VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky
a komunikačních technologií

BAKALÁŘSKÁ PRÁCE

Brno, 2021 Filip Dobrocký

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV TELEKOMUNIKACÍ
DEPARTMENT OF TELECOMMUNICATIONS

EXPERIMENTÁLNÍ SOFTWAROVÝ HUDEBNÍ NÁSTROJ, KOMBINUJÍCÍ KROKOVÝ SEKVENCER A SAMPLER
EXPERIMENTAL SOFTWARE MUSICAL INSTRUMENT, COMBINING STEP SEQUENCER AND SAMPLER

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR
Filip Dobrocký

VEDOUCÍ PRÁCE
SUPERVISOR
doc. Ing. MgA. Mgr. Dan Dlouhý, Ph.D.

BRNO 2021

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Bakalářská práce

bakalářský studijní program **Audio inženýrství**
specializace Zvuková produkce a nahrávání
Ústav telekomunikací

Student: Filip Dobrocký ***ID:*** 212553
Ročník: 3 ***Akademický rok:*** 2020/21

NÁZEV TÉMATU:

Experimentální softwarový hudební nástroj, kombinující krokový sekvencer a sampler

POKYNY PRO VYPRACOVÁNÍ:

Cílem práce je vytvoření funkčního prototypu experimentálního softwarového hudebního nástroje, kombinujícího krokový sekvencer a sampler. Neobvyklost řešení spočívá v kombinaci provádění i velmi komplikovaných polyrytmů, rozeznívání uživatelem vložených zvukových vzorků, a využití mikrointervalového ladění.

DOPORUČENÁ LITERATURA:

[1] PUCKETTE, M. Theory and Techniques of Electronic Music, 2006. 337 s. online: http://msp.ucsd.edu/techniques.htm

[2] FORRÓ D. Svět MIDI. Grada, Praha, 1997. 375s. ISBN 80-7169-412-6.

Termín zadání: 1.2.2021 ***Termín odevzdání:*** 31.5.2021

Vedoucí práce: doc. Ing. MgA. Mgr. Dan Dlouhý, Ph.D.

doc. Ing. Jiří Schimmel, Ph.D.
předseda rady studijního programu

ABSTRAKT

Cieľom tejto práce je navrhnúť a zhotoviť experimentálny softvérový hudobný nástroj kombinujúci krokový sekvencer a sampler. Výnimočnosť nástroja spočíva v schopnosti realizovať komplikované polyrytmy a vo využití mikrointervaliky. V teoretickej časti práce je preskúmaná problematika polyrytmov, hudobných ladení, mikroladenia elektronických hudobných nástrojov a je objasnený princíp krokových sekvencerov a samplerov. Praktická časť práce zahŕňa návrh a implementáciu nástroja v podobe VST plug-in modulu implementovaného pomocou frameworku JUCE.

KĽÚČOVÉ SLOVÁ

sampler, sekvencer, mikrotonalita, polyrytmy, VST, JUCE

ABSTRACT

The aim of this thesis is to design an experimental software musical instrument combining a step sequencer and a sampler. The distinctive qualities of said instrument lie in the ability to produce complicated polyrhythms and make use of microtonality. The theoretical part of the thesis explores concepts such as polyrhythms, musical intonation systems, microtuning of electronic musical instruments, and explains the fundamentals of step sequencing and sampling. The practical part of the thesis features the design and implementation of the instrument, which takes form of a VST plug-in module implemented using the JUCE framework.

KEYWORDS

sampler, sequencer, microtonality, polyrhythms, VST, JUCE

DOBROCKÝ, Filip. *Experimentální softwarový hudební nástroj*. Brno, 2020, 52 s. Bakalárska práca. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací. Vedúci práce: doc. Ing. MgA. Mgr. Dan Dlouhý, Ph.D.

VYHLÁSENIE

Vyhlasujem, že svoju bakalársku prácu na tému „Experimentální softwarový hudební nástroj“ som vypracoval samostatne pod vedením vedúceho bakalárskej práce, s využitím odbornej literatúry a ďalších informačných zdrojov, ktoré sú všetky citované v práci a uvedené v zozname literatúry na konci práce.

Ako autor uvedenej bakalárskej práce ďalej vyhlasujem, že v súvislosti s vytvorením tejto bakalárskej práce som neporušil autorské práva tretích osôb, najmä som nezasiahol nedovoleným spôsobom do cudzích autorských práv osobnostných a/alebo majetkových a som si plne vedomý následkov porušenia ustanovenia § 11 a nasledujúcich autorského zákona Českej republiky č. 121/2000 Sb., o práve autorskom, o právach súvisiacich s právom autorským a o zmene niektorých zákonov (autorský zákon), v znení neskorších predpisov, vrátane možných trestnoprávnych dôsledkov vyplývajúcich z ustanovenia časti druhej, hlavy VI. diel 4 Trestného zákonníka Českej republiky č. 40/2009 Sb.

Brno ....

podpis autora

POĎAKOVANIE

Rád by som poďakoval vedúcemu bakalárskej práce doc. Ing. MgA. Mgr. Danovi Dlouhému, Ph.D. za odborné vedenie, konzultácie a podnetné návrhy k práci.

Obsah

Zoznam obrázkov

Úvod

Cieľom tejto práce je vytvoriť experimentálny softvérový hudobný nástroj kombinujúci krokový sekvencer a sampler. Hudobné nástroje kombinujúce tieto dve zariadenia majú väčšinou podobu tzv. grooveboxov – nástrojov používaných pre živú produkciu elektronickej hudby založenej na repetícii. Navrhnutý nástroj bude fungovať na podobnom princípe, avšak oproti bežným grooveboxom bude dovoľovať väčšiu rytmickú a intonačnú slobodu, a to využitím polyrytmov a mikrointervaliky.

Experimentálnosť navrhnutého sekvenceru spočíva v schopnosti realizovať polyrytmy v ľubovolných pomeroch, zachovávajúc jednoduchosť zadávania rytmov krokových sekvencerov. Oproti bežným 16krokovým sekvencerom bude schopný variabilného počtu krokov v každom hlase, a teda tvorby značne zložitejších rytmov v rovnako jednoduchom uživateľskom rozhraní.

Oproti bežným samplerom, ktoré sa viažu na dvanásťtónové rovnomerne temperované ladenie prevaľujúce v západnej hudbe, bude navrhnutý sampler schopný využívať iné ladenia, a to nahrávaním ladiacich tabuliek zo súborov už zavedených formátov. Sampler bude polyfonický a bude mať možnosť vkladania vlastných zvukov osobitných pre každý hlas.

V teoretickej časti práce je preskúmaná problematika spojená s návrhom sekvenceru – kapitoly *Sekvencer* a *Polyrytmy* – a sampleru – kapitoly *Hudobné intervaly a ladenia*, *Mikroladenie softvérových hudobných nástrojov* a *Sampling*. Samotný úvod približuje tradíciu elektronických hudobných nástrojov, sústrediac sa na oblasti súvisiace s navrhnutým nástrojom. Tiež je predstavený protokol MIDI, ktorý nástroj bude využívať. Nástroj bude vytvorený pomocou aplikačného frameworku JUCE a bude mať podobu VST plug-in modulu – preto sú v teoretickej časti stručne predstavené aj tieto technológie.

Praktická časť práce spočíva v návrhu a realizácii funkčného experimentálneho hudobného nástroja. V tejto časti práce je priblížený princíp jednotlivých programových častí a spôsob ich implementácie pomocou programovacieho jazyka C++ a frameworku JUCE.

Sekvencery a samplery bežne existujú v rôznych podobách, ich kombinácia využívajúca polyrytmy a mikrointervaliku v rozšírenom formáte ako je VST však neexistuje, čo robí vytvorený nástroj výnimočným.

1 Teoretická časť

1.1 Elektronické hudobné nástroje

Elektronické nástroje majú veľkú tradíciu a ich história trvá už viac ako 140 rokov [1]. Tieto nástroje využívajú pre tvorbu zvuku elektrický zdroj, avšak ich vývoj je spojený aj s nástrojmi elektro-mechanickými – napr. Hammondov organ využívajúci rotujúce elektro-mechanické oscilátory alebo Mellotron, ktorého zdrojom zvuku je magnetická páska. Elektronické hudobné nástroje sú schopné napodobovať tradičné nástroje, ale zároveň vytvárajú veľký priestor pre experimentáciu a otvárajú nové zvukové a kompozičné možnosti. Táto práca sa okrem iného venuje mikrointervalike (viď kapitolu 1.4), ktorá bola v elektronických hudobných nástrojoch prítomná už v ich počiatku.

V roku 1897 Thaddeus Cahill vytvoril prvú verziu nástroja známeho ako *Telharmonium*, ktorý sa považuje za predchodcu elektronických hudobných nástrojov. Nástroj fungoval na princípe aditívnej syntézy realizovanej pomocou rotujúcich elektro-mechanických oscilátorov, teda na princípe, ktorý bol neskôr použitý v slávnom Hammondovom organe. Neskoršie verzie Telharmonia vážili až 200 ton. Keďže mechanický princíp nástroja vychádzal z prirodzeného ladenia, nástroj používal tri klaviatúry, ktoré teoreticky dokázali deliť oktávu na 36 segmentov [1]. Nástroj tak umožňoval použitie rôznych ladení, avšak bol určený pre bežnú hudbu v dvanásťtónovom rovnomerne temperovanom ladení, ktoré nástroj, i keď trochu zložitejšie, dovoľoval.

V 20. rokoch 20. storočia vzniklo viacero elektronických hudobných nástrojov s neobvyklým spôsobom ovládania, ktoré umožňovali mikrointervalovú hru. Nástroj *Theremin* z roku 1922 produkoval nepretržitý tón, ktorého výšku a hlasitosť bolo možné meniť plynulo, bezkontaktne pomocou dvoch antén. Nástroj sa pre jeho unikátny zvuk a spôsob hry stále používa. Podobný charakter majú aj *Martenotove vlny* z roku 1928 založené na princípe Thereminu. Oproti nemu Martenotove vlny pridávajú väčšiu tembrálnu variabilitu a klaviatúru, pričom obsahoval aj drôt paralelný s klaviatúrou, ktorý dovoľoval väčšiu intonačnú slobodu.

Ďalším podobným nástrojom je *Trautonium* z roku 1930, ktorého účelom bolo oslobodenie sa od temperovaného ladenia. Bol ovládaný pomocou drôtu, ktorého odpor hráč dotykom menil – horizontálny pohyb menil výšku tónu a prítlak ovplyvňoval dynamiku. Nástroj fungoval na princípe subtraktívnej syntézy a na tú dobu umožňoval veľkú tembrálnu rozmanitosť.

Nemeckého hudobníka a vynálezcu Jörga Magera jeho celoživotná fascinácia mikrointervalovou hudbou viedla k vytvoreniu viacerých elektronických hudobných nástrojov. V 20. rokoch 20. storočia vyvíjal nástroj s názvom *Sphäraphon*. Nástroj

bol najskôr ovládaný otočnou pákou plynulo meniacou tón a neskôr vznikla verzia s dvoma monofónnymi klaviatúrami, ktorých tónový rozsah bolo možné meniť – rozsah oktávy sa dal zmenšiť až na veľkú sekundu.

Jedna z oblastí, pre ktorú sú elektronické hudobné nástroje veľkým prínosom, je možnosť tvorby hudby bez nutnosti hudobníkov. Túto možnosť už dávno predtým naplňovali tzv. *automatofóny* ako hracie skrinky, flašinety alebo pianola, fungujúce na mechanickom princípe. Elektronické hudobné nástroje však priniesli rozsiahlejšie možnosti automatizácie hudobného prejavu prostredníctvom *hudobných sekvencerov* (viď kapitolu 1.2). Krokové sekvencery, na ktoré sa táto práca zameriava, sa objavili s analógovými syntetizátormi v 60. rokoch. Tie sú oproti mechanickým automatofónom z hľadiska rytmických možností krokom späť a môžu pôsobiť repetitívne – to však môže slúžiť ako umelecký prostriedok a na tejto forme repetície sú založené viaceré hudobné žánre.

Medzi najznámejšie syntetizátory zo 60. rokov patria modulárne systémy *Moog* a *Buchla*, zložené z voľne prepojiteľných častí – oscilátorov, filtrov, obálok, sekvencerov a pod. V roku 1967 vyšiel modulárny syntetizátor Buchla box spolu s modulmi série 100, medzi ktorými boli aj jedny z prvých krokových sekvencerov.

Popri rýchlom vývoji syntetizátorov sa vynorili samplery – nástroje využívajúce predom nahrané zvuky. V 60. rokoch mali analógovú podobu, pričom zvuk bol prehrávaný z magnetickej pásky. Najznámejším z týchto nástrojov je *Mellotron*, ktorého prvá verzia prišla v roku 1963. Nástroj mal dve plne polyfonické klaviatúry po 35 klávesách, pričom každej klávese bola pridelená jedna páska s nahrávkou dlhou 8 sekúnd. Páska bola po stlačení klávesy uvedená do pohybu a po jej pustení následne pretočená na začiatok. Nástroj bol mechanicky zložitý a náročný na údržbu, a preto bol nahradený digitálnymi samplermi, avšak pre jeho osobitý zvuk je stále vyhľadávaný.

V 80. rokoch zažili rozmach digitálne samplery, schopné držať väčšie množstvo zvukov. Prvým digitálnym samplerom bol *Fairlight CMI*, ktorý sa skladal zo 73klávesovej klaviatúry, centrálnej jednotky s dvoma disketovými mechanikami, alfanumerickej klávesnice a monitoru so svetelným perom. Nástroj však nebol cenovo dostupný a masové rozšírenie samplerov prišlo až so systémami *E-mu*, ktoré boli značne lacnejšie a mali podobu bežných prenosných syntetizátorov. Dnešné samplery majú často softvérovú podobu a sú schopné pracovať s obrovskými databankami a pomerne verne imitovať akustické nástroje. Princíp digitálnych samplerov je bližšie popísaný v kapitole 1.6.

Obr. 1.1: Fairlight CMI, prevzaté z [1]

1.2 Sekvencer

V širšom zmysle slova je *hudobný sekvencer* programovateľný systém, ktorý umožňuje nahrávanie a prehrávanie riadiacich dát – informácií potrebných na generovanie želaného zvuku. Môže ísť o informácie o výške, intenzite alebo trvaní tónu a časová postupnosť (sekvencia) týchto informácií môže tvoriť rytmus.

Súčasťou systémov *DAW* (Digital Audio Workstation) bývajú audio sekvencery a MIDI sekvencery. Audio sekvencer je nástupcom páskového magnetofónu a slúži na digitálne nahrávanie a prehrávanie zvukových dát. MIDI sekvencer namiesto zvukových dát pracuje s MIDI dátami (viď kapitolu 1.7) nesúcimi detailné informácie o hudobnom prejave, pričom je schopný tieto dáta zaznamenávať v reálnom čase alebo ich umožňuje zadávať ručne, a to v rámci veľmi jemného časového delenia [2].

MIDI sekvencery sú riadené časovačom, ktorého interval sa udáva v tikoch za štvrťovú dobu (*Pulses Per Quarter Note* – PPQN). Interval teda závisí na tempe a určuje rozlíšenie sekvenceru. Bežne používaným minimálnym rozlíšením je 24 PPQN, no pre zachytenie rytmických nuansí hudobného prejavu sa používajú aj vyššie rozlíšenia.

Táto práca sa zameriava na *krokové sekvencery*, pochádzajúce z čias analógových syntetizátorov. Ich výhodou je možnosť intuitívneho zadávania rytmu a preto sa stále používajú napríklad na tvorbu bicích partov v systémoch DAW.

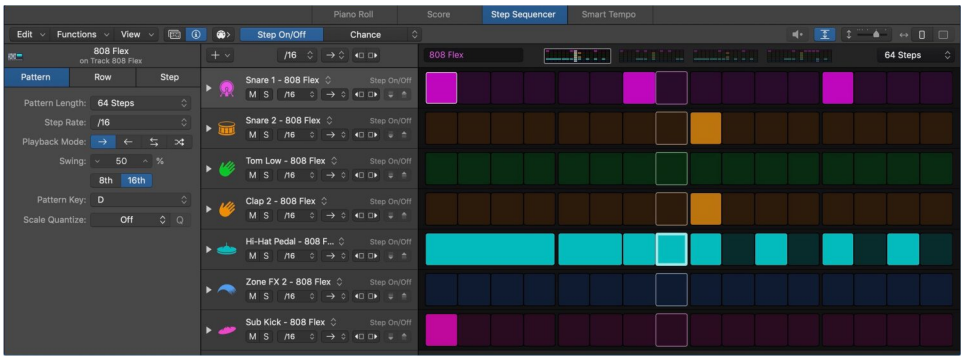
Krokový sekvencer

Krokový sekvencer (angl. *step sequencer*) vytvára tzv. *patterny* typicky o dĺžke 16 krokov[1] (nôt) a vyskytujú sa varianty s 8, 12, 16, 24 alebo 32 krokmi. Tieto

[1]Stačí teda rozlíšenie 4 PPQN.

patterny sekvencer prehráva plynule v slučke [2].

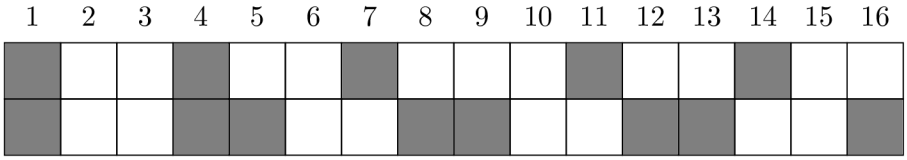
Tento typ sekvenceru môže mať podobu modulu syntetizátoru, býva súčasťou bicích automatov a grooveboxov, prípadne môže byť samostatným kusom hardvéru alebo softvéru. Moderné digitálne krokové sekvencery bežne umožňujú polyfóniu, nutne sa neviažu na určitý počet krokov a dĺžku slučky je možné v niektorých prípadoch meniť. Dnešné krokové sekvencery bežne pracujú s protokolom MIDI.



Obr. 1.2: Krokový sekvencer DAW Logic Pro

TUBS

Štandartná notácia nemusí byť vždy najvhodnejším spôsobom pre zápis hudby. *Time Unit Box System (TUBS)* je alternatívny systém notácie umožňujúci intuitívnu a prehľadnú reprezentáciu rytmu. Zaviedol ho Philip Harland v roku 1962 a je často používaný v etnomuzikológii. TUBS rozdeľuje rytmický celok na diely, ktoré predstavujú rovnaké časové úseky. Jedna z možných variánt tohoto systému je uvedená na obrázku 1.3. Tento systém sa zhoduje s princípom krokového sekvenceru a pattern zobrazený na bežnom rozhraní sekvenceru je de facto reprezentáciou rytmu v TUBS [3].



Obr. 1.3: Jednoduchý bossa nova pattern v TUBS

1.3 Polyrytmy

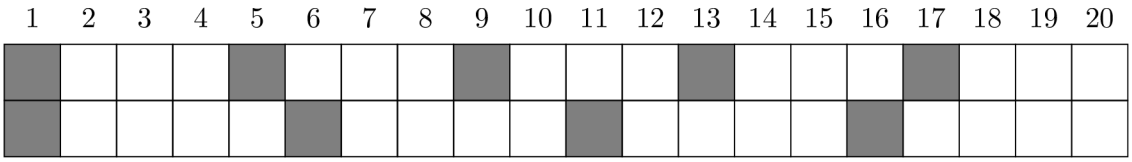
Pojem *polyrytmus* môžeme chápať ako superpozíciu rozdielnych rytmov alebo metier [4]. Polyrytmus vzniká, ak máme aspoň dva súčasne znejúce hlasy, ktoré rozdeľujú

rytmický celok na rôzne počty dielov, ktoré nie sú vzájomným celočíselným násobkom. Podľa kontextu tento jav môžeme chápať aj ako *polytempo* – rôzne party hrané v rozdielnych tempách v určitom pomere. Pod polyrytmom sa ale väčšinou rozumie opakujúca sa štruktúra v rámci rytmického celku. Bežne sa značia ako pomer v tvare $X : Y$.

Asi najbežnejším polyrytmom je tzv. hemiola, ktorá zodpovedá pomeru 3 : 2. Je základom hudby subsaharskej Afriky a obecne sa v hudbe bežne vyskytuje vyjadrená ako triola proti dvom osminovým notám alebo ako synkopácia v rámci trojštvrťového taktu. Polyrytmy jednoduchších pomerov sa bežne objavujú v artificiálnej aj nonartificiálnej hudbe. Polyrytmy zložitejších pomerov[2] (prípadne aj zložených pomerov pri viac ako dvoch rytmoch – napr. 5 : 4 : 3) sú ale znakom experimentálnej hudby.

Pre presné vyjadrenie polyrytmu n rytmov $a_1 : a_2 : \ldots : a_n$ je potrebné rytmický celok rozdeliť na m častí, kde m je najmenší spoločný násobok a_1 až a_n. Jednotlivé rytmy je potom možné rozložiť rovnomerne, teda po $\frac{m}{a_i}$ dieloch.

Bežné krokové sekvencery teda neumožňujú prehrávanie polyrytmov – pre polyrytmus 5 : 4 je potrebných $\mathrm{nsn}(5, 4) = 20$ krokov (prípadne celočíselné násobky). Na prehranie polyrytmu 5 : 4 : 3 by bolo treba 60krokový sekvencer, resp. rozlíšenie 15 PPQN. Pri zložitejších pomeroch môžu vychádzať počty krokov zodpovedajúce rozlíšeniam väčším aké je človek schopný rozoznať alebo aké je možné technicky realizovať[3].



Obr. 1.4: Polyrytmus 5:4

1.4 Hudobné intervaly a ladenia

Keďže sa táto práca okrem iného zaoberá mikrointervalikou, ktorá bude využitá aj v nástroji navrhnutom v praktickej časti, je vhodné najprv objasniť problematiku hudobných intervalov a ladení.

[2]Ide o subjektívnu zložitosť, ktorá by sa dala vyjadriť aj ako veľkosť členov pomeru, pokiaľ je pomer v základnom tvare. Napríklad pomer 4 : 3 je jednoduchší ako 11 : 9.

[3]Napríklad najmenší spoločný násobok členov pomeru 19 : 17 : 13 : 11 : 9 : 5 je 2078505, teda zodpovedajúce rozlíšenie v rámci štvorštvrťového taktu by bolo až 519626 PPQN, čo pri bežných tempách ďaleko presahuje ľudské vnímanie rytmu.

V hudobnej teórii sa pod pojmom interval rozumie vzdialenosť medzi dvoma tónmi. Veľkosť intervalu sa dá popísať dvoma spôsobmi – *frekvenčným pomerom* alebo pomocou jednotky *cent*.

1.4.1 Prirodzené ladenia

Prirodzené (čisté) ladenia sú také, ktoré obsahujú len intervaly, ktoré sa dajú popísať celočíselným pomerom frekvencií. Vychádzajú z harmonického radu – frekvenčné pomery zodpovedajú pomerom vyšších harmonických voči fundamentu.

1.4.2 Temperované ladenia

S vývojom klávesových a dychových nástrojov, ktoré nedovoľujú jednoduchú zmenu ladenia, začali vznikať kompromisné systémy ladenia – *temperatúry*, ktoré umožňujú hru v rôznych tóninách. Temperovanie (vyrovnávanie) intervalov bolo najskôr nerovnomerné, teda niektoré intervaly zostali v prirodzenom ladení a niektoré boli temperované [5].

Dvanásťtónové rovnomerne temperované ladenie (12-TET) je v súčasnosti najpoužívanejším ladením v západnej hudbe. Je v ňom zachovaný jediný čistý interval – oktáva. Ladenie rozdeľuje oktávu na 12 poltónov. Pre vyjadrenie malých odchýliek ladení sa používa Ellisovo delenie oktávy na 1200 centov. Jeden temperovaný poltón zodpovedá 100 centom. Cent môžeme vyjadriť ako frekvenčný pomer výrazom

$$1\,\text{cent} = \sqrt[100]{\sqrt[12]{2}} = \sqrt[1200]{2} \approx 1,00057779\,. \tag{1.1}$$

Veľkosť intervalu v centoch z frekvenčného pomeru naopak dostaneme pomocou

$$n = 1200 \cdot \log_2 \frac{b}{a}\,. \tag{1.2}$$

1.4.3 Mikrointervalika

Pojem mikrointervalika označuje používanie intervalov menších alebo, paradoxne, väčších[4] ako temperovaný poltón. Prakticky to znamená použitie ladenia iného ako 12-TET. Mikrointervalové ladenia nemusia byť temperované a vzťahy medzi susednými tónmi môžu tvoriť nerovnomerné intervaly. Patria sem prirodzené ladenia, rôzne nezápadné ladenia (napr. indické rozdelenie oktávy na 22 šruti), nerovnomerne temperované ladenia, rovnomerné temperácie iného počtu tónov ako 12 a pod. [6]

[4]Pojem makrointervalika sa však štandardne nepoužíva.

1.5 Mikroladenie softvérových hudobných nástrojov

Napriek tomu, že väčšina elektronických a softvérových hudobných nástrojov pracuje s dvanásťtónovým rovnomerne temperovaným ladením, niektoré nástroje umožňujú použitie iných ladení – mikroladenie. Existuje viacero spôsobov prelaďovania virtuálnych softvérových nástrojov. Preladenie väčšinou dosiahneme importom tabuľky ladenia – súboru, v ktorom je uložené dané ladenie, prípadne aj s priradením jednotlivých tónov klávesom (MIDI notám). Medzi najrozšírenejšie formáty tabuliek ladení patria [7]:

- súbory softvéru Scala (`.scl` a `.kbm`),
- AnaMark TUN (`.tun`),
- MIDI Tuning Standard (MIDI SysEx dáta, súbor `.mid`).

Scala

Softvér Scala je rozsiahly nástroj pre experimentovanie s hudobnými ladeniami. Podporuje vytváranie, editáciu, porovnávanie, analýzu a ukladanie hudobných ladení, resp. stupníc.

Pracuje s dvoma primárnymi formátmi súborov – `.scl`, v ktorom je popísaná stupnica, a `.kbm`, ktorý popisuje rozloženie stupnice na klaviatúre (MIDI mapovanie). Tieto formáty podporuje mnoho softvérových hudobných nástrojov. Oba formáty sú textové súbory čitateľné ľudským okom.

Jeden súbor typu `.scl` môže obsahovať iba jednu stupnicu. Prvý riadok súboru obsahuje krátky popis stupnice. Druhý riadok popisuje počet stupňov stupnice a zároveň značí počet nasledujúcich riadkov, ktoré vyjadrujú intervaly jednotlivých stupňov vzhľadom na základný tón. Základný tón (interval 1/1) je implicitný. Ak riadok obsahuje bodku, značí to, že interval je v centoch, a v opačnom prípade ide o zlomok. Znak `!` na začiatku riadku značí komentár [8]. Príklad `.scl` súboru je uvedený na obrázku 1.5.

Ak použijeme iba `.scl` súbor, tóny stupnice sú na klaviatúre zoradené lineárne, čo v prípade iného počtu stupňov ako dvanásť spôsobí, že na dvanásťtónovej klaviatúre nie je zachované rovnomerné rozloženie intervalov. Preto je vhodné použiť aj súbor typu `.kbm`, ktorý pomocou MIDI mapovania dovoľuje prirodzenejšie rozloženie. Štruktúra súboru je uvedená v prílohe A.

AnaMark tuning file format (TUN)

Formát TUN vyvinul Mark Henning, vývojár VSTi syntetizátoru AnaMark. V súčasnosti patrí medzi najčastejšie formáty používané softvérovými nástrojmi. Má

```
! Pythagorean 432Hz.scl
!
Pythagorean 432Hz
 12
!
 256/243
 9/8
 32/27
 81/64
 4/3
 1024/729
 3/2
 128/81
 27/16
 16/9
 4096/2187
 2/1
```

Obr. 1.5: Príklad .scl súboru

rozsiahle možnosti popisu ladenia a MIDI mapovania, pričom na to oproti formátom Scala stačí len jeden súbor typu `.tun`. Umožňuje aj uloženie viacerých ladení v jednom súbore typu `.msf`. Ide o textové súbory čitateľné ľudským okom. Oproti formátom Scala je TUN robustnejší a obsahuje viacero sekcií. Okrem iného je implementovaný popis ladenia pomocou vzorcov (funkcionálne ladenie). Formát je plne popísaný v [9].

MIDI Tuning Standard (MTS)

MIDI Tuning Standard je špecifikácia mikroladenia MIDI nástrojov, umožňujúca vysoké rozlíšenie (0,0061 centov). Špecifikáciu vyvinuli skladatelia mikrotonálnej hudby Robert Rich a Carter Scholz. Je odsúhlasená MIDI Manufacturers Association a je súčasťou MIDI špecifikácie už od 90. rokov. Je implementovaná pomocou zvláštnych systémových MIDI správ (SysEx), takže okrem načítania ladení zo súboru umožňuje aj komunikáciu s ďalšími MIDI zariadeniami [7]. Umožňuje hromadné mikroladenie (systémová správa `MTS BULK TUNING DUMP`) alebo mikroladenie jednotlivých nôt (systémová správa `MTS KEY-BASED TUNING DUMP`). Keďže ide o MIDI dáta, nejde o formát čitateľný ľudským okom.

1.6 Sampling

Sampling v kontexte elektroakustickej hudby znamená zhotovenie digitálnej nahrávky relatívne krátkeho zvuku [10]. Pojem vychádza z anglického termínu pre vzorkovanie signálu. Hudobné nástroje využívajúce sampling (*samplery*) fungujú na báze prehrávania predom nahraného zvuku (*samplu*), s prípadnou možnosťou zmeny výšky tónu zvuku – teda možnosťou využiť jeden „nasamplovaný“ zvuk pre viac tónov.

1.6.1 Analógovo-digitálny prevod zvukových signálov

So samplingom úzko súvisí analógovo-digitálny prevod, ktorý spočíva v prevode spojitého signálu na diskrétny signál, ktorý je potom možné digitálne spracovávať. Prebieha v troch etapách:

- vzorkovanie,
- kvantovanie,
- kódovanie.

Vzorkovanie

Spojitý signál je prevedený na sled vzoriek diskrétnych v čase vzorkovaním. Časové rozlíšenie diskrétneho signálu určuje *vzorkovacia frekvencia* – f_{vz}. Pre vzorkovanie zvukového signálu sú bežne používané vzorkovacie frekvencie 44,1 kHz (používaná v CD), 48 kHz (prevzatá z digitálneho záznamu na pásku DAT a bežne používaná v štúdiovej technike), prípadne ich celočíselné násobky.

Vzorkovacia frekvencia je určená na základe tzv. *vzorkovacej podmienky*[5]

$$f_{vz} > 2f_{max}\,, \tag{1.3}$$

kde f_{max} je maximálna frekvencia vzorkovaného signálu. Počuteľné spektrum je ohraničené frekvenciami 20-20000 Hz a teda je vhodné použiť na vzorkovanie zvukového signálu vzorkovaciu frekvenciu vyššiu ako 40000 Hz. Význam vzorkovacej podmienky je zabránenie *aliasingu*. Tento jav znamená skreslenie signálu v dôsledku periodizácie spektra – pri nedodržaní vzorkovacej podmienky dochádza k prekrývaniu susedných spektier. Aby bolo možné podmienku dodržať, musí byť vzorkovaný signál obmedzený filtrom typu dolná priepusť, ktorý zaručí, aby signál nepresiahol maximálnu frekvenciu. Dolné priepusti ale nebývajú ideálne, a preto sú niekedy používané vyššie vzorkovacie frekvencie ako 96 kHz alebo 192 kHz.

[5] Tiež známa ako *Nyquistova*, *Shannonova* alebo *Kotelnikova* podmienka.

Kvantovanie

Kvantovanie je prevod časovo diskrétnych vzoriek so spojitou úrovňou na vzorky s diskrétnou úrovňou. Rozlíšenie úrovne udáva *bitová hĺbka* – počet kvantizačných hladín. Vo zvukovej technike sa používa rozlíšenie 16, 20 a 24 bitov. Odchýlka nakvantovaného signálu od pôvodného sa nazýva kvantovací šum.

Kódovanie

Kódovanie je prevod vzoriek diskrétnych v čase aj v úrovni na binárne čísla. Počet bitov na zápis čísla udáva bitová hĺbka. Pre digitálnu reprezentáciu zvukového signálu sa v profesionálnej zvukovej technike používa výhradne metóda PCM (Pulse Code Modulation).

Táto časť práce čerpala zo zdroja [11].

1.6.2 Zmena výšky tónu

Akustický nástroj najvernejšie napodobíme nasamplovaním každého tónu zahraného v rôznych dynamikách, prípadne rôznymi technikami rozoznievania. To zahŕňa namáhavú prácu a vyžaduje uchovávanie veľkého množstva dát. Pre ušetrenie pamäte sa v starších sampleroch používala technika zónovania, pri ktorej bol použitý jeden sample napríklad pre interval tercie. Dnešné samplery si môžu dovoliť používať pre každý tón aj viac ako jeden sample a prepínať medzi nimi na základe dynamiky. Stále je však používaný aj variant zónovania, keďže je praktický, najmä ak sú pri hudobnej produkcii namiesto dostupných databánk využité vlastné sample. Úlohou sampleru nutne nemusí byť imitácia tradičných nástrojov a „neprirodzenosť“ je niekedy dokonca žiadaná.

Samplery teda bežne umožňujú použiť jeden sample pre viac tónov, a to zmenou výšky tónu samplu (*transpozíciou*). Dve jednoduché metódy zmeny výšky tónu sú:

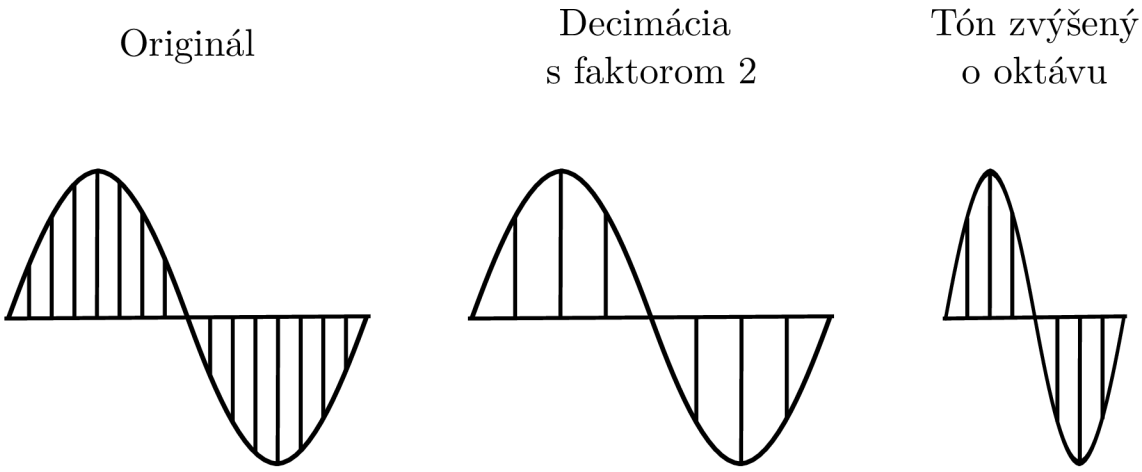
1. Zmena prehrávacej vzorkovacej frekvencie
2. Prevzorkovanie signálu

Keďže sa týmto spôsobom mení rýchlosť prehrávania samplu, tieto metódy pracujú v časovej doméne. Zmenou rýchlosti sa mení aj dĺžka samplu.

Prvá metóda mení vzorkovaciu frekvenciu prehrávania, a preto pri polyfónii vyžaduje osobitný D/A prevodník pre každú notu. Zväčšením frekvencie je výška výsledného tónu vyššia a naopak.

Druhá metóda dovoľuje transpozíciu pri konštantnej prehrávacej vzorkovacej frekvencii. Zvýšenie tónu dosiahneme podvzorkovaním. Napríklad signál s polovičnou vzorkovacou frekvenciou ako pôvodný bude znieť o oktávu vyššie – viď obrázok 1.6. Pri podvzorkovaní sú niektoré vzorky vynechané. Tento proces sa nazýva *de-*

cimácia. Pri polovičnej vzorkovacej frekvencii je teda ponechaná každá druhá vzorka pôvodného signálu. Zníženie tónu naopak dosiahneme nadvzorkovaním. Pri nadvzorkovaní je potrebné medzi pôvodné vzorky vložiť nové vzorky. Tento proces sa nazýva *interpolácia*.



Obr. 1.6: Ilustrácia zvýšenia tónu o oktávu podvzorkovaním, prevzaté z [10]

Aby bol sample transponovaný na želanú frekvenciu f_2, je potrebné poznať pôvodnú základnú frekvenciu tónu f_1. Pomer prevzrorkovania (transpozičný faktor) je potom získaný ako frekvenčný pomer

$$t = \frac{f_2}{f_1}\,, \tag{1.4}$$

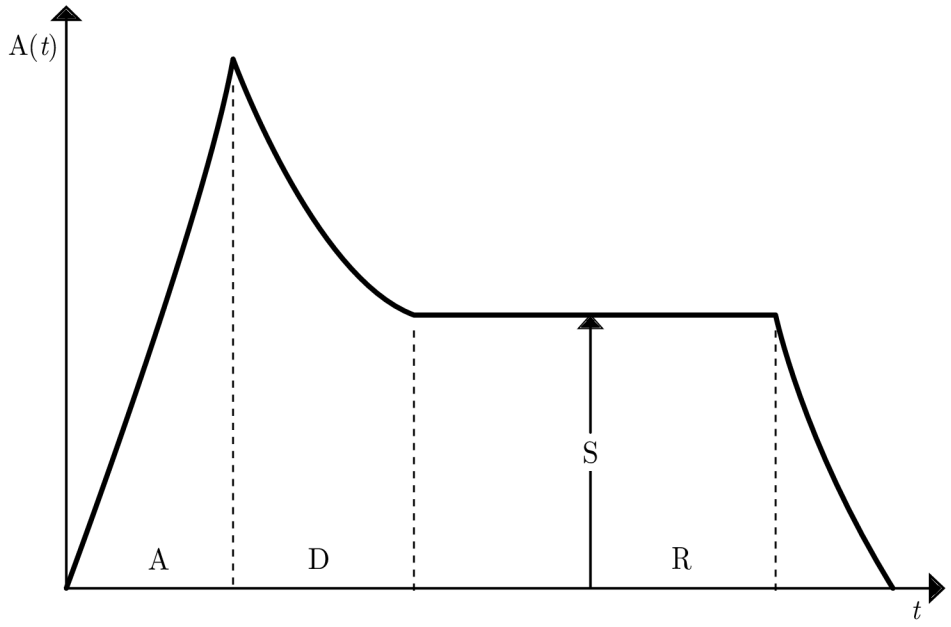
za predpokladu, že vzorkovacia frekvencia samplu sa zhoduje s výstupnou vzorkovacou frekvenciou.

Transpozícia v časovej doméne znamená okrem posunu základnej frekvencie aj posun alikvót a teda mení farbu zvuku. Táto technika je ale v sampleroch bežne používaná a transpozícia vo frekvenčnej doméne je zložitejšou problematikou. [10]

Problémy prevzorkovania

Decimácia a interpolácia môže viesť ku skresleniu signálu. Vynechanie vzoriek znamená väčší skok medzi susednými vzorkami. Zvýšenie frekvencie signálu (resp. zníženie vzorkovacej frekvencie) zase môže viesť k porušeniu vzorkovacej podmienky (1.3) a tým pádom dôjde k aliasingu. Tieto problémy je možné potlačiť filtráciou.

K aliasingu tiež môže dôjsť lineárnou interpoláciou. Existujú metódy interpolácie, ktoré potlačujú aliasing. Avšak, bežne je v sampleroch využitá lineárna interpolácia skombinovaná s filtráciou pre potlačenie aliasingu [10].



Obr. 1.7: Priebeh ADSR obálky

1.6.3 Časová obálka

Pre zaručenie čistého začiatku a konca samplu bez nežiadúcich skokov je vhodné v sampleri použiť časovú obálku. Časová obálka sa dá obecne vyjadriť ako modulačná funkcia upravujúca amplitúdu signálu. Pre výsledný signál $s(t)$ platí vzťah

$$s(t) = A(t) \cdot g(t)\,, \tag{1.5}$$

kde $A(t)$ je modulačná funkcia a $g(t)$ je modulovaný signál.

Pre účely zvukovej syntézy sa bežne používa tzv. *ADSR* obálka, ktorá má štyri fázy: A – attack, D – decay, S – sustain a R – release. Hodnoty A, D a R sú časové, S značí úroveň – viď obrázok 1.7. Attack znamená čas, za ktorý signál dosiahne maximálnu hodnotu, decay čas ustálenia na hodnotu zakmitaného stavu, sustain úroveň zakmitaného stavu a release čas dokmitávacích pochodov [5].

1.7 Komunikačné rozhranie MIDI

Protokol *MIDI* (Musical Instruments Digital Interface) je univerzálne uznávaným hudobným komunikačným štandardom už od 80. rokov. Jeho pôvodným účelom je možnosť univerzálneho prepojenia elektronických hudobných nástrojov, avšak okrem toho poskytuje efektívny spôsob ukladania informácií o hudobnom prejave. Je podporovaný všetkými bežnými operačnými systémami a je použitý vo väčšine dnešných elektronických hudobných nástrojoch a systémoch DAW. Výhodou záznamu

hudobných informácií ako MIDI dáta je možnosť jednoduchej editácie, kvantizácie, transpozície, zmeny tempa a pod.

Protokol MIDI používa sériový asynchrónny dátový prenos, pričom sa jeden rámec skladá z jedného startbitu, ôsmich dátových bitov a jedného stopbitu. Jeden fyzický MIDI kanál zahŕňa šestnásť logických kanálov.

MIDI dáta pozostávajú z *MIDI správ*. Tie tvoria stavové a dátové bity. Stavový bit obsahuje identifikátor typu správy a identifikátor MIDI kanálu, na ktorý je správa poslaná. Dátové bity obsahujú samotné dáta a ich počet závisí na type správy. Protokol MIDI pozná následujúce typy správ:

- Kanálové správy
  - Nota zapnutá (Note On)
  - Nota vypnutá (Note Off)
  - Polyfónna tlaková citlivosť (Polyphonic Key Pressure)
  - Zmena kontroleru (Control Change)
  - Zmena programu (Program Change)
  - Kanálová tlaková citlivosť (Channel Pressure)
  - Ohýbanie tónu (Pitch Bend Change)
- Systémové správy
  - Zvláštne systémové dáta (System Exclusive Message)
  - Spoločné systémové dáta (System Common Message)
  - Systémové dáta reálneho času (System Real Time Message)

Kanálové správy obsahujú dva dátové byty – napr. v správe Note On prvý dátový byte značí číslo noty a druhý dynamiku (resp. rýchlostné dáta – velocity). Systémové správy sa vzťahujú na celé zariadenie, a preto v stavovom byte nemajú identifikátor kanálu. Štandard MIDI 1.0 je plne popísaný v zdroji [12], z ktorého táto kapitola čerpá.

1.8 Technológia VST

Systém VST (*Virtual Studio Technology*) vyvinutý firmou Steinberg poskytuje formát zásuvných (plug-in) modulov určených pre digitálne spracovanie signálov v reálnom čase priamo v centrálnej procesorovej jednotke počítača. Formát je podporovaný vo väčšine systémov DAW a je najčastejšou voľbou pre tvorbu softvérových zvukových efektov alebo virtuálnych softvérových hudobných nástrojov.

Systém VST verzie 3 podporuje dynamický počet vstupných a výstupných kanálov a disponuje dvoma vstupnými zbernicami pre zvukový signál. Zvukové dáta sú spracovávané ako 32bitové alebo 64bitové čísla s pohyblivou rádovou čiarkou. Sú podporované dva typy zásuvných modulov – efektový modul so zvukovým vstupom

aj výstupom a generátor zvukového signálu (VST Instrument, VSTi) iba s výstupom.

VST plug-in modul má oddelený proces spracovania zvukových dát – *procesor* – a proces riadenia parametrov a grafického uživateľského rozhrania – *editor*. Editor tiež slúži na abstrakciu štruktúry plug-in modulu. Parametre nie sú medzi editorom a procesorom predávané priamo, ale prostredníctvom hostiteľskej aplikácie. To umožňuje napr. automatizáciu riadenia parametrov v DAW. Z hostiteľskej aplikácie je možné prenášať aj tzv. udalosti, čo môžu byť napr. MIDI správy. VST plug-in modul teoreticky nemusí obsahovať vlastné uživateľské rozhranie a jeho parametre je možné riadiť iba z hostiteľskej aplikácie. [11]

1.9 JUCE framework

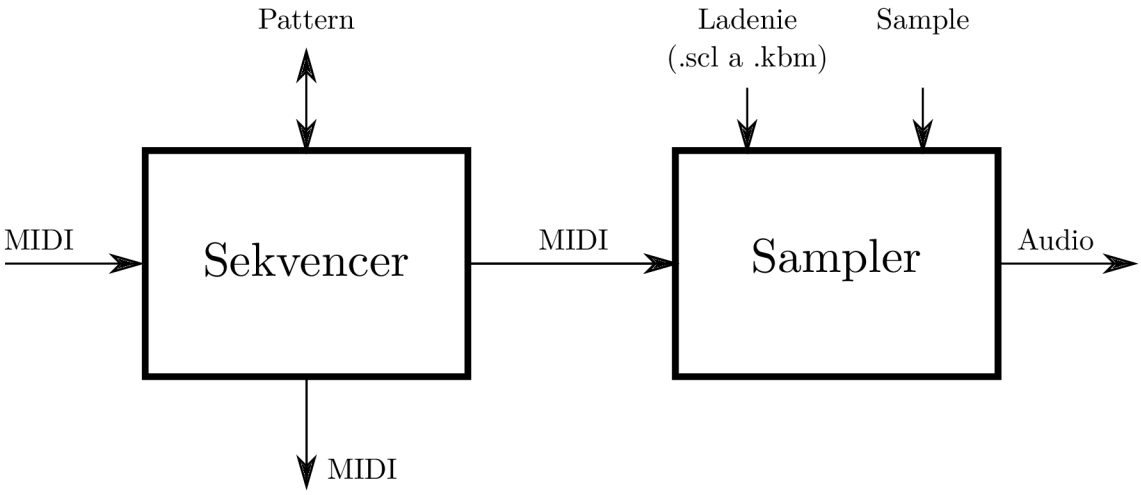
JUCE je aplikačný framework pre vývoj multiplatformového softvéru v jazyku C++. Zahŕňa veľké množstvo tried uľahčujúcich riešenie bežných softvérových záležitostí ako tvorba grafických rozhraní, správa súborov, prístup k sieti a pod. Vďaka obsiahlej podpore v oblasti zvuku je populárnou voľbou pre vývoj audio aplikácií a zásuvných modulov. JUCE obsahuje mnoho modulov pre syntézu a spracovanie zvuku alebo prácu so zvukovými súbormi a MIDI dátami. Podporuje všetky bežné desktop (Windows, Mac OS, GNU/Linux) a mobilné (Android, iOS) platformy. Tiež podporuje zásuvných modulov formátov VST, VST3, AU, RTAS a AAX. Z jedného zdrojového kódu je možné zostaviť aplikáciu pre rôzne platformy alebo zostaviť rôzne formáty zásuvných modulov [13].

Generovanie projektov podporovaných vývojových prostredí[6] uľahčuje aplikácia *Projucer*. Projucer sa postará o záležitosti spojené so zostavovaním aplikácie a generovaním šablón zdrojových súborov na základe zvolených atribútov určenia aplikácie (napr. či ide o audio aplikáciu, či má MIDI vstup a pod.) a cieľových platforiem.

[6]Vývojové prostredia s aktuálne najlepšou podporou sú *Microsoft Visual Studio* na Windows a *Xcode* na Mac OS.

2 Praktická časť

2.1 Návrh nástroja



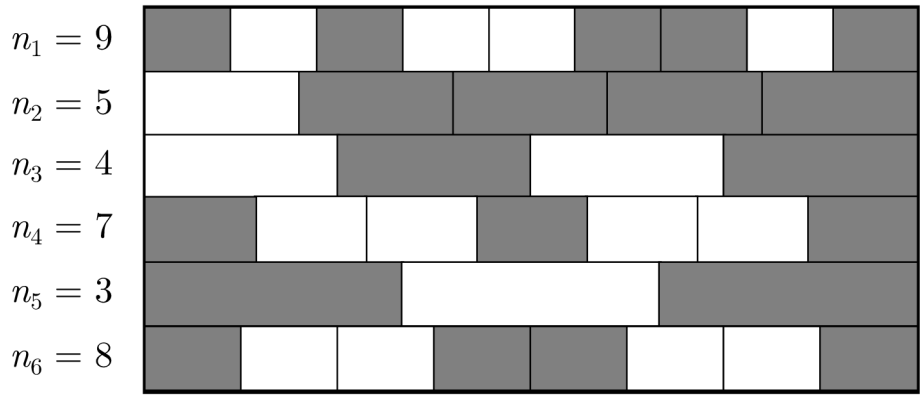
Obr. 2.1: Schéma základného princípu nástroja

Výsledný nástroj má podobu VSTi plug-in modulu pozostávajúceho z dvoch hlavných častí – sekvenceru a sampleru.

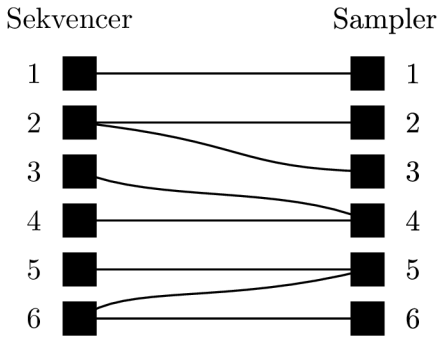
Sekvencer má šesť hlasov, pričom každý môže mať ľubovoľný počet krokov obmedzený maximálnou hodnotou. Sekvencie jednotlivých hlasov pozostávajú z nôt, resp. pomlčiek. Vlastnosti noty sú výška tónu (pitch) a intenzita (velocity). Pre dodanie variácií a náhodnosti je zavedená tretia vlastnosť – pravdepodobnosť – ktorá určuje s akou pravdepodobnosťou bude konkrétna nota zahraná. Noty je do sekvenceru okrem myši alebo klávesnice možné zadávať aj prostredníctvom MIDI zariadenia. Sekvencer má tiež možnosť transpozície patternu v reálnom čase pomocou MIDI klaviatúry. Nástroj tým nadobúda variabilitu a nemusí sa spoliehať na repetíciu. Výstup sekvenceru (MIDI dáta) je predaný sampleru a zároveň ho posiela plug-in modul na svoj MIDI výstup – vzniká teda možnosť použiť sekvencer s inými zdrojmi zvuku.

Sampler má zhodný počet hlasov. Pre každý hlas je možnosť nahrať osobitný sample. Mikroladenie sampleru je dosiahnuté nahraním dvojice `.scl` a `.kbm` súborov, ktoré priradia MIDI notám konkrétne frekvencie. Na základe nich sa mení výška tónov všetkých samplov.

Vo východzom stave zodpovedá každý hlas sekvenceru súhlasnému hlasu sampleru. Okrem toho je možnosť hlasy ľubovoľne prepojiť. Jedna sekvencia teda môže



Obr. 2.2: Návrh rozhrania sekvenceru



Obr. 2.3: Možnosti prepojenia sekvenceru a sampleru

ovládať viacero samplov a jeden sample môže byť ovládaný viacerými sekvenciami – viď obrázok 2.3.

2.2 Realizácia nástroja

2.2.1 Relevantné triedy frameworku JUCE

Nasledujúce triedy frameworku JUCE uľahčujú implementáciu navrhnutého nástroja. Ich plný popis je obsiahnutý v oficiálnej dokumentácii JUCE [14].

`MidiMessage`

Trieda `MidiMessage` v JUCE reprezentuje MIDI správu. Dôležitá je najmä statická metóda `MidiMessage noteOn(int channel, int noteNumber, float velocity)` vytvárajúca MIDI správu Note On, ktorá umožní sekvenceru zahrať danú notu, a tiež metóda `MidiMessage noteOff(int channel, int noteNumber)`, ktorá danú notu vypne.

`MidiBuffer`

Trieda `MidiBuffer` drží sekvenciu časovo zoradených MIDI udalostí. Pomocou metódy `addEvent()` je možné pridávať objekty typu `MidiMessage`, no trieda ďalej pracuje už len so surovými MIDI dátami.

`AudioBuffer`

Trieda `AudioBuffer` predstavuje viackanálový buffer vzoriek zvukového signálu zapísaných pohyblivou rádovou čiarkou.

`Synthesiser`

Trieda `Synthesiser` reprezentuje abstrakciu hudoného zariadenia tvoriaceho zvuk. Na základe prijatých MIDI správ riadi objekty generujúce zvuk. Generovanie samotného zvuku prebieha až v triede `SynthesiserVoice`, ktorá je pridaná objektu `Synthesiser` pomocou metódy `addVoice()`. Zvukové dáta sú získané zavolaním metódy `renderNextBlock()`, ktorej je ako argument predaná referencia na `MidiBuffer` so vstupnými dátami a referencia na výstupný `AudioBuffer`.

`SynthesiserSound`

Trieda `SynthesiserSound` reprezentuje zvuk syntetizátoru, ktorý je prípadne obmedzený na určité MIDI noty alebo kanály. Obmedzenie je dosiahnuté implementovaním virtuálnych metód `appliesToNote()` a `appliesToChannel()`. Rozšírením tejto triedy môžu byť pridané ďalšie informácie o danom zvuku.

`SynthesiserVoice`

Trieda `SynthesiserVoice` reprezentuje jeden hlas nástroja, ktorý môže objekt `Synthesiser` použiť pre prehranie konkrétneho zvuku určeného objektom `SynthesiserSound`. Aby bol nástroj polyfonický, musí byť objektu `Synthesiser` priradených viacero objektov `SynthesiserVoice`. Metódy triedy `SynthesiserVoice` sú zväčša virtuálne, a preto je potrebné od triedy dediť a tieto metódy implementovať.

`SamplerSound`

Trieda `SamplerSound` je potomkom triedy `SynthesiserSound` a pridáva funkcionalitu potrebnú pre implementáciu sampleru. Uchováva zvukový súbor, drží informáciu o referenčnej MIDI note a tiež obsahuje ADSR obálku s definovateľnými parametrami *attack* a *decay*.

`SamplerVoice`

Trieda `SamplerVoice` dedí od triedy `SynthesiserVoice` a má implementovanú funkcionalitu prehrávania a transpozíciu zvuku definovaného objektom `SamplerSound`. Na prevzorkovanie používa jednoduchú lineárnu interpoláciu.

`AudioProcessor`

Trieda `AudioProcessor` predstavuje procesorovú časť audio aplikácie, resp. zásuvného modulu. Je možné ju použiť v rôznych „wrapperoch“ predstavujúcich rôzne formáty zásuvných modulov, prípadne samostatnú („standalone“) aplikáciu. Audio aplikácia má implementovanú vlastnú triedu, ktorá je potomkom triedy `AudioProcessor`. Spracovanie audio a MIDI dát sa deje v metóde `processBlock()`, ktorá má ako argumenty referencie `AudioBuffer&` a `MidiBuffer&`. V bufferoch sú uložené vstupné dáta. Dáta, ktoré v týchto bufferoch zostanú po skončení metódy sa považujú za výstup. Operácie v tejto triede prebiehajú v samostatnom vlákne s vysokou prioritou – *audio thread*.

`AudioProcessorEditor`

Trieda `AudioProcessorEditor` je základnou triedou používateľského rozhrania aplikácie. Editor sa vzťahuje ku konkrétnemu procesoru a je vytvorený v metóde `AudioProcessor::createEditor()`. Jednotlivé prvky používateľského rozhrania pracujú a vzájomne komunikujú vo vlákne *message thread*.

2.2.2 Princíp experimentálneho sekvenceru

Pre zhotovenie krokového sekvenceru realizujúceho polyrytmy, teda sekvenceru umožňujúceho rôzny počet krokov v každom hlase v ľubovoľných pomeroch, je potrebné vyriešiť problém časovej synchronizácie. Ponúka sa možnosť použiť samostatný časovač s vlastným tempom pre každý hlas, čo ale samo o sebe nezaručuje synchronizáciu, a aby bola dosiahnutá dostatočná presnosť, je potrebný veľký výpočetný výkon – pravdepodobne by každý časovač vyžadoval samostatné vlákno.

Vhodnejšia možnosť je použiť jeden, centrálny časovač, ktorý bude ovládať všetky hlasy, ako to funguje pri bežných sekvenceroch. Interval tohoto časovača je pri m hlasoch získaný vzťahom

$$T = \frac{60 \cdot d \cdot \text{takt} \cdot 4}{\text{tempo} \cdot \text{nsn}(n_1, \ldots, n_m)} \qquad [\text{s}]\,, \tag{2.1}$$

kde n je počet krokov daného hlasu, d je dĺžka cyklu v taktoch, tempo je uvedené v BPM a takt je vyjadrený ako zlomok. Daný počet taktov sa teda rozdelí na počet

globálnych krokov, ktorý zodpovedá najmenšiemu spoločnému násobku počtov krokov jednotlivých hlasov, pričom sa sekvencia konkrétneho hlasu posunie na základe modulo operácií.

Pre implementáciu logiky sekvenceru boli vytvorené triedy `PolySequencer` a `SequencerVoice`. Trieda `PolySequencer` riadi celý sekvencer a trieda `SequencerVoice` drží sekvenciu nôt a aktuálnu pozíciu daného hlasu. Metóda `PolySequencer::tick()` je pravidelne volaná v intervale definovanom vzorcom 2.1 z metódy `processBlock()` v procesore aplikácie, pričom je časovač realizovaný počítadlom inkrementovaným každú vzorkovaciu periódu, ktoré sa vynuluje, keď dosiahne hodnotu odpovedajúcu intervalu vo vzorkách. To, či má daný hlas v istý tik sekvenceru zahrať notu a posunúť pozíciu v sekvencii sa rozhoduje nasledujúcim spôsobom:

```
bool PolySequencer::shouldPlay(SequencerVoice* v)
{
    return !(position % (steps / v->getLength()));
}
```

Teda daný hlas sa posunie, ak je pozícia globálneho kroku deliteľná počtom globálnych krokov odpovedajúcich jednému kroku daného hlasu. To, či sa daná nota zahrá, je rozhodnuté na základe pravdepodobnosti vygenerovaním náhodného čísla v metóde `MidiBuffer SequencerVoice::getNoteOn(int sample)`, ktorá pri pozitívnom výsledku vráti `MidiBuffer` obsahujúci správy *Note On* so zahranou notou na všetkých kanáloch priradených hlasu. Vrátený buffer je následne zahrnutý do bufferu sekvenceru.

Výpis 2.1: Výpočet globálnych krokov

```
int PolySequencer::calculateSteps()
{
    int result = voices[0]->getLength();
    for (int i = 1; i < NUM_VOICES; i++)
        result = math::lcm(result, voices[i]->getLength());

    int max = sampleRate * (240.0f / tempo) * duration
            * ((double)timeSignature.a / (double)timeSignature.b);

    return result > max ? max : result;
}
```

Pri zmene počtu krokov daného hlasu je potrebné zistiť najmenší spoločný násobok a prepočítať tak počet globálnych krokov, resp. interval časovača. Pri zložitých pomeroch a rýchlych tempách môže nastať situácia, keď je vypočítaný interval

menší ako jedna vzorkovacia perióda. Aby bol sekvencer schopný polyrytmus zahrať v tempe, je v tejto situácii potrebné zvoliť počet globálnych krokov zodpovedajúci najmenšiemu možnému intervalu – sekvencer tak bude pracovať v najvyššom možnom rozlíšení. Pri zmene počtu krokov je potrebné prepočítať aj pozíciu prehrávania.

2.2.3 Implementácia mikrotonálneho sampleru

Keďže transpozícia implementovaná v triede `SamplerVoice` sa viaže na MIDI noty zodpovedajúce ladeniu 12-TET, štandartná implementácia sampleru v JUCE neumožňuje mikroladenie. Preto je potrebné vytvoriť vlastnú implementáciu transpozície samplov. Boli vytvorené nasledujúce triedy založené na pôvodných triedach `SamplerVoice` a `SamplerSound`:

```
MicroSamplerSound(const String& name,
        AudioFormatReader* source,
        String path,
        int midiChannel,
        const BigInteger& midiNotes,
        double frequencyForNormalPitch,
        double attack,
        double release);

MicroSamplerVoice(std::shared_ptr<Tunings::Tuning> tuning);
```

Narozdiel od pôvodných tried sa počíta transpozičný faktor na základe frekvencií a nie MIDI nôt. Je tak možné danej MIDI note prideliť ľubovoľnú frekvenciu. Transpozičný faktor je kalkulovaný v metóde `MicroSamplerVoice::startNote()` nasledujúcim spôsobom:

```
pitchRatio = (tuning->frequencyForMidiNote(midiNoteNumber)
           / sound->rootFrequency)
           * sound->sourceSampleRate / getSampleRate();
```

Pri prevzorkovaní sa počíta aj s rozdielnymi vzorkovacími frekvenciami samplu a výstupu.

Na parsing `.scl` a `.kbm` súborov je využitá knižnica `Tunings.h` [15] z projektu *Surge*. Knižnica tiež umožňuje tvorbu ľubovoľných rovnomerne temperovaných ladení. Obsahuje metódu `frequencyForMidiNote()`, ktorá priradí danej MIDI note konkrétnu frekvenciu, vychádzajúc z ladiacej tabuľky a MIDI mapovania.

Rovnako ako v triede `SamplerVoice` je použitá jednoduchá implementácia lineárnej interpolácie (viď prílohu B). Oproti pôvodnej triede je možné nastaviť začiatok

a koniec samplu a je tiež možné prehrávanie odzadu. Parametre *attack* a *decay* je možné meniť – tie zároveň plnia funkciu „fade in“ a „fade out“ a dovoľujú tak hladký začiatok a koniec samplu.

Jedna inštancia triedy `MicroSamplerSound` sa viaže ku konkrétnemu MIDI kanálu. Do sampleru (inštancie triedy `Synthesiser`) je pridaný počet zvukov (inštancií `MicroSamplerSound`) definovaný konštantou `NUM_VOICES = 6`, teda zvuky zodpovedajúce prvým šiestim MIDI kanálom. Zložitejšie prepojenia sekvenceru a sampleru môžu viesť k veľkému množstvu súčasne hrajúcich nôt a pri dlhom nastavení parametru decay môže hrať viac nôt aj v jednom hlase – preto je počet objektov `MicroSamplerVoice` pridaných do sampleru definovaný ako $2n^2$, kde n je počet hlasov sekvenceru, resp. počet zvukov sampleru. Tieto objekty trieda `Synthesiser` využíva podľa potreby. Daný hlas môže byť využitý vtedy, ak aktuálne nehrá žiadny zvuk (nie je aktívna ADSR obálka).

2.2.4 Zahrnutie sampleru a sekvenceru do aplikácie

Riadenie sekvenceru, generovanie MIDI dát aj zvuku prebieha v metóde `processBlock`, ktorá je uvedená v prílohe C.

V metóde je najprv spracovaný MIDI vstup – podľa neho je prípadne pattern sekvenceru transponovaný (ak je táto funkcia zapnutá), pričom prioritu má prvá stlačená klávesa.

V prípade, že je aplikácia spustená ako zásuvný modul v hostiteľskej aplikácii, ktorá poskytuje synchronizáciu tempa a v zásuvnom module je táto synchronizácia zapnutá, využije sa v sekvenceri tempo a takt z hostiteľskej aplikácie – je takto možné využiť napr. automatizáciu tempa v DAW.

Ďalej je vypočítaný interval sekvenceru a v prípade, že je zapnuté prehrávanie sekvenceru, je inkrementovaný časovač a pri dosiahnutí intervalu je volaná metóda `sequencer.tick()`, s parametrom indexu aktuálne spracovávanej vzorky – ten sa využije ako časová značka („timestamp“) prípadne generovanej MIDI správy. `MidiBuffer` sekvenceru je vymenený s bufferom metódy `processBlock()` a teda sa z neho stáva výstupný buffer.

Pri vypnutí prehrávania sú na prvých 6 MIDI kanálov poslané správy Note Off pre všetky noty, aby bolo zabezpečené, že po vypnutí nezostane žiadna nota hrať.

Nakoniec je zavolaná metóda `renderNextBlock()` sampleru, ktorej je predaný `MidiBuffer` a výstupný `AudioBuffer` – podľa prijatých MIDI správ generuje zvukové dáta. Na výstupný `AudioBuffer` je aplikovaný *master gain*.

2.2.5 Grafické používateľské rozhranie

Grafické používateľské rozhranie aplikácie má dve hlavné časti – „Play“ a „Config“ (viď obrázky 2.4 a 2.5).

Na ľavej časti sekcie „Play“ je rozhranie sekvenceru. Noty je do neho možné zadávať pomocou myši – horizontálny pohyb pri držaní ľavého tlačítka na konkrétnom kroku ovláda číslo noty, vertikálny pohyb ovláda parameter velocity. Kolieskom myši je možné meniť pravdepodobnosť konkrétneho kroku. Ovládacie prvky v dolnej časti slúžia na presnejšie ladenie týchto parametrov. Noty je tiež možné zadávať pomocou MIDI klaviatúry, a to dvojklikom na konkrétny krok a následným stlačením klávesy. V danom kroku je v strede zobrazené číslo noty, horizontálna čiara vyjadruje dynamiku a priehľadnosť farby pozadia vyjadruje pravdepodobnosť.

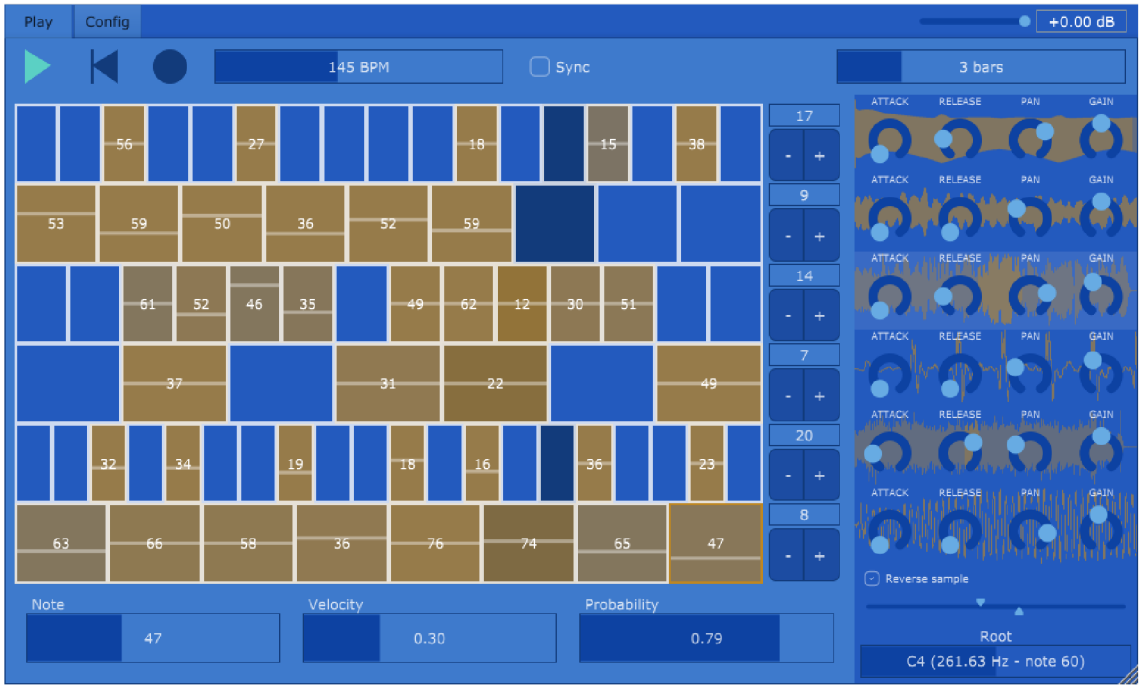
V hornej časti sa nachádzajú ovládacie prvky sekvenceru – zľava tlačítko „Play / Stop“ (prehrávanie), tlačítko „Reset“ (vrátenie pozície prehrávania na začiatok), tlačítko „Record“ (zapnutie transpozície patternu pomocou MIDI klaviatúry), ovládač tempa, prepínač „Sync“ (synchronizácia tempa s hostiteľskou aplikáciou) a ovládač dĺžky patternu v taktoch.

V pravej časti sa nachádzajú ovládacie prvky sampleru. Priradenie zvuku konkrétnemu hlasu sa dá docieliť dvojklikom na políčko hlasu a následným vybratím zvukového súboru v dialógu. Funguje tiež funkcia „Drag and drop“ a zvuk je možné priradiť pretiahnutím súboru na políčko. Pri každom zvuku je možné meniť parametre attack a decay, panorámu a hlasitosť zvuku. v dolnej časti sú ovládacie prvky, ktoré sa vzťahujú k práve vybranému samplu – prehrávanie odzadu, nastavenie začiatku a konca samplu a nastavenie referenčnej noty resp. frekvencie. Posuvný ovládač sa vo východzom stave pohybuje po notách ladenia 12TET (pretože sample jednotlivých tónov takto často bývajú označované), ale podržaním klávesy Ctrl je možné frekvenciu nastavovať jemnejšie – prípadne je možné notu alebo frekvenciu zadať ako text.

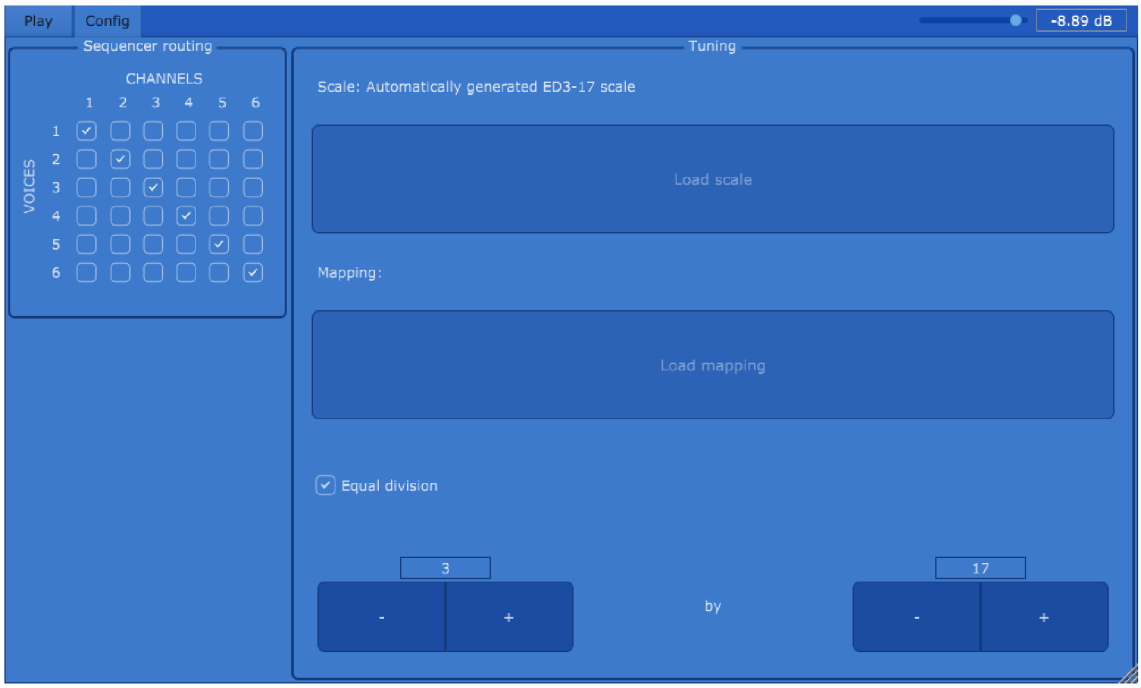
V sekcii „Config“ sa nachádza matica prepojenia sampleru a sekvenceru – riadky predstavujú hlasy sekvenceru a stĺpce zvuky sampleru, resp. kanály MIDI výstupu – a nastavenie ladenia. Tu je možné načítať `.scl` a `.kbm` súbory alebo generovať rovnomerne temperované ladenia – prvé číslo predstavuje interval, ktorý bude rozdelený (napr. 2 znamená 2/1 = oktáva), druhé číslo znamená počet rovnomerných rozdelení.

Realizácia grafického používateľského rozhrania

Základným stavebným prvkom grafického používateľského rozhrania v JUCE je trieda `Component`. Framework JUCE tiež poskytuje implementácie množstva ovládacích prvkov, ktoré sú potomkami tejto triedy. Jeden *komponent* môže obsaho-



Obr. 2.4: Sekcia „Play“ aplikácie



Obr. 2.5: Sekcia „Config“ aplikácie

vať ďalšie podradené komponenty a je tak možné stavať komplexné štrukturované používateľské rozhrania. Vzájomná interakcia komponentov a práca s používateľským vstupom je zabezpečená modelom *Broadcaster – Listener*. Trieda `Component` je potomkom triedy `MouseListener` – každý komponent teda môže implementovať tzv. *callback* metódy spracúvajúce vstup myši. Komponent môže obsahovať vnorenú triedu `Listener`, volať jej metódy a plniť tak úlohu „broadcasteru“ – informovať o zmene stavu „listenerov“ (ostatné komponenty, ktoré sú potomkami tejto vnorenej triedy).

Vzhľad komponentu je definovaný v metóde `paint()`, v ktorej je dostupný grafický kontext v podobe referencie na inštanciu triedy `Graphics`, ktorá umožňuje vykresľovanie grafiky. Metóda nie je volaná priamo, prekreslenie komponentu je dosiahnuté zavolaním metódy `repaint()`;

Rozloženie podradených komponentov je definované v metóde `resized()`, ktorá je volaná pri zmene veľkosti komponentu. Je tak možné tvoriť dynamické rozloženia grafického používateľského rozhrania, ktoré odpovedajú na zmenu veľkosti okna. S touto metódou je automaticky volaná metóda `repaint()`, pretože pri zmene veľkosti je potrebné komponenty prekresliť.

V rámci tvorby grafického používateľského rozhrania aplikácie bolo vytvorených mnoho vlastných komponentov, buď využívajúcich komponenty frameworku JUCE alebo vytvorených od základu – framework napríklad neposkytuje žiadne komponenty vhodné k vytvoreniu rozhrania sekvenceru, takže museli byť vytvorené nové komponenty s vlastnou metódou `paint()`.

2.2.6 Vnútorná štruktúra aplikácie

Pri návrhu štruktúry aplikácie bol volený objektovo orientovaný prístup. Táto podkapitola má priblížiť funkciu jednotlivých častí aplikácie. Podrobnejší zoznam najdôležitejších štruktúr, tried, ich metód a atribútov je uvedený v prílohe D.

Trieda `PolySequencer`

Predstavuje abstrakciu polyrytmyckého sekvenceru. Obsahuje metódy pre riadenie prehrávania sekvenceru, nastavenie tempa, získanie intervalu časovača, transpozície patternu a pod. Pred započatím prehrávania je potrebné upresniť použitú vzorkovaciu frekvenciu. Prehrávanie je docielené volaním metódy `tick()` v intervale vráteným metódou `getIntervalInSamples()`. Výstupné MIDI dáta sú uchované vo verejnom atribúte `midiMessages`. Hlasy sekvenceru sú prístupné prostredníctvom poľa ukazovateľov na objekty `SequencerVoice voices`.

Trieda `SequencerVoice`

Slúži ako dátová štruktúra uchovávajúca sekvenciu nôt jedného hlasu sekvenceru a umožňuje riadenie daného hlasu. Obsahuje metódy pre zmenu počtu krokov hlasu, riadenie pozície prehrávania, priraďovanie MIDI kanálov danému hlasu a pod. K jednotlivým notám sekvencie sa pristupuje prostredníctvom metódy `getNotePtr()`. Pomocou metód `getNoteOn()` a `getNoteOff()` môžu byť získané MIDI dáta relevantné pre aktuálnu pozíciu prehrávania.

Štruktúra `Note`

Dátová štruktúra uchovávajúca informácie o danom kroku, resp. note. Má atribúty `number`, `velocity` a `probability`. Prázdny krok (pomlčka) je definovaný ako štruktúra `Note` s atribútom `number = -1`.

Trieda `SequencerStep`

Grafická reprezentácia kroku sekvenceru – potomok triedy `Component`. Umožňuje zadávanie nôt pomocou myši alebo MIDI klaviatúry.

Trieda `MicroSamplerSound`

Uchováva samotný sample a jeho atribúty ako sú parametre obálky, začiatok a koniec, MIDI kanál, hlasitosť, panoráma, prehrávanie odzadu a pod.

Trieda `MicroSamplerVoice`

Táto trieda sa stará o prehrávanie samplu a zmeny jeho výšky tónu. Pri vytváraní objektu je konštruktoru predaný ukazovateľ na objekt `Tuning`[1], podľa ktorého je vypočítaný transpozičný faktor.

Trieda `SampleSource`

Grafická reprezentácia samplu – potomok triedy `Component`. Obsahuje ovládacie prvky vzťahujúce sa k objektu `MicroSamplerSound`. Pomocou triedy `AudioThumbnail` vykresľuje priebeh signálu samplu. Trieda zabezpečuje načítanie zvukových súborov a má implementovanú funkciu „Drag and drop“.

[1] Vzhľadom na spôsob konštrukcie objektu `Tuning` je využitý „smart pointer“ `std::shared_ptr` – pri nahrávaní ladenia je zakaždým vytvorený nový objekt. Ukazovateľ `std::shared_ptr` sa stará o deštrukciu nepoužívaných objektov.

2.3 Využitie nástroja

Výsledkom práce je funkčný prototyp nástroja ponúkajúceho rozsiahle hudobné možnosti. Pomocou vytvoreného sekvenceru je možné veľmi jednoducho vytvárať zaujímavé rytmické štruktúry, ktoré by inak boli len ťažko dosiahnuteľné. Nástroj môže vytvárať krátke slučky charakteristické pre repetitívnu elektronickú hudbu, ktorým ale môže dodávať veľmi veľkú rytmickú komplexnosť. Pri nastavení dlhšieho trvania slučky a väčšieho počtu krokov je možné tvoriť aj štruktúry, ktoré nepôsobia tak repetitívne. Nastavenie pravdepodobnosti jednotlivých nôt vytvára možnosť tvorby aleatorickej hudby. Tvorbu patternov sekvenceru je možné realizovať aj plynulo za prehrávania, takže je pomocou nástroja možné tvoriť hudbu naživo. Možnosť transpozície patternu prostredníctvom MIDI vstupu rozširuje možnosti živej hry a robí zo sekvenceru vlastne veľmi komplexný arpeggiátor.

Možnosť vkladania ľubovoľných samplov prináša rozsiahle zvukové možnosti – okrem konvenčných zvukov melodických hudobných nástrojov je možné vkladať napr. nahrávky ľudského hlasu, zvuky perkusívnych nástrojov, rôzne ruchy alebo aj celé hudobné skladby. Vložené zvuky je v aplikácii možné do istej miery upraviť (nastavením začiatku a konca alebo prehrávaním odzadu) a rozložiť ich do sterea. Rytmizácia vložených zvukov sekvencerom a ich prelaďovanie v rámci definovaných mikrointervalových terénov môže tvoriť zaujímavé zvukové štruktúry využiteľné ako v experimentálnej hudbe, tak aj konvenčnejšej, alternatívnej hudbe (napr. v žánroch ako IDM[2]).

Zmenou prepojenia sekvenceru a sampleru je možné z už definovaných patternov dostať nové, nečakané zvuky. Možnosť využiť MIDI výstup sekvenceru na ovládanie iných plug-in modulov ďalej rozširuje zvukové možnosti.

V rámci práce bol vytvorený nástroj v podobe VST plug-in modulu aj standalone aplikácie pre platformu Windows. Vďaka univerzálnosti frameworku JUCE by nebolo náročné vytvoriť aj verziu nástroja určenú pre mobilné platformy, na čo je vhodné aj vytvorené grafické používateľské rozhranie prispôsobené rôznym veľkostiam obrazovky. V rámci práce bol však nástroj testovaný len na platforme Windows, kde funguje najlepšie s ovládačom zvukovej karty ASIO – rytmické štruktúry je schopný prehrávať veľmi presne a jeho zvukový výstup je plynulý.

[2]Intelligent Dance Music

Záver

Cieľom tejto práce bolo vytvoriť funkčný prototyp experimentálneho softvérového hudobného nástroja kombinujúceho krokový sekvencer a sampler. Výnimočnosť nástroja spočíva v schopnosti jednoducho tvoriť aj veľmi komplikované polyrytmy, možnosti prehrávania používateľom vložených zvukov a využití rôznych mikrointervalových ladení.

Aby bolo možné nástroj realizovať, bolo potrebné najprv preskúmať oblasti ako je princíp hudobných sekvencerov, polyrytmy, hudobné ladenia, mikroladenie elektronických hudobných nástrojov a sampling. Všetky tieto témy preto boli popísané v rámci teoretickej časti práce. V tejto časti sú tiež predstavené technológie využité v praktickej časti, ako protokol MIDI, rozhranie VST a framework JUCE.

V rámci praktickej časti práce bol bližšie špecifikovaný návrh nástroja a nástroj bol aj úspešne realizovaný. Na realizáciu bol použitý programovací jazyk C++ a aplikačný framework JUCE vhodný k tvorbe audio aplikácií. K vytvoreniu výslednej aplikácie bolo nutné dôkladne študovať dokumentáciu tohoto frameworku a zistiť jeho špecifiká. Cieľom bolo čo najefektívnejšie využiť časti frameworku pre vytvorenie navrhnutého nástroja, aj tak však veľká časť programu vyžadovala vlastnú implementáciu. Pri programovaní bol volený objektovo orientovaný prístup, a teda tvorba aplikácie zahŕňala objektový návrh a implementáciu jednotlivých programových častí. V praktickej časti práce je popísaná štruktúra vytvorenej aplikácie a princíp jej súčastí.

Výsledkom práce je funkčná aplikácia, ktorá má všetky funkcie obsiahnuté v návrhu. Elektronická príloha práce obsahuje zdrojový kód a zostavený program vo forme VST3 plug-in modulu a standalone aplikácie pre Windows. Cieľ práce bol teda dosiahnutý. Aplikáciu je však možné ďalej rozširovať, pridať ďalšie funkcie (napr. podporu iných formátov ladiacich tabuliek alebo ich editáciu priamo v aplikácii), rozšíriť integráciu v DAW alebo vytvoriť verzie aplikácie aj pre iné platformy.

Literatúra

[1] *120 Years of Electronic Music* [online]. Hastings: Simon Crab, 1995-2019 [cit. 2020-12-05]. Dostupné z: <http://120years.net/>

[2] RUSS, Martin. *Sound synthesis and sampling*. 2nd ed. Amsterdam: Elsevier, 2004. ISBN 0-240-51692-3.

[3] MILLEA, Timothy A. a Jonathan P. WAKEFIELD. *Automating the Composition of Popular Music: The Search For a Hit. EvoPhD* [online]. Istanbul, 2010 [cit. 2020-12-07]. Dostupné z: <http://eprints.hud.ac.uk/id/eprint/7788/>

[4] GALVAO, Martim. *Metric Interplay: A Case Study In Polymeter, Polyrhythm, And Polytempo*. Irvine, California, USA, 2014. Dizertácia. University of California, Irvine. Dostupné tiež z: <https://escholarship.org/uc/item/8m46811d>

[5] SYROVÝ, Václav. *Hudební akustika*. 3. dopl. vyd. Praha: Akademie múzických umění, 2013. Akustická knihovna Zvukového studia Hudební fakulty AMU. ISBN 978-80-7331-297-8.

[6] Microtonal music. In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2020-12-09]. Dostupné z: <https://en.wikipedia.org/wiki/Microtonal_music>

[7] LIGON, Jacky. *Microtuning And Alternative Intonation Systems*. The MIDI Association [online]. The MIDI Association, 2017 [cit. 2020-11-24]. Dostupné z: <https://www.midi.org/midi-articles/microtuning-and-alternative-intonation-systems>

[8] *Scala scale file format* [online]. Amsterdam: Stichting Huygens-Fokker centrum voor microtonale muziek, 2020 [cit. 2020-11-24]. Dostupné z: <http://www.huygens-fokker.org/scala/scl_format.html>

[9] HENNING, Mark. AnaMark tuning file format. V2.00. Nemecko: Mark Henning, 2009. Dostupné tiež z: <https://www.mark-henning.de/files/am/Tuning_File_V2_Doc.pdf>

[10] ROADS, Curtis. *The computer music tutorial*. Cambridge, MA, USA: MIT Press, 1996. ISBN 0252181583.

[11] SCHIMMEL, Jiří. *Studiová a hudební elektronika* [online]. Druhé vydání. Brno: Vysoké učení technické v Brně, 2015 [cit. 2020-12-08]. ISBN 978-80-214-4452-2. Dostupné z: <https://www.vutbr.cz/>

[12] *The Complete MIDI 1.0 Detailed Specification: Incorporating all Recommended Practices*. Third edition. Los Angeles: The MIDI Manufacturers Association, 2014. Dostupné tiež z: <https://www.midi.org/specifications-old/item/the-midi-1-0-specification>

[13] ROBINSON, Martin. *Getting Started With JUCE*. Birmingham: Packt Publishing, 2013. ISBN 978-1-78328-331-6.

[14] JUCE Documentation. *JUCE* [online]. [cit. 2020-12-02]. Dostupné z: <https://docs.juce.com/master/index.html>

[15] *Tuning Library* [online]. Surge Synth Team, 2020 [cit. 2020-12-10]. Dostupné z: <https://surge-synth-team.org/tuning-library/>

Zoznam symbolov, veličín a skratiek

12-TET	dvanásťtónové rovnomerne temperované ladenie – 12-tone Equal Temperament
BPM	údery za minútu – Beats Per Minute
CD	Compact Disc
D/A	digitálno-analógový (prevodník)
DAT	Digital Audio Tape
DAW	Digital Audio Workstation
f_{max}	maximálna frekvencia [Hz]
f_{vz}	vzorkovacia frekvencia [Hz]
MIDI	Musical Instruments Digital Interface
MTS	MIDI Tuning Standard
PCM	Pulse Code Modulation
PPQN	Pulses Per Quarter Note
SysEx	zvláštna systémová MIDI správa – MIDI System Exclusive Message
TUBS	Time Unit Box System
VST	Virtual Studio Technology
VSTi	VST Instrument

Zoznam príloh

A Šablóna `.kbm` súboru

```
! Šablóna .kbm MIDI mapovania
!
! Veľkosť mapy. Vzorec sa opakuje každých n kláves:
12
! Prvá MIDI nota na preladenie:
0
! Posledná MIDI nota na preladenie:
127
! Prostredná MIDI nota, na ktorú je namapovaná prvá položka:
60
! MIDI nota pre referenčnú frekvenciu:
69
! Referenčná frekvencia (desatinné číslo v Hz):
440.0
! Stupeň považovaný za formálnu oktávu:
12
! Mapovanie.
! Čísla reprezentujú stupne stupnice.
! Prvá položka je vyššie uvedená prostredná nota,
! pod ňou sú následujúce klávesy.
! Nenamapované klávesy označíme pomocou "x".
0
1
2
3
4
5
6
7
8
9
10
11
```

Prevzaté z [8].

B Metóda `renderNextBlock()`

Výpis B.1: Metóda `renderNextBlock()` triedy `MicroSamplerVoice`

```
void MicroSamplerVoice::renderNextBlock(AudioBuffer<float>&
    ↪ outputBuffer, int startSample, int numSamples)
{
  if (auto* playingSound = static_cast<MicroSamplerSound*> (
      ↪ getCurrentlyPlayingSound().get()))
  {
    auto& data = *playingSound->data;
    const float* const inL = data.getReadPointer(0);
    const float* const inR = data.getNumChannels() > 1 ? data.
        ↪ getReadPointer(1) : nullptr;

    float* outL = outputBuffer.getWritePointer(0, startSample);
    float* outR = outputBuffer.getNumChannels() > 1 ? outputBuffer.
        ↪ getWritePointer(1, startSample) : nullptr;

    while (--numSamples >= 0)
    {
      auto pos = (int)sourceSamplePosition;
      auto alpha = (float)(sourceSamplePosition - pos);
      auto invAlpha = 1.0f - alpha;

      float l = (inL[pos] * invAlpha + inL[pos + 1] * alpha);
      float r = (inR != nullptr) ? (inR[pos] * invAlpha + inR[pos +
          ↪ 1] * alpha)
        : l;

      auto envelopeValue = adsr.getNextSample();

      l *= lgain * envelopeValue;
      r *= rgain * envelopeValue;

      if (outR != nullptr)
      {
        *outL++ += l;
        *outR++ += r;
```

```
      }
      else
      {
        *outL++ += (l + r) * 0.5f;
      }

      sourceSamplePosition += pitchRatio;

      if (sourceSamplePosition > playingSound->endSample)
      {
        stopNote(0.0f, false);
      }
      else if (sourceSamplePosition >= playingSound->endSample - adsr
          ↪ .getParameters().release * playingSound->sourceSampleRate
          ↪ )
      {
        stopNote(0.0f, true);
      }
    }
  }
}
```

Časť kódu je prevzatá z triedy `SamplerVoice`, ktorá je súčasťou open source frameworku JUCE [14].

C Metóda `processBlock()`

Výpis C.1: Metóda `processBlock()`

```
void PolyBoxAudioProcessor::processBlock (juce::AudioBuffer<float>&
    ↪ buffer, juce::MidiBuffer& midiMessages)
{
    juce::ScopedNoDenormals noDenormals;
    auto totalNumInputChannels = getTotalNumInputChannels();
    auto totalNumOutputChannels = getTotalNumOutputChannels();

    for (auto i = totalNumInputChannels; i < totalNumOutputChannels;
        ↪ i++)
        buffer.clear (i, 0, buffer.getNumSamples());

    for (auto m : midiMessages)
    {
        auto message = m.getMessage();
        if (message.isNoteOn())
        {
            playedNote = message.getNoteNumber();
            DBG("Note:␣" + String(playedNote));
            break;
        }
        if (message.isNoteOff() && message.getNoteNumber() ==
            ↪ playedNote)
        {
            playedNote = -1;
            DBG("Note:␣" + String(playedNote));
        }
    }

    midiMessages.clear();

    if (transposeOn && playedNote != -1)
    {
        sequencer.transpose(60, playedNote);
    }
    else
```

```
{
    sequencer.transposeOff();
}

if (syncOn)
{
    if (auto ph = getPlayHead())
    {
        AudioPlayHead::CurrentPositionInfo info;
        if (ph->getCurrentPosition(info))
        {
            sequencer.setTempo(info.bpm);
            sequencer.setTimeSignature(info.timeSigNumerator, info
                ↪ .timeSigDenominator);
        }
    }
}

//Sequencer Control
auto interval = sequencer.getIntervalInSamples();
if (clockInterval != interval)
{
    DBG("INTERVAL␣" + String(interval));
    clockInterval = interval;
}

if (sequencer.isPlaying())
{
    stopped = false;
    for (int i = 0; i < buffer.getNumSamples(); i++)
    {

        if (sampleCounter++ == 0)
            sequencer.tick(i + 1);
        if (sampleCounter >= clockInterval)
            sampleCounter = 0;

    }
}
```

```
    midiMessages.swapWith(sequencer.midiMessages);
    sequencer.midiMessages.clear();

    // turn all notes off on stop
    if (!stopped && !sequencer.isPlaying())
    {
        for (int i = 1; i <= NUM_VOICES; i++)
            for (int j = 0; j < 128; j++)
                midiMessages.addEvent(MidiMessage::noteOff(i, j), 1);
        sampleCounter = 0;
        stopped = true;
    }

    sampler.renderNextBlock(buffer, midiMessages, 0, buffer.
        ↪ getNumSamples());
    buffer.applyGain(level);
}
```

D Zoznam tried a štruktúr

V tejto prílohe je uvedená stručná dokumentácia najdôležitejších tried a štruktúr aplikácie.

D.1 Dokumentácia triedy PolySequencer

Verejné metódy

- void **tick** (int sample)
- void **play** ()
- void **stop** ()
- void **reset** ()
- int **getSteps** ()
- int **getPosition** ()
- int **getTempo** ()
- int **getDuration** ()
- float **getTimeSignature** ()
- bool **isPlaying** ()
- int **getIntervalInSamples** ()
- void **setTempo** (int tempo)
- void **setDuration** (int duration)
- void **setTimeSignature** (int a, int b)
- void **setSampleRate** (int sampleRate)
- void **transpose** (int rootNote, int transposeNote)
- void **transposeOff** ()

Verejné atribúty

- SequencerVoice ∗ **voices** [6]
- MidiBuffer **midiMessages**

Dokumentácia pre túto triedu bola generovaná z nasledujúcich súborov:

- PolySequencer.h
- PolySequencer.cpp

D.2 Dokumentácia triedy SequencerVoice

Verejné metódy

- **SequencerVoice** (int index, int length)
- int **getPosition** ()
- void **setPosition** (int position)
- int **getLength** ()

- Note * **getNotePtr** (int index)
- Note * **getLastNotePtr** ()
- MidiBuffer **getNoteOn** (int sample, int transposition)
- MidiBuffer **getNoteOff** (int sample)
- void **assignChannel** (int channel)
- void **deassignChannel** (int channel)
- bool **hasChannel** (int channel)
- void **eraseNote** (int index)
- void **grow** ()
- void **shrink** ()
- void **advance** ()

Verejné atribúty

- std::function< void()> **onLengthChange**

Dokumentácia pre túto triedu bola generovaná z nasledujúcich súborov:

- SequencerVoice.h
- SequencerVoice.cpp

D.3 Dokumentácia štruktúry Note

Verejné metódy

- **Note** (int num, float vel, double prob)

Verejné atribúty

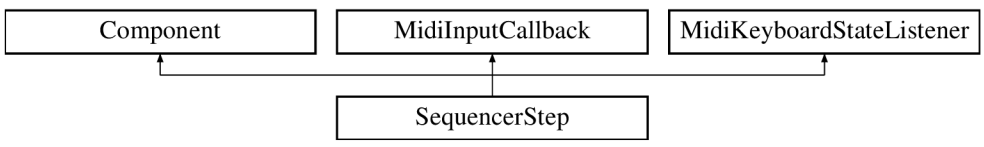
- int **number**
- float **velocity**
- double **probability**

Dokumentácia pre túto štruktúru (struct) bola generovaná z nasledujúceho súboru:

- SequencerVoice.h

D.4 Dokumentácia triedy SequencerStep

Diagram dedičnosti pre triedu SequencerStep



Triedy

- class Listener

Verejné typy

- enum **StepColour** {
  cInactive = 0xff235ABE , **cActive** = 0xff123b7b , **cBorder** = 0xc8ffffff ,
  cSelected = 0xffBE8723 ,
  cRecording = 0xe0f40600 }

Verejné metódy

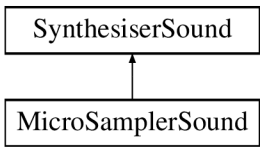
- **SequencerStep** (Note *note)
- void **paint** (juce::Graphics &) override
- void **resized** () override
- void **mouseDown** (const MouseEvent &event) override
- void **mouseDoubleClick** (const MouseEvent &event) override
- void **mouseDrag** (const MouseEvent &event) override
- void **mouseWheelMove** (const MouseEvent &event, const MouseWheelDetails &wheel) override
- void **handleNoteOn** (MidiKeyboardState *source, int midiChannel, int midiNoteNumber, float velocity) override
- void **handleNoteOff** (MidiKeyboardState *source, int midiChannel, int midiNoteNumber, float velocity) override
- void **handleIncomingMidiMessage** (juce::MidiInput *source, const juce::MidiMessage &message) override
- void **erase** ()
- void **setNoteNumber** (int number)
- void **setVelocity** (float velocity)
- void **setProbability** (double probability)
- int **getNoteNumber** ()
- float **getVelocity** ()
- double **getProbability** ()
- void **setActive** (bool active)
- void **setSelected** (bool selected)
- void **setRecording** (bool recording)
- bool **isActive** ()
- bool **isSelected** ()
- void **addListener** (Listener *listener)
- void **removeListener** (Listener *listener)

Dokumentácia pre túto triedu bola generovaná z nasledujúcich súborov:

- SequencerStep.h
- SequencerStep.cpp

D.5 Dokumentácia triedy MicroSamplerSound

Diagram dedičnosti pre triedu MicroSamplerSound



Verejné metódy

- **MicroSamplerSound** (const String &name, AudioFormatReader ∗source, String path, int midiChannel, const BigInteger &midiNotes, double frequencyForNormalPitch, double attack, double release)
- const String & **getName** () const noexcept
- AudioBuffer< float > ∗ **getAudioData** () const noexcept
- void **setEnvelopeParameters** (ADSR::Parameters parametersToUse)
- bool **appliesToNote** (int midiNoteNumber) override
- bool **appliesToChannel** (int midiChannel) override
- void **setAttack** (double a)
- void **setRelease** (double r)
- void **setStart** (double s)
- void **setEnd** (double e)
- void **setRoot** (double frequency)
- void **reverse** ()
- double **getAttack** ()
- double **getRelease** ()
- double **getStart** ()
- double **getEnd** ()
- double **getPlayingLengthInSeconds** ()
- double **getRoot** ()

Verejné atribúty

- double **pan** = 0.0f
- double **gain** = 1.0f
- bool **reversed** = false
- int **channel**
- String **sourcePath**

Priatelia (friends)

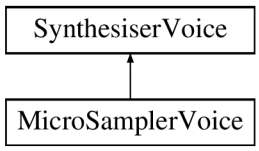
- class **MicroSamplerVoice**

Dokumentácia pre túto triedu bola generovaná z nasledujúcich súborov:

- MicroSampler.h
- MicroSampler.cpp

D.6 Dokumentácia triedy MicroSamplerVoice

Diagram dedičnosti pre triedu MicroSamplerVoice



Verejné metódy

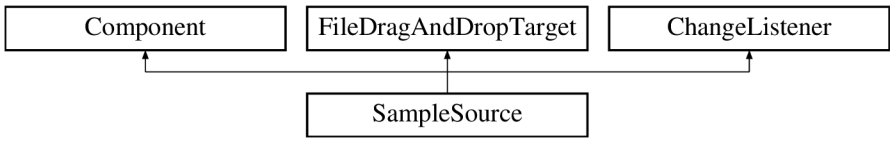
- **MicroSamplerVoice** (std::shared_ptr< Tunings::Tuning > tuning)
- bool **canPlaySound** (SynthesiserSound ∗) override
- void **startNote** (int midiNoteNumber, float velocity, SynthesiserSound ∗, int pitchWheel) override
- void **stopNote** (float velocity, bool allowTailOff) override
- void **pitchWheelMoved** (int newValue) override
- void **controllerMoved** (int controllerNumber, int newValue) override
- void **renderNextBlock** (AudioBuffer< float > &, int startSample, int numSamples) override

Dokumentácia pre túto triedu bola generovaná z nasledujúcich súborov:

- MicroSampler.h
- MicroSampler.cpp

D.7 Dokumentácia triedy SampleSource

Diagram dedičnosti pre triedu SampleSource



Triedy

- class Listener

Verejné metódy

- **SampleSource** (Synthesiser &s, int ch)
- void **paint** (Graphics &g) override
- void **resized** () override
- bool **isInterestedInFileDrag** (const StringArray &files) override
- void **fileDragEnter** (const StringArray &files, int x, int y) override
- void **fileDragExit** (const StringArray &files) override
- void **filesDropped** (const StringArray &files, int x, int y) override
- void **mouseDown** (const MouseEvent &event) override
- void **mouseDoubleClick** (const MouseEvent &event) override
- void **changeListenerCallback** (juce::ChangeBroadcaster ∗source) override
- void **addListener** (Listener ∗l)
- void **removeListener** (Listener ∗l)

Verejné atribúty

- const int **channel**
- MicroSamplerSound ∗ **sound** { nullptr }
- bool **selected** = false

Dokumentácia pre túto triedu bola generovaná z nasledujúceho súboru:

- SampleSource.h