language [46] nebo-li HTML. Je to značkovací jazyk používaný pro definování struktury, ale také sémantiky webových stránek. Zdrojový kód s příponou html se skládá ze značek, takzvaných tagů. Těmto tagům můžeme přiřadit atributy, ty umožňují celou řadu věcí, v mém případě hlavně zjednodušují práci JavaScriptu a CSS (více 5.3). Je interpretovatelný webovými prohlížeči a o jeho standardy se stará W3C[7]. HTML 5 je pak hlavní označení pro nejnovější verze.

Podobně jako JavaScript, je HyperText Markup Language pro vývoj webových stránek klíčový a ačkoliv web převážně generuji pomocí technologie Vue.js, i ta je však na HTML závislá, například v případě šablon, viz kód 2.

5.3 CSS 3

Cascading Style Sheets [47] nebo-li CSS. Jde o jazyk, který popisuje, jak jsou HTML elementy zobrazeny uživateli, takzvaně „ostylovány“. Jednotlivé styly zapisujeme pomocí CSS pravidel. Skládají se ze selektoru určující HTML elementy, vlastnosti a hodnoty. V této práci používám CSS nejen napojením souboru s příponou css do HTML, ale také u Vue.js, kde tvoří jednu ze tří částí komponenty.

[5] JavaScript je implementací ECMAScriptu.
[6] Část syntaxe programovacího jazyka, jejíž účelem je usnadnit zápis.
[7] Konsorcium W3C (World Wide Web Consorcium) https://www.w3.org/.

Stylování pro moji aplikaci bylo klíčové, ačkoliv řada stránek, které jsou založené hlavně na obsahu (například internetové encyklopedie), mohou být funkční i bez stylu. Má aplikace, jakožto webová aplikace pro práci s hudbou, je na stylech závislá, jelikož jednotlivé prvky musí zobrazovat informace o MIDI souborech, jejich vlastnostech, ale i chování samotného přehrávání. CSS je tak pro mě stejně nezbytné jako předešlé technologie.

5.4 Node.js

Node.js [48] je asynchronní, událostmi řízené běhové prostředí pro JavaScript, využívající, v C++ napsaný, V8 engine od Googlu. Ten využívá například i Google Chrome. Node.js se používá na serveru. Tam vyniká tím, že při psaní kódu Node.js používá pouze jedno vlákno se smyčkou událostí [49]. Ta umožňuje provádět neblokující I/O operace. Tyto vlastnosti dovolují v Node.js tvořit škálovatelné systémy. Prostředí Node.js navíc umožňuje využívat řadu modulů, které se dají jednoduše stáhnout pomocí npm[8]. V mém případě je to například `express`.

Node.js využívám pouze pro zpracování uživatelského hodnocení, jeho volba tak klíčová nebyla. Zvolil jsem ho hlavně kvůli dobré spolupráci s klientským JavaScriptem, práci s JSON, vhodnosti pro jednostránkové aplikace a také kvůli možnosti použít jeden jazyk na serveru i klientovi. Pro možné budoucí rozšíření je pak i dobře škálovatelný.

5.5 Vue.js

Vue.js [36] je JavaScriptový framework používaný pro vytváření rozhraní. Pro jeho používání nám stačí jen HTML, JavaScript a CSS. Mezi jeho silné stránky patří jednoduchost a možnost škálovatelnosti. Hlavní jádro toho frameworku je využito převážně na zobrazovací funkcionalitu, uživatelské rozhraní. Vue.js však nabízí další knihovny a nástroje, které z něho dělají komplexní řešení. Je tak možné ho využít na plnohodnotné jednostránkové aplikace i na jednoduché, znovu použitelné komponenty. Ty mají vlastní stav a funkce. Nabízí také virtuální DOM[9], kdy je reprezentace uživatelského rozhraní uložena v paměti a do reálného DOMu jsou efektivně promítnuty jen důležité změny, které pak zbytečně aplikaci nezpomalují.

Pro Vue.js jsem se rozhodl kvůli tomu, že z povahy mojí aplikace nevyužiji všechny funkce větších frameworků. To co naopak potřebuji je rychlost, přehlednost a dobrá interakce s jinými knihovnami, což jsou body ve kterých je Vue.js velmi silný. Navíc, díky znovu použitelným komponentám, mohu k uživatelskému prostředí jednotlivých metod přistupovat podobně, ačkoliv jsou svojí funkcionalitou velmi rozdílné.

[8] Správce balíčků https://www.npmjs.com/.

[9] Document Object Model - reprezentace HTML dokument.

```
<template>
  <div class="score-container">
    <span v-for="i in stars" :key="i">
      <span class="star" @click="() => { changeScore(i) }"><i v-if="
          score < i" class="far fa-star"></i></span>
      <span class="star star-active" @click="() => { changeScore(i) }
          "><i v-if="score >= i" class="fas fa-star"></i></span>
    </span>
  </div>
</template>
```

Zdrojový kód 2: Šablona komponenty ScoreWindow.

5.6 Python

Python [37] je vysokoúrovňový, univerzální, multiplatformní programovací jazyk. Vznikl již v roce 1991 a řadí se do kategorie interpretovaných. Nabízí dynamické typování, umožňuje různá programovací paradigmata a to včetně objektově orientovaného, procedurálního nebo funkcionálního. Vyniká svou bez závorkovou syntaxí a možnostmi. Python ve verzi 3 je velmi populární, existuje pro něj celá řada knihoven, má široké využití. Mezi oblasti, kde se využívá patří: desktopový i webový vývoj, věda, matematika, vzdělání a v dnešní době i strojové učení.

Python nabízí ohromné možnosti, je relativně rychlý a jednoduchý. Hlavním důvodem pro jeho výběr byl však framework Tensorflow.

5.7 Tensorflow

Tensorflow [38] je populární open source platforma pro strojové učení, jenž se používá ve velkém množství domén. Obsahuje rozsáhlý ekosystém nástrojů, knihoven a zdrojů pro strojové učení. Poskytuje stabilní Python a C++ API. Navíc nabízí i variantu pro zrychlení výpočtu pomocí grafické karty `TensorFlow GPU` která, ačkoliv má grafická karta nebyla na seznamu podporovaných karet, dokázala výpočet zrychlit zhruba pětkrát.

S využitím vysokoúrovňového API Keras [35] je možné s Tensorflow relativně jednoduše navrhovat i vytvářet neuronové sítě různých druhů. To nabízí ale i jiné technologie. Důležité pro mě bylo to, že existuje ve dvou verzích již popisované verzi v Pythonu, ale také verzi v JavaScriptu [39], která umožňuje pracovat s modely strojového učení přímo v prohlížeči. Nabídne mi tak možnost generovat skladby přímo na webu bez toho, aniž bych ztratil výhody učení na desktopu. Tensorflow tak pro mě byla jediná rozumná volba.

5.8 Web Audio API

Webové rozhraní [40], které nabízí velké možnosti práce s audiem na webu. Umožňuje si zvolit zdroj zvuku, přidání efektů, vizualizaci audia, práci s jeho kanály,

přehrávání, ale i analýzu hudby. V reálném čase je tak možné například mikrofonem zachytit zvuk, získat jeho frekvenci a určit jeho tón. Volba byla v tom případě určena, jelikož se jedná v podstatě o jedinou možnost práce s audiem v prohlížeči.

5.9 Tone.js

Tone.js [41] je JavaScriptový framework postavený na Web Audio API, který je určený pro interaktivní vytváření hudby v prohlížeči. Na rozdíl od klasického Web Audio API přináší pokročilé plánování, syntezátory a řadu funkcí, které běžně najdeme DAW. Navíc nabízí větší přístupnost pro uživatele znalé hudební teorie. Je tak možné na jeho základech vybudovat pokročilejší funkce jako je například přehrávač nebo metody práce s časovými vlastnostmi skladby.

Pro tento framework jsem se rozhodl kvůli zvolení MIDI technologie napříč celou aplikací a propojení metod s hudební teorií. Pro delší skladby, a ty složené z několika stop, má v některých oblastech výkonnostní potíže a obsahuje drobné chyby, například při určení délky skladby. Většinu věcí však řeší velmi dobře.

5.10 MongoDB

MongoDB [50] je databázový systém, jenž je jedním z hlavních představitelů NoSQL[10] databází. Řádek je nahrazen strukturovaným dokumentem. Zde je tímto dokumentem BSON (binární JSON), ten se uchovává v kolekcích. MongoDB umožňuje provádět dotazy přímo na data, a tím zachovává velkou část možností, kterou nabízí například i populární jazyk SQL[11].

Jelikož ukládám pouze jednoduchá anonymní uživatelská hodnocení, stačí k aplikaci pouze jedna tabulka/kolekce. Důvodem tak byla hlavně dobrá spolupráce s Node.js, rychlost, nenáročnost a flexibilní schéma, které se může hodit nejen pro postupný vývoj, ale také pro případné rozšíření aplikace.

[10] Databáze, kde se nepoužívají tabulková schémata (jako u relačních databází).
[11] Dotazovací jazyk, který je používán pro práci s daty v relačních databází.

6 Programátorská dokumentace

V této části obecně popíši rozdělení aplikace do adresářů, návrh databáze a přístupy, které jsem zvolil při vytváření aplikace.

6.1 Adresářová struktura

Aplikace se skládá ze dvou částí, webové aplikace ve Vue.js a server v Node.js. Podobné rozdělení jsem zachoval i v samotné adresářové struktuře, kterou jsem rozdělil na dvě části `app` a `backend`. V adresáři `backend` nalezneme soubory s kódem serveru. Rozložení složek a souborů částečně vychází z kostry aplikace vzniklé při vygenerování[12] nového Express projektu. Neobsahuje však adresáře, které souvisí s generováním obsahu, jelikož Express využívám pouze jako API hodnocení a o samotný obsah se stará Vue.js.

Vue.js nepřichází se striktním rozdělením souborů. Podobně jako u Express jsem tak zvolil strukturu nově vygenerované Vue.js aplikace. Pro práci nejdůležitější složkou je pak `src`. Ta se nachází v adresáři `app`. Obsahuje většinu zdrojových kódů komponent, pomocných funkcí a hlavně metod.

V obou adresářích se také nachází, pro JavaScriptový projekt typický, soubor `package.json`. V něm najdeme informace o projektech, závislostech, ale také o skriptech. Je velmi důležitý pro lokální spuštění aplikace (viz příloha A).

6.2 Databáze

Pro práci s hodnocením je potřeba jeho perzistentního uložení. Jednotlivá uložená data nejsou závislá na jiných záznamech či kolekcích a ani spolu plně nesouvisí. Relační databáze by zde tedy nevyužila plný potenciál. Do MongoDB kolekce „scores“ v databázi ukládám pouze jeden typ dokumentu. Ten představuje jedno uživatelské hodnocení. Tento dokument pak vychází ze schématu stanoveným Mongoose. Pro konkrétní databázi využívám cloudovou službu mLab[13], jelikož je velmi dobře dostupná a zdarma.

6.3 Web Worker

Využívání neuronových sítí na webu je stále spojeno s relativně náročnou režií, jelikož často dochází k velkému zpracování dat. Mimo stahování potřebných souborů, jako je model nebo dataset, jsou náročné také samotné výpočty. Abych se vyhnul zablokování grafického uživatelského prostředí nebo celkově špatné uživatelské zkušenosti, využil jsem technologii, kterou JavaScript v dnešní době nabízí – vlákna. Web Worker [42] dokáže spustit skript v pozadí, tam je oddělený od vykonávání hlavního vlákna a komunikace probíhá pomocí zasílání zpráv. Výhodou je pak tedy hlavně absence blokace nebo zpomalení aplikace pro uživatele. Bylo

[12] https://expressjs.com/en/starter/generator.html
[13] https://mlab.com/

však potřeba změnit některé přístupy, kvůli špatné spolupráci několika technologií s „Web Workerem“.

6.4 Vlastní komponenty

Projekty tvořené pomocí frameworku Vue.js jsou složeny z komponent viz 5.5. Těch jsem musel, v této aplikaci, sám vytvořit velké množství. Důvodem bylo, že je velmi málo Vue.js komponent pracujících s hudbou. Ačkoliv některé mé komponenty jsou úzce spjaty s touto aplikací, většina je plně nezávislá, připravená pro použití i v jiných projektech. Příkladem takovým komponent je piano, přehrávač a editor stop. Ty je pak možné využít například v plnohodnotnějším DAW. Dále také systém vyskakovacích oken, který se obecně hodí do velkého množství webových aplikací. Komponenty jsou chráněny před zbytečným překreslováním a zapouzdřují složitou funkcionalitu. Tyto a další komponenty jsou umístěny v aplikaci `\src\Components` a jsou hlavně propojeny `.\App.js`.

6.5 Přehrávač

Důležitým prvkem celé aplikace je přehrávání. Ať už jde o existující MIDI písně, skladby, které si naklikal uživatel nebo výsledky implementovaných metod. Přehrávač musí dokázat přehrát píseň, která obsahuje jen informace o tom, jak dané noty zahrát. Musí dokázat správně interpretovat hlavičku MIDI souborů pro korektní časové hodnoty. Také je potřeba správně zpracovat různé zdroje zvuku, aby uživatel mohl píseň opravdu slyšet. Jelikož jsem nenalezl vhodný přehrávač, který by splňoval všechny požadavky, rozhodl jsem se vytvořit vlastní pomocí frameworku Tone.js. Můj přehrávač funguje ve verzi taktů a sekund a přináší řadu možností. Je stavěn hlavně na známá časová předznamenání jako je 3/4 nebo 4/4 takt. Pro další neproběhly podrobnější testy. Mimo velmi neobvyklého předznamenání, jako je například 11/8 nebo 8/12, by však chování přehrávače mělo být korektní. Toto případné omezení nemá v rámci této práce velký vliv, jelikož metody, kvůli objektivnějšímu hodnocení generují pouze skladby s časovým předznamenáním, na které by měl být uživatel zvyklý. Přehrávač se vyskytuje na různých místech stránek ať už v podobně hlavního přehrávače, či přehrávače výsledku metody nebo přehrávačů jednotlivých skladeb při hodnocení v genetické metodě.

6.6 Syntezátory

Jelikož nám MIDI soubory pouze říkají co a jak hrát a samy o sobě zvuk neobsahují, je nutné je nějakým způsobem ozvučit. V rámci aplikace jsem poskytl dva způsoby jakým produkovat zvuk, a to využitím oscilátorů a samplů. Ve Web Audio API [40] jsou základní oscilátory označeny jako matematicky vypočítané zvuky. Jsou jednoduše zobrazitelné jako zvuková vlna, jejíž tvar určuje zvuk. Čtyři základní tvary vln jsou „sine“, „triangle“, „square“, a „sawtooth“. Ty jsem poskytl i v rámci aplikace. Zvuk těchto oscilátorů však zní velmi uměle. V písních

se s nimi často nesetkáváme. Obecně se tak hodí spíše do určitých typů hudby. V ostatních případech můžou celkový zážitek metody spíše narušit.

Přirozenější zvuk pak řeší zmiňované samply. Jednoduše řečeno se, v tomto případě, jedná o malé zvukové soubory, které odpovídají jednotlivým notám. Když pak přehrávač narazí například na notu C4, přehraje požadovaný soubor. To je tedy rozdílné od oscilátorů, které zvuk vygenerují. Samply sebou však přináší několik problémů. Řada z nich podléhá nějaké licenci, ne vždy jsou poskytnuty všechny noty, které potřebuji a zvětšuje se režie stránky při načítání (pro více přehrávačů je třeba stáhnout samply zvlášť). Tyto problémy jsou pak většinou řešitelné. Existují i licence zdarma. Vhodným použitím knihovny Tone.js lze samply transponovat, takže je lze použít i pro noty, které v souborech nemáme. Poslední bod pak řeší to, že režie stažení všech samplů je v dnešní době velikosti webů zanedbatelná (hlavně na počítačích, pro které je aplikace určená).

Logiku přehrávače jsem naprogramoval, tak, aby bylo možné to, že uživatel bude moci přidat samply vlastní, a to pomocí zadání url s koncovými body odpovídajících samplů, principiálně jako AJAX. Zde jsme však omezení množstvím těchto stránek, které se v podstatě nevyskytují a když ano, je potřeba, aby byly korektní z hlediska CORS[14]. Řešením by bylo mít server, vlastní úložiště pro samply, kde by je uživatel mohl nahrát. Zde by ale hrozila varianta, že by uživatel nahrál samply, které by podléhaly nějaké licenci, která by to znemožňovala. Z tohoto důvodu jsem se rozhodl tuto možnost před uživatelem skrýt a dát k dispozici jen skupinu samplů, kterou lze v budoucnu jednoduše rozšířit jen tím, že je umístím na server.

6.7 Export audia

V rámci funkcionality, přidané nad rámec zadání, jsem se pro lepší funkcionalitu rozhodl implementovat export základní skladby do souboru WAV. Obecně však na tento export nejsou mnou použité webové technologie příliš dobře připravené.

Pro převedení písně do AudioBuffer (aby ji bylo možné vygenerovat) je nutné nejdříve zpracovat nový přehrávač s novým audio kontextem, v tomto případě OfflineContext, a skladbu předpřipravit. Předpříprava spočívá v připravení bufferů pro samply, konfigurace časových předznamenání a naplánování not. Samotné zpracování je pak v režii frameworku Tone.js. Ten ale, při složitějších skladbách, selhává z hlediska trvání zpracování, které může trvat i kolem minuty. Toto omezení jsem však akceptoval, jelikož neexistuje příliš mnoho alternativ, časově to netrvá nepřijatelně dlouho (vzhledem k délkám skladeb) a nakonec je to užitečná funkce, kterou uživatel nebude provádět příliš často. Po přípravě AudioBufferu je pak nutné ho převést do formátu, který je možné přehrát na co nejvíce platformách. Rozhodl jsem se tak využít formát WAV.

[14]CORS - Mechanismus umožňující sdílení zdrojů webové stránky pro aplikaci na jiné doméně.

6.8 Knihovna hudební teorie

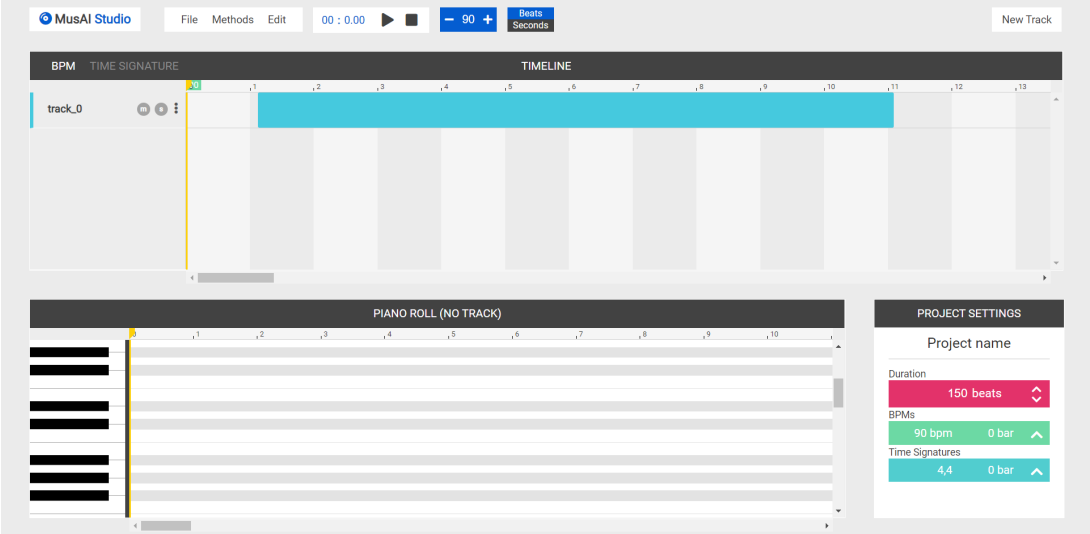
Z důvodu velkého využití hudební teorie je třeba algoritmicky vyřešit celou řadu problémů. Mezi ně patří: konstrukce všech požadovaných stupnic s odpovídajícími akordy, určení pravděpodobné tóniny, převádění mezi různými časovými předznamenáními nebo rovnou sekundami a dobami, změny hodnot not, transponování písní, pracování s modalitou a řada dalších. Jelikož jsem nenašel knihovnu, která by toto vše poskytovala, rozhodl jsem se většinu řešení těchto problémů navrhnout a implementovat sám. Příkladem je určení tóniny. V praxi se běžně používají neuronové sítě nebo Krumhansl-Schmuckler Key-Finding Algoritmus [43] (ten je založen na tóninových profilech, což je vektor 12 hodnot). Ty byly však relativně nevhodné, protože bylo potřeba určit tóninu na velmi krátkých melodiích, které neobsahují harmonické složky a často vychází z modality. Tedy hlavně kvůli větší časové a výpočetní režii jsem vytvořil vlastní algoritmus přímo na tyto účely pracující dle daných not a klíčových intervalů. Tuto funkci a většinu ostatních naleznete v `utils.js` a `scales.js`. Dohromady pak poskytují základ, který je využíván napříč celou aplikací a je možné ho využít i v aplikacích jiných.

7 Uživatelská dokumentace

Tato část popisuje základní kroky pro ovládání aplikace a použití metod. Funguje tak jako příručka pro uživatele. Protože se jedná o webovou aplikaci, k použití popsaných funkcí stačí pouze internetový prohlížeč. Doporučeny jsou novější verze Google Chrome, více informací v příloze D.

7.1 Rozložení aplikace

Editor se skládá ze čtyř částí – kontejnerů. Hlavním prvkem stránky je editor stop. Ten obsahuje nejen všechny stopy, ale také panel s časovými jednotkami (na výběr jsou sekundy nebo doby) pro vizualizaci a úpravu času. Nad editorem stop je pak menu pro práci se soubory, metodami, časem a vpravo vytvoření nové stopy. Na konci stránky nalezneme pianový editor, který slouží pro práci s notami. Vpravo od něj leží kontejner pro nastavení celého projektu, stopy nebo noty, jehož obsah se mění dle vybraného prvku.

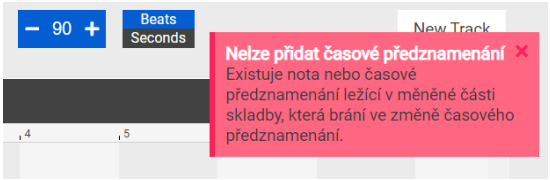


Obrázek 18: Rozložení úvodní stránky.

Při užívání aplikace uživatel může narazit na řadu informačních hlášení. Jejich pozice i vzhled je napříč aplikací konzistentní. Vyskakují vpravo nahoře, řadí se pod sebe a vždy po nějakém čase zmizí. Tyto hlášení pak můžeme rozdělit na tři typy, jsou barevně odlišené – varovné (červená), informativní (modrá) a oznamující úspěch (zelená). Objevují se pak v horní části obrazovky (například obrázek 19).

7.2 Menu

Menu je rozděleno do několika částí. První je část pro práci se soubory. Ta umožňuje import a export souborů. Existují tři typy souborů, které lze exportovat a



Obrázek 19: Chybové hlášení při přidání časového předznamenání.

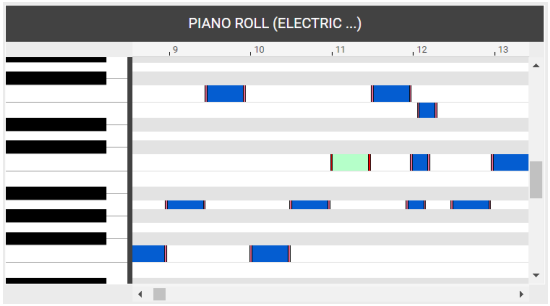
to: MIDI, WAV a JSON – ten představuje samotný projekt, který je poté v budoucnu možné znova načíst. Stejně tak lze importovat i soubor MIDI. Další částí je volba samotných metod a klasická tlačítka vpřed a zpět. Mimo to, v horní části stránky, leží i ovladač pro přehrávač, vstup pro změnu velikosti šířky doby nebo sekundy, přepínač zobrazeného módu a tlačítko pro vytvoření nové stopy.

7.3 Editor stop

V rámci editoru stop je možné zobrazit si změny BPM a časových předznamenání. Umožňuje odstranění, kopírování, vyjmutí a vložení stop. S těmi dokáže také hýbat (a tím změnit začátek všech not) nebo zařídit, aby daná stopa nehrála nebo naopak hrála jako jediná.

7.4 Pianový editor

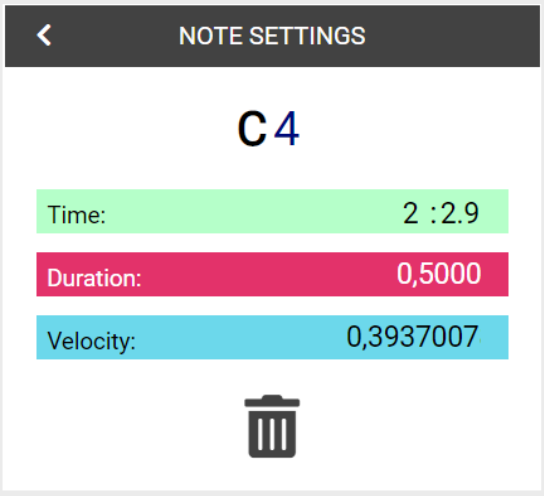
Jediným způsobem pro vkládání not je pianový editor, jenž je možné vidět na obrázku 20. Ten představuje klasické piano a vztahuje se k aktuálně vybrané stopě. Do ní dokáže, pomocí kliknutí, vkládat noty anebo noty naopak upravit (rozšířit, posunout a transponovat). Funguje také hromadný výběr pomocí klávesy CTRL.



Obrázek 20: Piano zaplněné notami.

7.5 Okno nastavení

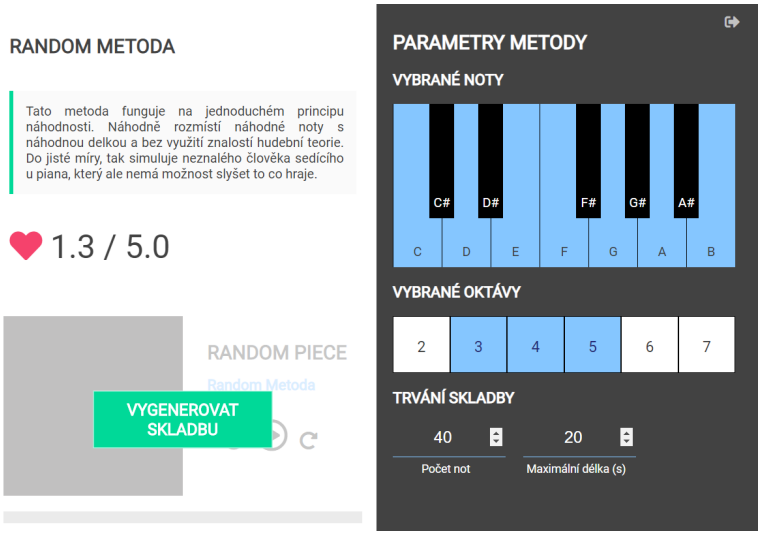
Tento kontejner slouží pro nastavování prvků dle jejich výběru. Jeho obsah se mění na základě výběru na projektový, stopový a notový. V rámci nich dokáže například měnit jméno projektu, BPM, časové předznamenání, samply, jméno stopy a vlastnosti not (na obrázku 21).



Obrázek 21: Příklad editace noty.

7.6 Metody

Mimo hlavní stránku je zde také jiná - s možnými metodami (v menu tlačítko „Methods“). Konkrétní prvky v těchto stránkách se liší dle použité metody (příklad a popis jejich prvků naleznete v sekci 4). Základ však zůstává stejný a je možné ho vidět na obrázku 22. Levá strana obsahuje jednoduchý popis metody a tlačítko pro generování. To po kliknutí vygeneruje skladbu a zobrazí přehrávač. Přehrávač je doplněn o obrázek, který částečně vychází z náhody a z vygenerované písně a na něm se objeví vstup pro hodnocení metody. V přehrávači je dále možné vyžádat generování znovu a skladbu přehrát nebo ji vybrat do editoru. Pravá strana pak obsahuje konkrétní vstupy metody.



Obrázek 22: Příklad UI metody Random.

8 Rozšíření aplikace

V práci jsem se zaměřil a implementoval všechny důležité metody. Ve světě počítačového skládání hudby je však ještě celá řada věcí k prozkoumání. Logickým krokem pro vylepšení práce je zaměření se na jiné varianty zde popsaných metod, případně zapojení specifičtějšího generování zvuku – například na základě emoce, příběhu, složitosti hraní a další. Co se týče samotné aplikace, tu jsem se snažil psát tak, aby tyto případné změny v budoucnu dokázala reflektovat. Metody jsou zde ve formě modulů, které lze nezávisle upravovat na celkové aplikaci. Důležité je, aby měla řádný vstup a vracela správný výstup.

Aplikaci je však možné rozšířit i mimo metody. Nyní se jedná, pomineme-li generování hudby, o poměrně nadstandardní MIDI editor (alespoň na poměry webových aplikací). Dalo by se však jeho funkcionalitu rozšířit na téměř plnohodnotný DAW software (tak jak to dovolí webové technologie), ve kterém by bylo možné píseň plnohodnotně dokončit. Umožnit aplikaci pracovat s co největším počtem druhů souborů hudby. Jelikož jsem většinu funkcí pro práci s hudbou navrhl sám, nebylo plně možné odzkoušet je na různých typech skladeb (např. s netypickými časovými předznamenáními, které se často mění), v rámci rozšíření by tak bylo vhodné je otestovat. Další možností by bylo přidání vlastních samplů. Tato funkce je již nyní podporována, problémem je však absence stránek, ze kterých by se daly zpracovat. Také by se hodilo aplikaci v některých částech zrychlit nebo zpříjemnit uživatelskou zkušenost. Často však, jelikož se jedná o webovou aplikaci, narážíme na různá omezení. Dalším rozšířením aplikace by mohla být možnost dovolit uživateli pracovat s vlastními datasety a vytrénovat váhy vlastní. Již nyní je aplikace nezávislá na datasetu. Vlastní váhy by si však musel vytrénovat skrze program v Pythonu, který leží mimo webovou aplikaci na desktopu.

Tyto funkce jsou však nad rámec zadání mé práce, jejíž úkolem bylo prozkoumat a implementovat typické metody generování hudby a umožnit s nimi pracovat. Nicméně přidaná funkcionalita by určitě zlepšila zejména komerční využití aplikace a její celkové používání.

Závěr

Svět generování hudby je velmi obsáhlý a jeho historie sahá až před samotnou existenci počítačů. Ve své práci jsem prozkoumal nejdůležitější metody. Ty pracují s úplnou náhodou, s formálními gramatikami, statistickými údaji, hudební teorií, genetickým vývojem i s neuronovou sítí. Práce se, na rozdíl od většiny ostatních, nezaměřuje pouze na jednu metodu a neposkytuje jen obecný přehled.

Použil jsem již zavedené metody, do kterých jsem se snažil přivést invenci a navrhl jsem i metodu vlastní. Kvalitu metod jsem zhodnotil sám, ale umožnil jsem i uživatelské hodnocení, jelikož nebylo možné využít jiný hodnotící prostředek, který by byl korektní ke každé metodě. Díky možnosti hodnotit přímo v aplikaci jsem dostal více objektivnější hodnocení, které netrpí únavou uživatele a reflektuje lépe jejich hudební vkus.

Výsledkem práce je tak webová aplikace pro skládání hudby. Aplikace slouží jako platforma pro lidi, kteří hudbě rozumí, ale i pro amatéry. Ať už mají vlastní hudební myšlenku, kterou si chtějí vytvořit nebo využijí ideu některé metody. Je to tak hlavně nástroj pro skladatele, nejedná se o prostředek pro produkci hudby. Vytvořenou skladbu pak mohu přehrát, vyexportovat do pokročilejšího DAW nebo vytvořit základní poslouchatelnou verzi ve formátu WAV. I bez skládání je možné využít aplikaci jako MIDI editor. Díky návrhu a použití Vue.js komponent je relativně snadné aplikaci rozšířit. Tyto komponenty, které jsem sám vytvořil, jsou použitelné i pro podobné projekty. Aplikaci budu nadále rozvíjet.

Conclusions

The world of music generation is extensive and its history dates to times before existence of computers. In this thesis, I have explored the most important methods. They operated with random values, formal grammars, statistical data, music theory, genetic algorithms and with a neural network. Unlike most others, this thesis does not focus only on one method or providing of a general overview.

I used existing methods. Into the methods I tried to bring invention. I have also designed my own method. I evaluated the quality of the methods myself, but I also provided user evaluation, because it was not possible to use another evaluation tool that would be correct for each method. Thanks to the possibility to evaluate directly in the application, I received a more objective results, which does not suffer from user fatigue and better reflects their musical taste.

The result of the thesis is a web application for composing music. The application serves as a platform for people who know a lot about music theory, but also for amateurs. Whether they want to create their own musical idea or use the idea of a method. It is mainly a tool for composers, not for producing music. It can play the composed piece, export it to a more advanced DAW or create a version in WAV. Even without composing, it is possible to use the application as a MIDI editor. Thanks to the design and use of Vue.js components, it is relatively easy to expand the application. These components, which I have created by myself, are also usable for similar projects. I will continue to develop the application.

A Instalace a spuštění serveru

A.1 Požadavky:

Pro spuštění serveru lokálně je nutné mít:

- Nainstalovaný Node.js (k vývoji byla použita verze `v10.16.0`, dostupná na adrese https://nodejs.org/en/download/)
- Nainstalovaný správce balíčků npm, který je součástí instalace Node.js
- Neobsazený zvolený port (typicky 3000)

A.2 Spuštění serveru:

Pro spuštění serveru proveďte v adresáři `backend` na vašem disku příkazy v příkazovém řádku:

1. `npm install` (stažení požadovaných závislostí)
2. `npm run start` (spuštění serveru)

Po správně provedeném startu server běží na adrese http://localhost:3000/

B Instalace a spuštění aplikace:

B.1 Požadavky

Pro spuštění aplikace je nutné mít:

- Nainstalovaný správce balíčků npm viz. A.1
- Neobsazený port 8080

B.2 Spuštění aplikace:

Pro spuštění aplikace proveďte v adresáři `app` příkazy v příkazovém řádku:

1. `npm install` (stažení požadovaných závislostí)
2. `npm run serve` (spuštění aplikace)

Po správně provedeném startu server běží na adrese http://localhost:8080/

C Neuronová síť

Pro spuštění a trénování neuronové sítě je nutné mít Python - 3.7.3 a provést potřebné instalace, které jsou uvedené na přiloženém DVD. V rámci spuštění sítě musí být zavolána funkce `train` v souboru `.\src\net\lstm.py`, dále je možné změnit řadu hodnot uvedených v daném souboru.

D Testování

Pro účely testování při tvorbě posudků práce je možné využít lokální instalace (viz přílohy A a B) nebo také veřejně dostupné verze zprovozněné na serveru:

- aplikace - http://158.194.92.80:3000/

Testovací verze je omezena na pouze 4 samply, obsahuje zmenšenou historii změn a má vypnuté hodnocení (v rámci zachování stejných hodnot s textem práce).

Aplikace byla vyvíjena a testována v prohlížeči Google Chrome a to ve verzi `84.0.4147.105` na 64bitovém Windows 10 Home. Dále proběhl rychlý test ve Firefox `79`, u něj a ostatních však není možné zaručit plnou funkčnost. Stejně tak funkčnost nelze plně zaručit u všech typů MIDI souborů z důvodu vlastního návrhu hudebních funkcí, které bylo nad rámec zadání práce.

E Obsah přiloženého CD/DVD

Přiložené CD/DVD obsahuje aplikaci a digitální verzi diplomové práce.

`doc/`
: Text práce ve formátu PDF, včetně všech příloh a všech souborů potřebných pro bezproblémové vygenerování PDF dokumentu textu.

`src/`
: Zdrojové kódy webové aplikace, její serverové části i Python neuronové sítě.

`readme.txt`
: Instrukce pro instalaci, spuštění serveru, webové aplikace a Python neuronové sítě včetně požadavků pro jejich bezproblémový lokální provoz. Nachází se zde webová adresa, na které je aplikace nasazena pro účel testování při tvorbě posudků práce a další informace.

Literatura

[1] HERREMANS, Dorien, CHUAN Ching-Hua a CHEW Elaine. A Functional Taxonomy of Music Generation Systems. 2017.

[2] HEDGES, Stephen A. Dice Music in the Eighteenth Century. Music & Letters, Vol. 59, pp. 180-187. 1978.

[3] LOVELACE, Ada. „Notes on L. Menabrea's 'Sketch of the Analytical Engine Invented by Charles Babbage, Esq.'" 1843.

[4] HILLER, Lejaren A., ISAACSON, Leonard M. Dice Musical Composition with a High-Speed Digital Computer. JAES Volume 6 Issue 3 pp. 154-160. 1958.

[5] SCHOENBERG, Arnold. Fundamentals of Musical Composition. 1999. ISBN: 9780571196586

[6] PEARCE, Marcus, WIGGINS Geraint. Towards A Framework for the Evaluation of Machine Compositions. 2001.

[7] TOKUI, Nao, IBA Hitoshi. Towards Music Composition with Interactive Evolutionary Computation. 2000.

[8] CHRISTENSEN Thomas W. The Cambridge History of Western Music Theory. 2001. ISBN: 9780521686983

[9] BÖHMOVÁ, Zdenka, GRÜNFELDOVÁ Arnoštka, SARAUER Alois. Klavírní škola pro začátečníky. 2002. ISBN: 979-0-2601-0158-6

[10] WALTER Piston. Harmony. 1987. ISBN: 978-0393954807

[11] LEVINE Mark. Harmony. 1989. ISBN: 9780961470159

[12] ALDWELL, Edward, SCHACHTER Carl. Harmony and Voice Leading. 2010. ISBN: 9781439083253

[13] HARRISON, Mark. Intro to Jazz Piano. 2011. ISBN: 9781617803109

[14] GALLON, Justion. A Practical Approach to Understanding Time Signatures. 2011. ISBN: 978-1466205710

[15] Abc notation [online]. [cit. 2020-07-26]. Dostupné z: http://abcnotation.com/

[16] HARTMANN, William M. Signals, Sound, and Sensation. 2004. ISBN: 9781563962837

[17] SAVARD, Alexandre.Overview of Homophonic Pitch Detection algorithms

[18] TAN Lin, KARNJANADECHA Montri. Pitch detection algorithm: Autocorrelation method and AMDF. 2003.

[19] MIDI [online]. [cit. 2020-07-29]. Dostupné z: https://www.midi.org/

[20] HOPCROFT, John E., MOTWANI Rajeev, ULLMAN Jeffrey D. Automata Theory, Languages, and Computation. 2002. ISBN: 978-0-321-45536-9

[21] MCCORMACK, Jon. Grammar Based Music Composition. Complex Systems 96, pp. 320-336. 1996.

[22] FACEVICOVÁ, Kamila, HRON Karel, KUNDEROVÁ Pavla. Markovovy řetězce a jejich aplikace. 2018. ISBN: 978-80-244-5432-0

[23] TYMOCZKO, Dmitri. A Geometry of Music: Harmony and Counterpoint in the Extended Common Practice. 2011. ISBN: 978-0195336672

[24] HYNEK, Josef. Genetické algoritmy a genetické programování. 2008. ISBN: 978-80-247-2695-3

[25] HOLLAND, John H. Adaptation in Natural and Artificial Systems. 1975. ISBN: 9780262082136

[26] MITCHELL, Melanie. An Introduction to Genetic Algorithms. 1998. ISBN: 9780262631853

[27] VOLNÁ, Eva. Neuronové sítě 1. 2008.

[28] GOODFELLOW, Ian, BENGIO, Yoshua, COURVILLE, Aaron. Deep Learning. 2016. ISBN: 978-0262035613

[29] LIPTON, Zachary C. A Critical Review of Recurrent Neural Networks for Sequence Learning. 2015.

[30] HOCHREITER, Sepp, SCHMIDHUBER, Jürgen. Long Short-Term Memory. 1997.

[31] Layer activation functions [online]. [cit. 2020-07-26]. Dostupné z: https://keras.io/api/layers/activations/

[32] Keras - Optimizers [online]. [cit. 2020-07-26]. Dostupné z: https://keras.io/api/optimizers/

[33] HEWAHI, Nabil, ALSAIGAL, Salman, ALJANAHI, Sulaiman. Generation of music pieces using machine learning: long short-term memory neural networks approach. 2019.

[34] PAYNE, Christine. Clara: Generating Polyphonic and Multi-Instrument Music Using an AWD-LSTM Architecture. 2018.

[35] Keras [online]. [cit. 2020-07-26]. Dostupné z: https://keras.io/

[36] Vue.js [online]. [cit. 2020-07-30]. Dostupné z: https://vuejs.org/

[37] Python [online]. [cit. 2020-07-24]. Dostupné z: https://www.python.org/

[38] Tensorflow [online]. [cit. 2020-07-26]. Dostupné z: https://www.tensorflow.org/

[39] TensorFlow.js | Machine Learning for Javascript Developers [online]. [cit. 2020-07-26]. Dostupné z: https://www.tensorflow.org/js

[40] Web Audio API [online]. [cit. 2020-07-28]. Dostupné z: https://www.w3.org/TR/webaudio/

[41] Tone.js [online]. [cit. 2020-07-28]. Dostupné z: https://tonejs.github.io/

[42] Using Web Workers API [online]. [cit. 2020-07-28]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/API/Web_Workers_API

[43] TEMPERLEY, David. What's Key for Key? The Krumhansl-Schmuckler Key-Finding Algorithm Reconsidered. Music Perception, Vol. 17, No.1, pp. 65-100. 1999.

[44] About JavaScript - JavaScript | MDN [online]. [cit. 2020-07-26]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/JavaScript/About_JavaScript

[45] Classes - JavaScript | MDN [online]. [cit. 2020-07-26]. Dostupné z: https://developer.mozilla.org/cs/docs/Web/JavaScript/Reference/Classes

[46] HTML5 - Developer guides | MDN [online]. [cit. 2020-07-26]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/Guide/HTML/HTML5

[47] CSS3 [online]. [cit. 2020-07-26]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/CSS/CSS3

[48] About [online]. [cit. 2020-07-26]. Dostupné z: https://nodejs.org/en/about/

[49] The Node.js Event Loop, Timers, and process.nextTick() [online]. [cit. 2020-07-26]. Dostupné z: https://nodejs.org/en/docs/guides/event-loop-timers-and-nexttick/

[50] What Is MongoDB? | MongoDB [online]. [cit. 2020-07-26]. Dostupné z: https://www.mongodb.com/what-is-mongodb