VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky
a komunikačních technologií

BAKALÁŘSKÁ PRÁCE

Brno, 2021 Václav Rychtecký

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 NA PLATFORMĚ ANDROID S MOŽNOSTÍ OVLÁDÁNÍ NĚKTERÝCH PARAMETRŮ PŘEVODEM TEXTU

EXPERIMENTAL SOFTWARE MUSICAL INSTRUMENT ON ANDROID PLATFORM WITH ABILITY TO CONTROL PARAMETERS BY TEXT CONVERSION

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR
Václav Rychtecký

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: Václav Rychtecký ***ID:*** 195807
Ročník: 3 ***Akademický rok:*** 2020/21

NÁZEV TÉMATU:

Experimentální softwarový hudební nástroj na platformě Android s možností ovládání některých parametrů převodem textu

POKYNY PRO VYPRACOVÁNÍ:

Cílem práce je kompletně naprogramovat mobilní aplikaci - experimentální softwarový hudební nástroj pro platformu Android. Experimentálnost spočívá v kombinaci jednoduchého syntetizéru, sampleru a sekvenceru. Hudební parametry bude možné ovládat mj. i pomocí textu sejmutého vestavěným fotoaparátem.

DOPORUČENÁ LITERATURA:

[1] Electronic and experimental music: technology, music and culture. Editace Thom Holmes. 3. vydání, Routledge, New York, 2008, 462 s. ISBN 978-0-415-95782-3.

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

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

Cílem práce je vytvoření mobilní aplikace na platformě Android, vyznačující se tvorbou zvuku za pomocí textu zaznamenaného zabudovaným fotoaparátem zařízení v kombinaci se syntetizérem, sekvencerem a samplerem. Realizace vychází částečně z návrhu zpracovaného v rámci semestrální práce a je rozdělena do několika částí. V první části jsou stručně popsány druhy syntéz, operační systém Android, programovací jazyk Pure Data, MIDI a Android knihovna pro ovládání Pure Data. V druhé části je zpracován návrh aplikace a ve třetí části je popsána realizace a především funkcionalita aplikace, která je vytvořena pomocí programovacího jazyka Kotlin a IDE Android Studio od společnosti JetBrains v kombinaci s jazykem Pure Data.

KLÍČOVÁ SLOVA

experimentální hudební nástroj, Android Studio, Kotlin, Pure Data, sampler, sekvencer, syntetizér, text

ABSTRACT

The purpose of the thesis is to create a mobile application on the Android platform, characterized by creation of sound using a text captured by a built-in camera of the device in combination with synthesizer, sequencer and sampler. The implementation is partly based on a design prepared within a semestral thesis and is divided into several parts. The first part briefly describes the types of syntheses, the Android operating system, the Pure Data programming language, MIDI and the Android library for controlling Pure Data. The second part deals with the design of the application and the third part describes the implementation and especially the functionality of the application which is created using the programming language Kotlin and IDE Android Studio from JetBrains in combination with the language Pure Data.

KEYWORDS

experimental musical instrument, Android Studio, Kotlin, Pure Data, sampler, sequencer, synthesizer, text

RYCHTECKÝ, Václav. *Experimentální softwarový hudební nástroj na platformě Android*. Brno, Rok, 75 s. Bakalářská práce. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací. Vedoucí práce: doc. Ing. MgA. Mgr. Dan Dlouhý, Ph.D.

PROHLÁŠENÍ

Prohlašuji, že svou bakalářskou práci na téma „Experimentální softwarový hudební nástroj na platformě Android“ jsem vypracoval samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené bakalářské práce dále prohlašuji, že v souvislosti s vytvořením této bakalářské práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

Brno ....

podpis autora

PODĚKOVÁNÍ

Rád bych poděkoval vedoucímu diplomové práce, panu doc. Ing. MgA. Mgr. Danovi Dlouhému, Ph.D., za odborné vedení, konzultace, trpělivost a podnětné návrhy k práci. Dále panu Jakubovi Juroškovi za konzultace ohledně kódu psaného v jazyce Kotlin a slečně Mgr. Janě Poskerové za kontrolu a korekturu textu.

Obsah

Seznam obrázků

1 Úvod

Tato práce si klade za cíl navrhnout a vytvořit experimentální hudební nástroj na platformě Android využívající nevšední vstupy pro tvorbu zvuku. Jako hlavní stavební prvek celého nástroje byl zvolen text a jeho zhudebnění. To není žádnou převratnou novinkou a na internetu můžeme nalézt hned několik podobných aplikací, jako je např. *LangoRythm*. Tento nástroj však využívá pouze 26 písmen abecedy, tedy nerozlišuje velká a malá písmena, diakritiku přeskakuje, bpm (beats per minute) se mění podle průměrné délky slov a neumí pracovat s čísly a interpunkcí. Ve webovém prohlížeči používá pouze jediný zdroj zvuku. [7] Tímto je uživatel ochuzen o možnost experimentovat s různými zvuky a pro dokončení nebo obohacení skladby tedy musí vyexportovat MIDI sekvenci a dále ji upravovat v některém z nahrávacích softwarů. Nemožnost přidání nahraného zvuku či bicí složky přímo v aplikaci dělá z *LangoRythm* pouze jakýsi přechodný nástroj umožňující ozvláštnění kompozic z dosti omezeného úhlu. Instrument popsán v této práci je tedy oproti *LangoRythm* daleko komplexnější a uživatelsky přívětivější, byť slouží výhradně jako nástroj.

Při hledání podobných experimentálních nástrojů, využívajících text, na službě *Google Play* (tedy využívající OS Android), byly nalezeny pouze aplikace pro převod textu na řeč, které primárně slouží jako pomůcka pro osoby s poruchami zraku nebo trpící dyslexií.

Uživatel bude mít možnost zaznamenat pomocí vestavěného fotoaparátu zařízení (tj. zejména smartphonu) text ve dvou módech. V prvním režimu budou kromě písmen (anglické abecedy) brána v potaz i čísla, speciální znaky, diakritika a interpunkce. Ve druhém režimu bude sejmutý text filtrován a ve výsledku bude obsahovat 52 znaků anglické abecedy s rozlišením velkých i malých písmen. Výsledný zvuk bude tvořen pomocí syntetizátoru.

Pro rozšíření hudebních možností nástroje bude možné využít zabudovaný sekvencer pro dodání rytmické složky nebo nahrávat zvuk pomocí mikrofonu zařízení a využívat ho v sampleru aplikace.

Zvuková část nástroje bude naprogramovaná v jazyce Pure Data. Veškerý sběr, ukládání dat a jejich následné posílání na vstupy souborů Pure Data bude probíhat v programu aplikace naprogramované pomocí jazyka Kotlin a IDE Android Studio. Časová synchronizace bude probíhat pomocí nastavení bpm v aplikaci a přepočtu na ms.

Na začátku práce jsou stručně popsány metody syntéz, operační systém Android, jeho vývojové prostředí a programovací jazyky Kotlin a Pure Data, dále MIDI a Android knihovna pro ovládání jazyka Pure Data. Ve třetí kapitole je popsán návrh aplikace ve vývojovém prostředí Android Studio a Pure Data. V části realizace se práce věnuje popisu jednotlivých funkčních částí nástroje. Zde jsou popsány mož-

nosti hudebního využití aplikace a jak by bylo možné nástroj vylepšit.

2 Teoretická část práce

V této kapitole jsou popsány programovací jazyky, jejich vývojová prostředí a použité programové knihovny, ve kterých proběhne realizace projektu. Součástí je i rozbor jednotlivých druhu syntéz, které bude aplikace využívat.

2.1 Metody Syntézy

Tato podkapitola má za úkol osvětlit typy syntéz, které budou použity při realizaci projektu.

2.1.1 Aditivní syntéza

Patří k nejstarším způsobům generace zvuku a představuje lineární proces zpracování zvukového signálu, kdy obsahuje pouze ty frekvenční složky, které byly na počátku syntézy. Jsou založeny na součtu jednoduchých signálů v časové i frekvenční oblasti a vycházejí z Fourierova rozvoje periodického signálu, kdy si lze signál představit jako nekonečný součet harmonických signálů, jejichž frekvence je celistvými násobky kmitočtu rozkládaného tónu. Aditivní syntézy můžeme docílit součtem několika oscilátorů. První oscilátor beze změny amplitudy nebo frekvence můžeme považovat za fundament, na vstupy dalších oscilátorů přivádíme znásobenou frekvenci fundamentu, současně můžeme měnit i amplitudu a jejich výstupy sčítáme. Díky necelistvým násobkům fundamentu vzniká inharmonické spektrum a výsledná barva zvuku nabývá větší zajímavosti. Při součtu dvou oscilátorů, kdy se frekvence druhého oscilátoru blíží prvnímu, můžeme docílit vzniku rázů, které vnímáme jako periodické změny hlasitosti. [1]

2.1.2 Subtraktivní syntéza

Subtraktivní syntéza je technika postavená na myšlence, že i reálný nástroj může být rozdělen do tří částí. Zdroj zvuku, modifikátor upravující výstup zdroje a kontrolér umožňující ovládání nástroje hráčem, jako je změna parametrů. Příkladem může být klarinet, kdy plátek je zdroj zvuku a tělo modifikátor, klapky potom kontrolér. Tento koncept však není možné vztáhnout na všechny nástroje. I tak je to vhodná metafora pro pochopení, jak některé nástroje a tato syntéza fungují. V případě této syntézy je zdroj zvuku, bohatý na harmonické složky, reprezentován vstupním signálem typicky ve tvaru obdélníku, čtverce, pily nebo trojúhelníku. Modifikátor pomocí filtrů odstraňuje harmonické části vstupního signálu. [1]

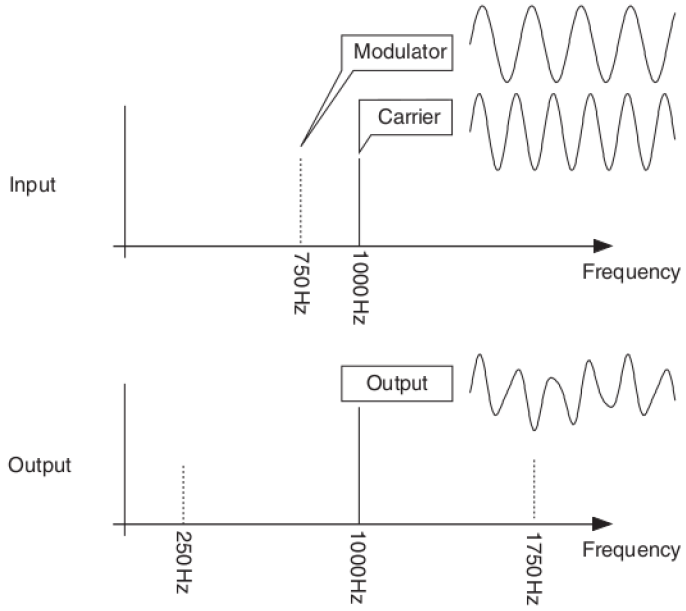
2.2 Modulační syntéza

Modulací ovlivňujeme některý parametr signálu (nosný) jiným signálem (modulačním). Tento způsob syntézy vede ke složitějšímu výslednému signálu, kdy vznikají úplně nové frekvenční složky, které nebyly obsaženy v nosném signálu.

Amplitudová modulace

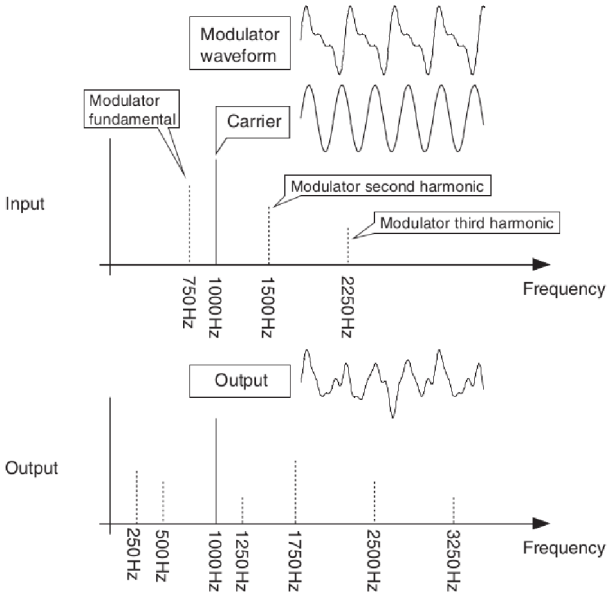
Amplitudovou modulací rozumíme řízení okamžité hodnoty amplitudy nosného signálu modulačním signálem.

Při sinusovém tvaru obou signálů, a je-li jejich frekvence ve slyšitelném spektru, vznikají dvě postranní pásma obsahující součet a rozdíl frekvencí nosného a modulačního signálu. [1]



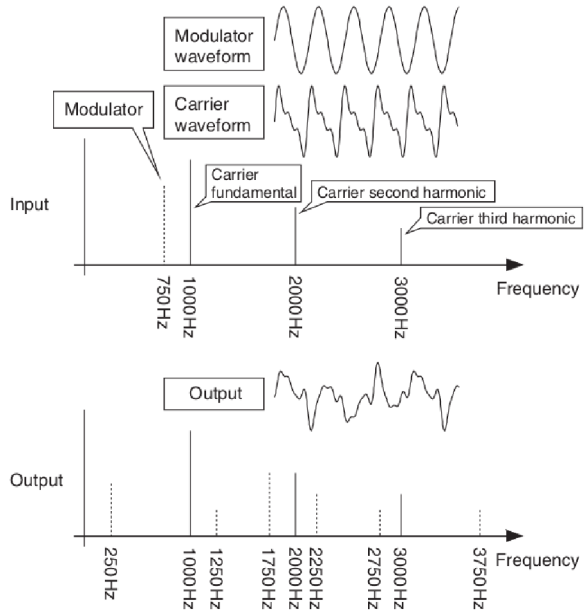
Obr. 2.1: Amplitudová modulace při sinusovém tvaru řídícího a nosného signálu (Zdroj: [1])

Nabývá-li modulační signál jiného tvaru než sinus a nosný zůstává sinusový, tak každá harmonická složka modulačního signálu vytváří nový pár součtu a rozdílu frekvencí na výstupu s absencí modulačních kmitočtů. [1]



Obr. 2.2: Amplitudová modulace při sinusovém tvaru nosného signálu (Zdroj: [1])

Při opačné situaci zde nacházíme zastoupení jak harmonických složek nosného signálu, tak i páry postranních pásem součtu i rozdílu nosné a modulační frekvence. Pokud je kmitočet modulačního signálu $f_m < 25$ Hz, získáváme efekt tremolo. [1]



Obr. 2.3: Amplitudová modulace při sinusovém tvaru řídícího signálu (Zdroj: [1])

Frekvenční modulace

Při tomto typu modulace je modulačním signálem řízený okamžitý kmitočet nosného signálu. Počet postranních pásem je dán hloubkou modulace, která je někdy označována jako modulační index. Ten je definován

$$h = \frac{f_\Delta}{f_m}, \tag{2.1}$$

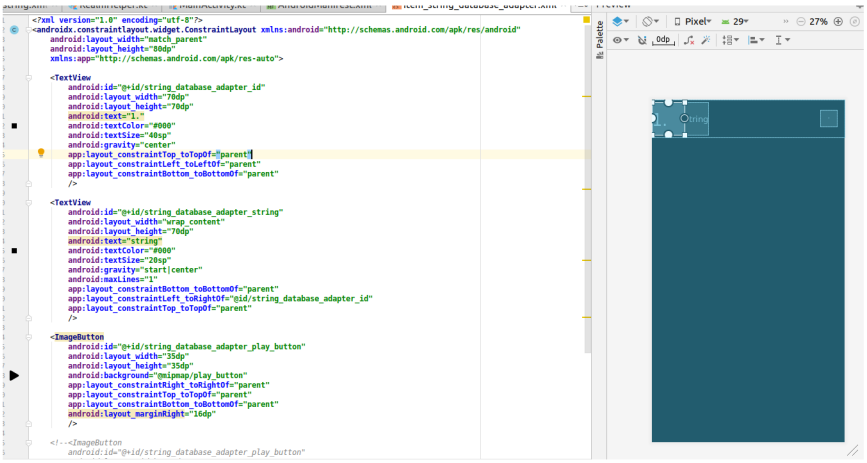
kde f_Δ znamená maximální rozdíl modulovaného a nosného kmitočtu a f_m je frekvence modulačního signálu. Pro jiný tvar než sinusový je výstup podobný jako u amplitudové modulace s větším zastoupením postranních pásem. Pokud je kmitočet modulačního signálu $f_m < 25$ Hz, získáváme efekt vibráto. [1]

2.3 Operační systém Android

Android je operační open source systém založený na Linuxovém jádře vyvíjený společností Google. Tento OS má největší počet uživatelů a zařízení na světě. Nachází se na smartphonech, tabletech, televizích, hodinkách, a dokonce i v autech. Díky tomu, že je open source, vzniká množství derivátů systému v rámci jednotlivých společností prodávajících chytré telefony, např. MIUI od Xaomi. V rámci programování je stále možné programovat aplikace na starší telefony díky podpoře starších API (Application Programming Interface). API můžeme vnímat jako soubor knihoven obsahujících metody a funkce, které jsou již napsány, a není potřeba si je programovat sám. Použitím API a jednoduchého průmyslového jazyka, jako je Java, se stává programování mobilních aplikací na Android dosažitelné téměř pro kohokoliv. [3]

2.3.1 Android Studio

Oficiální IDE (Integrated Development Enviroment) od společnosti JetBrains je nejrychlejší nástroj pro vývoj aplikací na platformě Android s výbornou kompatibilitou pro zařízení běžící na Linux, Windows nebo Mac. Součástí IDE je Visual Layout Editor, ve kterém lze pomocí značkovacího jazyka XML (Extensible Markup Language) vytvářet jednotlivé grafické návrhy aplikace nebo vlastní komponenty. Dále zde najdeme Android Virtual Device sloužící k testování aplikací na simulovaném Android zařízení a Realtime Profilers zobrazující vytížení CPU, paměti a síťové karty v reálném čase. [3]



Obr. 2.4: Ukázka XML editoru v Android Studiu

2.3.2 Kotlin

Jazyk Kotlin od společnosti JetBrains je od roku 2017 oficiálním vývojovým jazykem na programování aplikací pro operační systém Android. Kotlin byl navržen jako průmyslově spolehlivý a objektově orientovaný jazyk. Hlavní rozdíl mezi jazykem Java (druhý oficiální jazyk pro vývoj na OS Android) a Kotlin je syntax jazyka. Na rozdíl od jazyků odvozených od C, jakým je např. Java, se u jazyka Kotlin datový typ deklaruje až za jménem proměnné. Jména jednoduchých datových typů jsou psána s velkým počátečním písmenem a odpadá potřeba středníků na konci řádku. Kotlin běží nad JVM (Java Virtual Machine) stejně jako Java a je s ní interoperabilní. [6]

2.3.3 Realm

Realm Database je alternativní databázová knihovna k databázím *SQlite*, které se běžně používají v aplikacích Android. Díky designu *zero-copy*, kdy CPU neprovádí kopírování dat z jedné paměťové oblasti do druhé, je označován jako nejrychlejší databáze pro Android. [5]

2.4 MIDI

MIDI protokol hrál velkou roli při vývoji elektronické hudby od roku 1983 [1] a je určen např. pro zadávání a editaci hudebních dat mimo i v reálný čas, komunikaci mezi elektronickým nástrojem a počítačem nebo pro automatizovanou hru v podobě sekvencerů. Přenášená data nejsou zvukovým signálem, ale pouze řídícími daty zaznamenávající hudební události během hry na elektronický nástroj, tím pádem lze po nahrání měnit jednoduše zvuk nástroje pomocí jiných bank zvuků. [2]

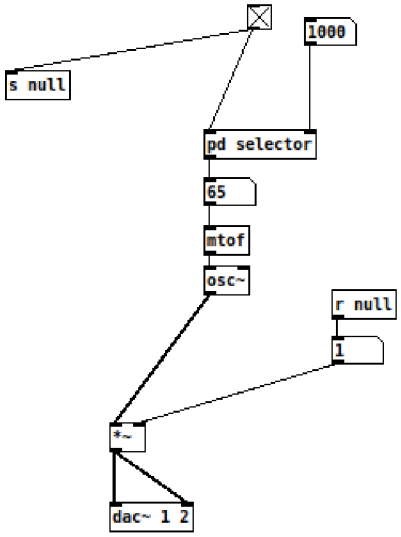
Tato práce bude využívat pouze hodnoty MIDI not, které budou převedeny na frekvence pomocí komponent v Pure Data, proto není potřeba tuto podkapitolu dále rozvádět.

Tab. 2.1: MIDI čísla not (Zdroj: [2])

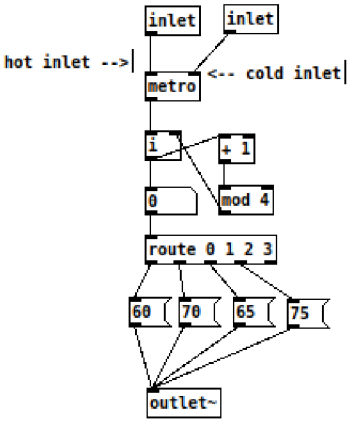
oktáva	hud. označení	C	C#	D	D#	E	F	F#	G	G#	A	A#	H
-2	sub-subkontra	0	1	2	3	4	5	6	7	8	9	10	11
-1	subkontra	12	13	14	15	16	17	18	19	20	21	22	23
0	kontra	24	25	26	27	28	29	30	31	32	33	34	35
1	velká	36	37	38	39	40	41	42	43	44	45	46	47
2	malá	48	49	50	51	52	53	54	55	56	57	58	59
3	jednočárkovaná	60	61	62	63	64	65	66	67	68	69	70	71
4	dvoučárkovaná	72	73	74	75	76	77	78	79	80	81	82	83
5	tříčárkovaná	84	85	86	87	88	89	90	91	92	93	94	95
6	čtyřčárkovaná	96	97	98	99	100	101	102	103	104	105	106	107
7	pětičárkovaná	108	109	110	111	112	113	114	115	116	117	118	119
8	šestičárkovaná	120	121	121	123	124	125	126	127				

2.5 Pure Data

Pure Data je open source vizuální jazyk vyvíjen v jazyku C, jehož autorem je Miller Smith Puckette. Může být použitý pro práci se zvukovými soubory, nahrávání a generování zvuku na různých zařízeních bez nutnosti psát řádky kódů. Za zmínku stojí i možnost práce s vizuálními a grafickými prvky. Jednotlivé objekty jsou reprezentovány boxy se jménem, vstupy (v rámci Pure Data o nich hovoříme jako o hot a cold inlets) a výstupy. Pomocí základních komponent lze vytvořit vlastní objekt. Tok dat mezi jednotlivými komponenty určují tzv. patch cords, čáry spojující výstupy a vstupy komponent. Dalším podobným softwarem z této rodiny jazyků je komerční *Max* od stejného autora. [4]



Obr. 2.5: Zapojení komponent



Obr. 2.6: Subpatch *pd selector*

Na obr. 2.5 můžeme vidět zapojení komponent v Pure Data, úplně nahoře je umístěné *toggle* tlačítko nabývající hodnot 1 v zapnutém a 0 ve vypnutém stavu, slouží jako spouštěč. Pomocí patch cordu je připojen k objektu *s null*, který posílá hodnotu tlačítka na komponentu *r null*, připojením na *number* můžeme vypisovat momentální hodnotu objektu *toggle*, kterou i násobíme výstup amplitudy z příkazu *osc~* reprezentující jednoduchý sinusový oscilátor, znaménko *tilda* v Pure Data znamená, že daný příkaz pracuje s audio signály. Objekt *mtof* slouží pro převod MIDI čísla na frekvenci tónu, který MIDI číslo symbolizuje, příkaz *dac~* (digital audio converter) slouží jako výstup. Uložením souboru v Pure Data vzniká patch, který do sebe může mít zanořeny další subpatche, jako v tomto příkladu objekt *pd selector* na obr. 2.6.

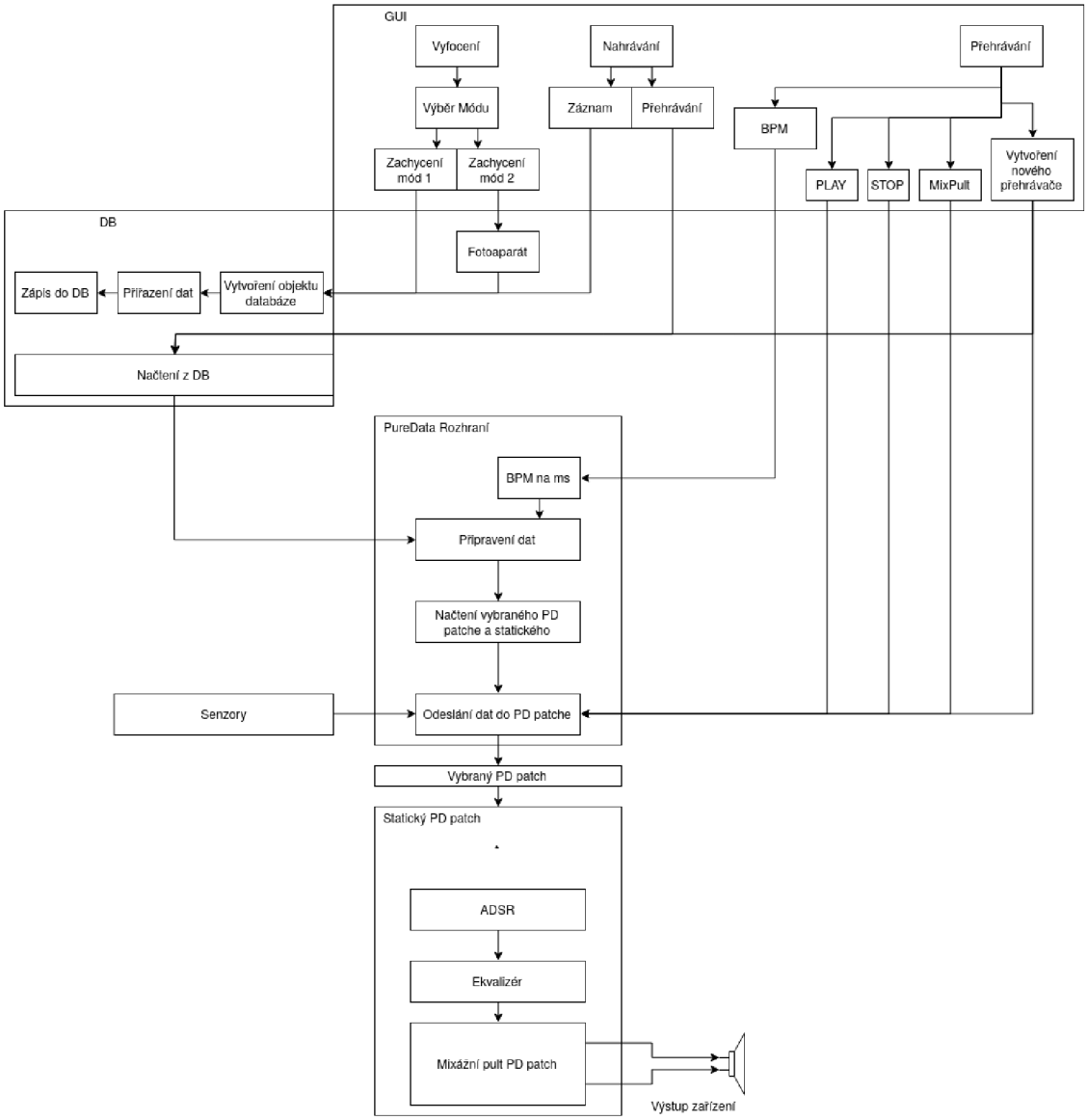
2.6 Android knihovna pro ovládání Pure Data

Soubory .pd lze na zařízení Android otevírat pomocí specializované knihovny *pd-for-android* od Libpd. Z Android zařízení lze ovládat .pd soubor posíláním dat jako je *float*, *string*, *bang* (speciální datový typ s textovou hodnotou *bang*, slouží jako spouštěcí signál). Android klient tedy nemůže měnit patch a veškeré pd. soubory musejí být připraveny předem a poté uloženy do aplikace jako raw soubory.

3 Návrh aplikace

3.1 Návrh aplikace v Android Studiu

Tato kapitola se zabývá návrhem aplikace pomocí jazyku Kotlin a XML v IDE Android Studio. Je zde popsána databáze a její objekty, GUI (general user interface), focení textu a jeho převod na data pro Pure Data patche, ovládání pomocí senzorů, nahrávání z mikrofonu zařízení, mixážní pult aplikace, přepočet bpm a komunikace mezi aplikací a Pure Data.



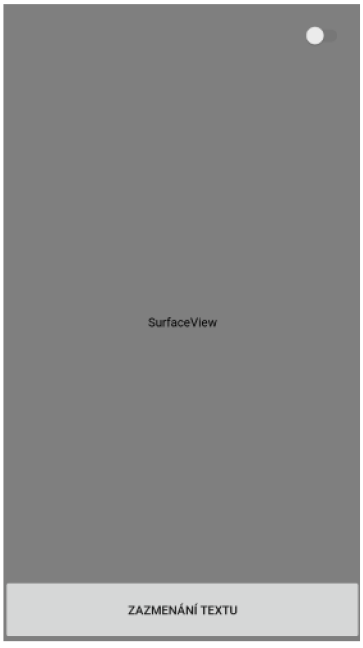
Obr. 3.1: Blokové schéma aplikace

3.1.1 GUI

Aplikace má povolenou orientaci pouze na výšku z důvodu možností ovládání potenciometrů pomocí senzorů. GUI bude vycházet z všeobecně přijímaného a Googlem doporučovaného stylu s komponentou *BottomNavigationBar* umístěnou na obrazovce dole, která pomocí fragmentů zprostředkovává tři různé části aplikace. Tlačítka, potenciometry a seznamy výběrů fungují klasicky na dotek. U potenciometrů však půjde pouze o aktivaci ovládání, nastavení hodnoty ovládá uživatel pomocí zabudovaných hardwarových senzorů zařízení, které jsou popsány v následující kapitole.

Fragment Vyfocení textu

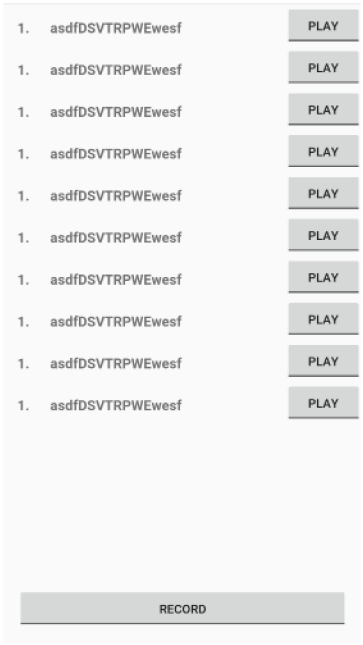
Na obr. 3.2 je grafický návrh fragmentu ve výchozím nastavení. Tlačítko úplně dole slouží pro zaznamenání textu, v pravém horním rohu je přepínač pro změnu módu zaznamenávání, zbytek obrazovky pokrývá záznam z kamery telefonu, který zprostředkovává komponenta *SurfaceView*. Při změně módu přepnutím v pravém horním rohu je uživatel přesměrován na aplikaci pro focení, kterou má každý smartphone již od svého výrobce.



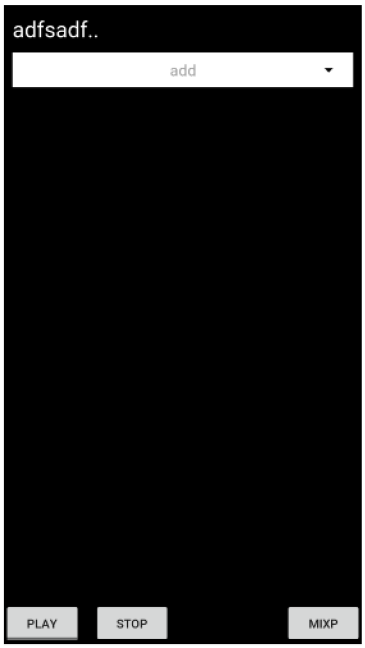
Obr. 3.2: Grafický fragmentu návrh Vyfocení textu

Fragment Nahrávání zvuku

Fragment slouží k zaznamenávání zvuku z mikrofonu telefonu při stisknutí tlačítka umístěného úplně dole. Zbytek vyplní přehled nahraných zvuků a možnost jejich přehrání. Na obr. 3.3 je přiblíženo, jak může fragment vypadat po naplnění daty.



Obr. 3.3: Grafický návrh fragmentu Nahrávání zvuku



Obr. 3.4: Grafický návrh fragmentu Přehrávání

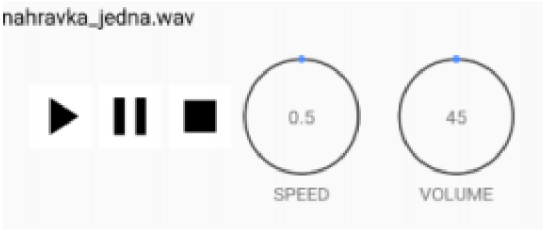
Fragment Přehrávání

V této části může uživatel vytvářet skladbu pomocí zaznamenaných textů. Vzhledem k možnosti kombinace různých textů je tato část grafického uživatelského rozhraní více komplexní než předešlé. Při otevření jsou zde pevná tlačítka umístěná v dolní části. Slouží pro navolení bpm, spuštění, zastavení, stopnutí přehrávání a otevření mixážního pultu aplikace a uložení presetu. Pro výběr jednotlivého modulu pro přehrávání textového řetězce, sampleru / sekvenceru nebo zvukové nahrávky bude určena otevírací nabídka.

Při zvolení se vytvoří nová komponenta složená ze základních komponent určených k tvorbě aplikací. Pro lepší představu je na obr. 3.5 přibližný náhled modulu pro nahrávku a na obrázku 3.4 je fragment určený pro přehrávání.

3.1.2 Senzory

Android zařízení mají několik zabudovaných hardwarových senzorů. Ke každému z nich se dá přistupovat softwarově a číst jeho aktuální hodnotu.



Obr. 3.5: Grafický návrh modulu pro přehrávání nahrávek

Proximity senzor

Proximity senzor zaznamenává většinou vzdálenost hlavy uživatele od zařízení a vrací hodnotu v centimetrech. Na některých telefonech nebo tabletech však vrací pouze 1 pro vzdálenou hodnotu nebo 0 pro hodnoty blízké [3], v tomto případě se stav určuje přičítáním nebo odečítáním po tak dlouhou dobu, jak bude senzor zapnutý a jakou bude mít hodnotu, tedy bude-li aktivní. Proximity senzorem jsou nastavovány hodnoty ADSR obálky.

Světelný senzor

Světelný senzor bývá u zařízení umístěn v přední kameře a zaznamenává intenzitu osvětlení v *lx*. [3] Při spuštění nahraného zvuku se jím ovládá rychlost přehrávání směrem vpřed i dozadu, například pomocí stínění ruky.

Gyroskop

Gyroskopem se ovládají potenciometry u přehrávání jednotlivých slov otáčením podél osy Z, telefon tedy bude muset ležet nejlépe na rovném povrchu. Výjimku tvoří pohyb podél osy Y, kterým se bude ovládat hlasitost vybraných potenciometrů v mixážním pultu.

3.1.3 Focení textu a jeho převod na data

Kvůli větší náročnosti na ostrost fotky při prvním módu pro přehrávání textu, kdy je brána v potaz interpunkce a diakritika, čte tento mód text ze zachycených snímků z fotoaparátu telefonu. Uživatel je tedy přesměrován na aplikaci pro focení, kterou má smartphone již od výrobce. Po pořízení a odsouhlasení fotky z ní bude pomocí Google Vision API vytažen text a fotografie smazána z telefonu. Pro možnost využívání fotoaparátu v aplikaci musí být uděleno povolení od uživatele. Bez udělení nelze výslednou aplikaci v tomto případě spustit. Záznam pro zhudebnění textu pomocí

druhého módu probíhá promítáním obrazu kamery do komponenty *SurfaceView*. Obraz se nedá zachytit do fotky, a to pro faktor náhodnosti. Ten vyplývá z rozostření kamery či třesu ruky uživatele. Následující řádky obsahují kód pro získávání textu ze zdroje kamery.

Výpis 3.1: Rozpoznávání textu ze zdroje

```
val textRecognizer = TextRecognizer
.Builder(activityContext)
.build()

textRecognizer.setProcessor(object :
Detector.Processor<TextBlock>{

  override fun release () {}
  override fun receiveDetections(detections:
  Detector.Detections<TextBlock>) {

    val items = detections.detectedItems
    if (items.size() != 0){
      val stringBuilder = StringBuilder ()
      for ( i in 0 until items.size()){
        val item = items.valueAt(i)
        stringBuilder.append(item.value)
      }
      capturedString = stringBuilder.toString()
      RealmHelper().filter(capturedString!!)
    }
  }
})
```

Takto získaný textový řetězec je dále zpracován metodou *filter()*. Zde se filtruje text, tomu odpovídá i jméno. Na výpisu 3.2 je znázorněna filtrace zachyceného textového řetězce při prvním módu.

Převod na data při prvním módu přehrávání

U prvního módu není vrchní hranice délky určena pevným číslem, ale bude záležet na velikosti vyfoceného textu. Přiřazování hodnot MIDI not je podle tabulky uložené v databázi, jedinou výjimku tvoří znak tečky, ke které se přiřazuje průměrná hodnota MIDI noty z předchozích znaků mezi touto a poslední tečkou. V rámci možnosti čtení diakritiky má každé písmeno přiřazen multiplikátor nabývající hodnotu prodloužení (čárky nad písmeny, kroužek nad ů) nebo zkrácení (háčky) doby přehrávání znaku.

Prodloužení je přítomno i u interpunkce a nabývá různé hodnoty v závislosti na znaku.

Výpis 3.2: Filtrace textových znaků při ukládání v prvním módu

```
val array = ArrayList<String>()
lateinit var list : List<String>

for( i in string.indicies){
  if (string[i].isLetter())
  array.add(string[i].toString())
}

if (array.isEmpty()){
  return
}

if(array.size < 20){
  array.reverse()
  for (i in 0..(array.size - 20)){
    list = array.drop(i)
  }
  list.reversed()
}
```

Převod na data při druhém módu přehrávání

U druhého módu je vrchní hranice délky textového řetězce určena na 20 znaků a při stisknutí tlačítka *záznam* je textový řetězec poslán do metody, kde probíhá filtrace. Výsledný *string* potom obsahuje pouze velká a malá písmena. Následují metody vytvářející *ArrayList<Float>* pro typy přehrávání. Důvod, proč ukládané hodnoty jsou v datovém typu *float*, byť jsou MIDI čísla not podle tabulky 2.1 pouze celá čísla, je vysvětlen v kapitole o komunikaci mezi aplikací a Pure Data.

Prvním typem je arpeggiator, kde se ukládá hodnota MIDI čísla pro všech dvacet znaků. Další typy pro přehrávání jsou čtvrťové noty, osminové a šestnáctinové.

Náhodným výběrem písmena z řetězce je dána stupnice, z níž se skládají jednotlivé tóny a postupy u zbylých typů přehrávání, čímž se zabrání, aby dva totožné textové řetězce zněly stejně. Hodnoty MIDI not ve stupnicích jsou uloženy v databázi.

Při přiřazování hodnot u typu se čtvrťovými notami je postup: základní tón, subdominanta, dominanta a opět kořenový tón.

Při osminách je zkopírován postup pro první čtyři tóny stejně, u dalších je výběr: subdominanta, dominanta, citlivý a základní tón.

U šestnáctinových not je zkopírovaný postup jako u osminových s drobnou úpravou, kdy první a poslední čtyři tóny jsou určeny stejně jako v předchozích případech a zbytek vyplňují náhodné tóny obsažené v dané tónině.

Vzhledem k velikosti textového řetězce a možnosti nezastoupení všech tónů stupnice se při absenci přiřazovaného tónu náhodně vybírá znak, který má o jedno vyšší MIDI číslo, nebo o jedno nižší. Pokud textový řetězec neobsahuje tento znak, vybere se automaticky druhý. Není-li v textovém řetězci zastoupený ani jeden znak, rozhoduje náhodný výběr písmene z textového řetězce.

3.1.4 Nahrávání zvuku

Nahrávání zvuku pomocí mikrofonu zařízení a následná práce s tímto zvukem je jedna z věcí, které podobné nástroje, využívající zhudebnění textu, neobsahují. Nahrávání zvuku, zápis a čtení z adresáře zařízení musí být povoleno uživatelem. Pokud tak nebude učiněno, aplikace sice funguje dál, ale bez této možnosti.

Po spuštění nahrávání se pomocí knihovny pro záznam zvuku zapisuje výstup z mikrofonu telefonu. Při stisknutí tlačítka *stop* může uživatel zadat název souboru. Soubor se poté ukládá do adresáře telefonu.

3.1.5 Záznam a načtení presetu

Pokud je uživatel s výslednou skladbou spokojen a chtěl by ji využít i v budoucnu, má aplikace možnost vytvoření presetu, kdy se při stisknutí tlačítka uloží veškeré číselné a textové hodnoty modulů užitých při přehrávání do databáze.

Nejjednodušší možností, jak řešit zápis tolika dat, je vcelku jednoduchý záznam do databáze, kdy se každý modul uloží do objektu (viz podkapitola 3.1.6, *Realm objekty*). Celou koncepci lze ještě zlehčit vytvořením listu obsahujícího listy, to znamená, že pro uložení presetu budou existovat dva objekty databáze, kdy jeden bude obsahovat informace o modulu a druhý bude list objektů s informacemi. Tato možnost ulehčí implementaci a přehlednost výsledného kódu odstraněním nutnosti větvení programu, které může nastat při ukládání pouze do jednoho objektu databáze.

3.1.6 Databáze

Databáze je zprostředkována pomocí knihovny *Realm*, veškeré operace spojené s databází jsou uloženy ve třídě s názvem *RealmHelper()*. Tabulka s uloženými hodnotami jednotlivých znaků a MIDI čísel společně s tabulkou obsahující data pro stupnice je vytvořena při prvním spuštění aplikace a nedá se jakkoliv editovat nebo

vytvářet znovu. Následující řádky kódu obsahují uložení hodnot MIDI not s přiřazeným znakem pro základní abecedu obsahující 26 znaků. V jediném *for()* cyklu, který tento výpis obsahuje, lze však vidět větší rozmezí – to je dáno tím, že jsou zde započítána i velká písmena.

Výpis 3.3: Ukládání tabulky pro MIDI noty do databáze

```
private fun findAndShowAllMIDI(): Boolean {
  return Realm.getDefaultInstance()
  .where(MIDITable::class.java)
  .findAll()
  .isEmpty()
}

fun saveAlphabet(){
  if (!findAndShowAllMIDI()){
    return
  }
  for (i in 60..111){
    val realmObject = MIDITable(alphabet[i - 60], i.toFloat())
    Realm.getDefaultInstance().use { it ->
      it.executeTransaction {
        it.copyToRealm(realmObject)
      }
    }
  }
}
```

Realm objekty

V rámci centralizace dat byly vytvořeny vlastní objekty, které rozšiřují třídu *RealmObject()*. Tento objekt databáze data pouze čte a nedělá kopie objektů ukládajících se do paměti jako při použití například *SQLite*.

Objekt pro uložení dat z textu

Základem tohoto objektu je primární klíč s identifikátorem uložený v proměnné s datovým typem *int*, který je povinný u každého objektu. Dále zde nacházíme v datovém typu *string* uložený celý textový řetězec slova a variace pro přehrávání s uloženými čísly MIDI not v datovém typu *RealmList<Float>*. *RealmList* rozšiřuje třídu *List*.

Objekt MIDI tabulky

Tento objekt se skládá z primárního klíče s identifikátorem v datovém typu *string* a reprezentuje znak. Číslo MIDI noty je v datovém typu *float*. Součástí je i multiplikátor pro speciální znaky s diakritikou či interpunkcí s výchozí hodnotou *0f* v datovém typu *float*.

Objekt stupnice

Primárním klíčem je zde název stupnice s indexem určujícím oktávu v datovém typu *string*. Index bude nabývat hodnoty 0 až 10, podle počtu oktáv, kterých můžeme dosáhnout pomocí MIDI (viz tabulka 2.1). Čísla not jsou zde zapsány jako pole o sedmi členech.

Objekt nahrávky

V objektu nahrávky je uložený název souboru jako primární klíč a cesta v datovém typu *string*.

Objekt *PresetObject*

Pro objekt presetu je zvolen jako primární klíč název modulu v datovém typu *string*, hodnoty potenciometrů v *RealmList<Float>* a textové řetězce v *RealmList<String>*.

Objekt presetu

Objekt je složen z názvu presetu jako primárního klíče v datovém typu *string*, presety modulů jsou potom uloženy v *RealmList<PresetObject>*.

3.1.7 Mixážní pult aplikace

Při stisknutí tlačítka pro mixážní pult bude uživatel přesměrován na nové okno aplikace vycházející ze fragmentu pro přehrávání. Konečný počet tahových potenciometrů bude odvozen od počtu navolených modulů pro přehrávání.

3.1.8 Přepočet bpm

Pro získání doby trvání čtvrťové noty v sekundách můžeme použít vzorec

$$t = \frac{60}{bpm}, \tag{3.1}$$

kdy číslo 60 symbolizuje počet sekund v jedné minutě. Jelikož aplikace bude fungovat pouze ve čtyř čtvrťovém taktu, jednoduchým násobením a dělením hodnoty čtvrťové noty můžeme získat dobu trvání ostatních not.

3.1.9 Třídy pro Pure Data

Vzhledem ke konečnému počtu modulů pro přehrávání bude každý jednotlivý patch mít svou vlastní třídu. Pro omezení duplikace kódu bude využita dědičnost, kde každá nová třída bude dědit ze svého předka. Týká se to např. metod pro otevření a zavření patche nebo posílání dat.

3.1.10 Komunikace mezi aplikací a Pure Data

Data z aplikace jsou posílána na objekty typu *reciever*, obsažené v kódu patche. Tyto přijímače zpráv mají svá jedinečná jména. Ta by mohla být obsažená v databázi, ovšem pro tyto účely postačí, aby byla uložena ve statické proměnné. Knihovna *pd-for-android* obsahuje metody posílající číselnou hodnotu pouze v datovém typu *float*, z toho důvodu je u všech proměnných v databázi nastaven stejný datový typ.

3.2 Návrh aplikace v Pure Data

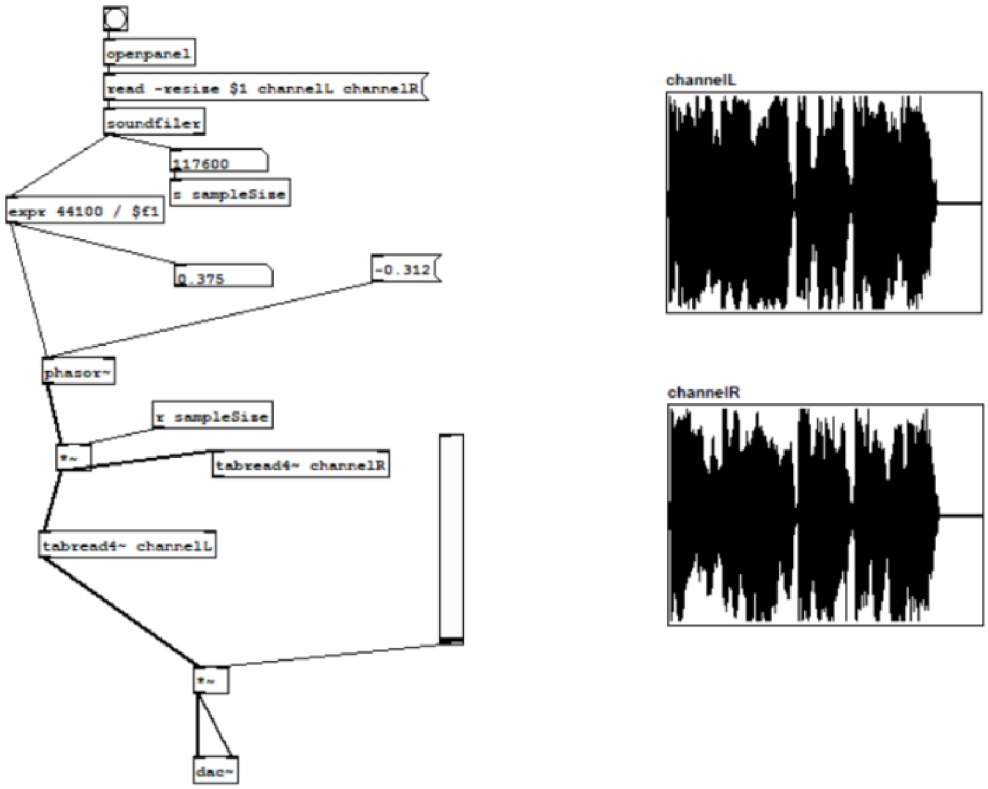
V této kapitole jsou popsány návrhy jednotlivých patchů a jejich částí určené pro audio část aplikace. Veškeré časování bude kvůli minimální latenci probíhat v Pure Data a konečný počet modulů je dán především nepřímým přístupem k patchi z aplikace, kdy přes ni nelze patch jakkoliv editovat.

3.2.1 Přehrávání a úprava zvukové nahrávky

Pro úpravu rychlosti přehrávání nahrávky nebo její délky je určen patch (obr. 3.6), do kterého se načte soubor z adresáře telefonu, zde se získá pomocí příkazu *soundfiler* jeho délka, tedy počet, z kolika vzorků je nahrávka tvořena. Dělením hodnoty vzorkovacího kmitočtu touto hodnotou bude získáno číslo a jeho napojením na *phasor~* bude docíleno plynulého přehrávání nahrávky. Pro změnu rychlosti už tedy stačí pouze přičítat hodnoty na vstupu komponenty *phasor~*. V záporných číslech bude docíleno přehrávání pozpátku.

3.2.2 ADSR obálka

ADSR obálka funguje na principu zvyšování a snižování amplitudy po určitý čas v rámci modulu ADSR. Uživatel navolí hodnoty jednotlivých částí v rámci urče-

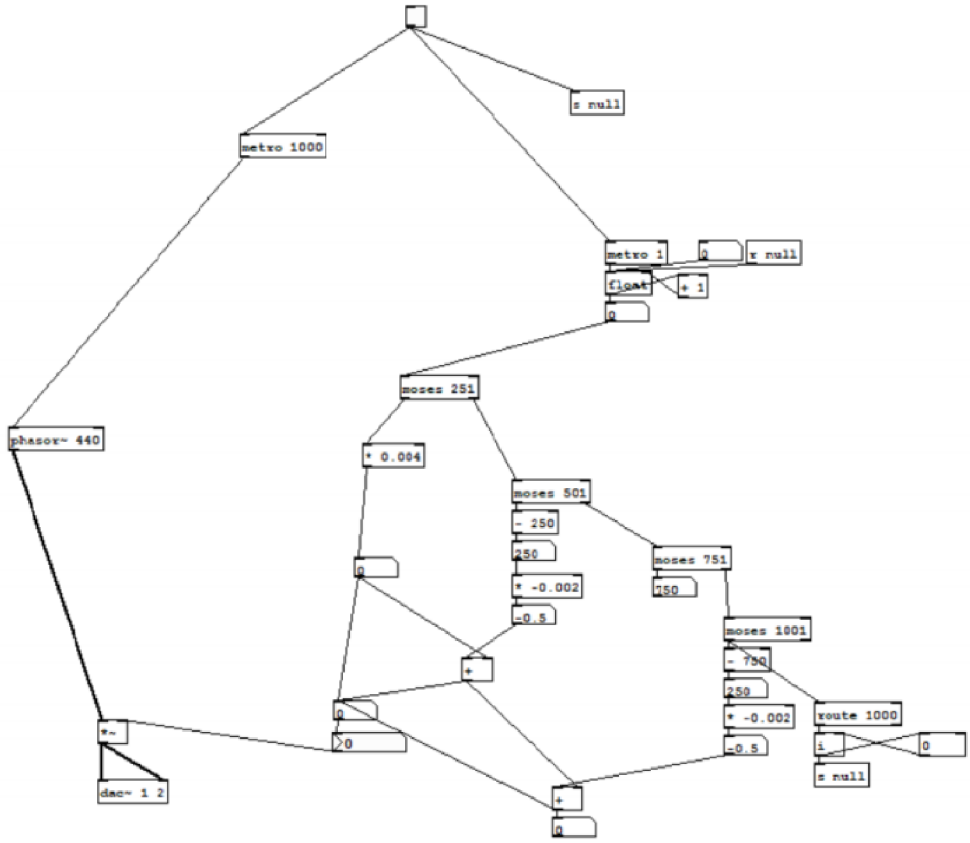


Obr. 3.6: Patch pro přehrávání nahrávek

ného bpm a zvolené noty, která bude vycházet ze zvoleného způsobu přehrávání. V pd patchi se vše řídí opět pomocí příkazu *metro*. Jednotlivé hodnoty ADSR jsou přiřazeny ke komponentám typu *moses*. Ta z jednoho nebo druhého outletu vysílá řídící signál podle nastavené podmínky. Podmínka určuje hranici, z jakého outletu bude posílán signál dál. Pro lepší přestavu se dá jeho funkce ztotožnit s obyčejnou podmínkou *if()*. Předpokládejme, že hodnota *moses* je nastavená na 10. Pravý outlet tedy obsahuje podmínku *je menší než nastavená hodnota*, zatím co pravý outlet obsahuje *větší nebo rovno 10*.

3.2.3 Řízení not

Přehrávání dané hodnoty noty probíhá pomocí patche, do kterého přicházejí data z Android aplikace, jako je bpm přepočítané na ms a MIDI číslo noty. Převod MIDI čísla noty na frekvenci tónu obstará komponenta *mtof*. Aby se noty střídaly podle určeného bpm, je zapotřebí komponent *metro* a *route*. Poslední zmíněná funguje jako příkaz *switch()* známý z rodiny jazyků C. *Route* při shodě čísla vyšle řídící příkaz, který doputuje na vstup číselné hodnoty, a ta ji přepošle až na vývod patche. Pro



Obr. 3.7: Patch ADSR

lepší ilustraci je příklad takového zapojení na obr. 2.6. Tímto bude zajištěno pravidelné střídání tónů.

3.2.4 Sampler

Patch pro sampler musí obsahovat časovače opět s využitím příkazu *metro*, které na základě přijatých dat ze zařízení vypočítávají hodnotu v milisekundách pro spuštění přehrávání jednotlivého vzorku.

3.2.5 Statický patch

Statický pd patch bude pro všechny patche v aplikaci stejný, a proto není potřeba ho duplikovat do jednotlivých patchů určených pro přehrávání. Recievery na vstupech přijímají tok audio dat.

Všechny patche posílají své výstupy na mixážní pult aplikace, kde se dá nastavit hlasitost jednotlivých modulů a vytvořit tak konečný mix kompozice. Maximální hodnota zde nabývá jedné, minimální nuly.

4 Realizace

Při realizaci práce se přiložený návrh aplikace zpracovaný v rámci semestrální práce projevil jako hrubě nedostačující, sloužil tedy pouze jako odrazový můstek a bylo potřeba přikročit k razantním změnám. Z velké části chyběl návrh v jazyce Pure Data, zejména pro syntetizátor, a původní návrhy byly často nepoužitelné nebo vyžadovaly větší modifikaci.

V části aplikace zpracovávající data došlo k výrazné změně v celé architektuře programu psaného v jazyce Kotlin, která se v realizaci snaží co nejvíce přiblížit modelu Clean Architecture (v češtině jako čistá architektura). Tento konstrukt můžeme zjednodušeně vnímat jako rozložení aplikace do vrstev, které mezi sebou komunikují pouze předáváním dat. Změněna byla i databáze aplikace. V mnoha případech se jednalo o úplné nahrazení původního konceptu objektů nebo jejich rozšíření o nová data.

Ve výsledku princip převodu textu na hudbu zůstal, byl však také upraven. Například při zachycení textu v prvním módu bylo nakonec přikročeno ke stanovení horní hranice počtu písmen v zaznamenávaném textu a vynechání symbolu tečky nebo odebrání možnosti fotit text přes fotoaparát zařízení. V praxi bylo totiž zjištěno, že ostrost fotky sice má velký vliv na zaznamenávání interpunkce a diakritiky, avšak zachycení textu v reálném čase, jako u druhého módu, je rychlejší a poslouží stejně dobře jako statická fotka. Pro lepší hudební možnosti nástroje bylo přikročeno k částečnému spojení obou módů zachycení. Jediným rozdílem mezi módy tedy bude pouze schopnost rozlišovat diakritická a interpunčkní znaménka. Opuštěn byl i původní záměr pro ovládání potenciometrů a táhel aplikace pomocí zabudovaných senzorů zařízení, který se v praxi příliš neosvědčil. Nastavování hodnot se kvůli tomu stalo zdlouhavým a neefektivním procesem. Poslední odebranou věcí je možnost ukládání presetů, tato funkcionalita zůstala pouze u sekvenceru.

Jednou z mála použitelných věcí v návrhu bylo uživatelské rozhraní. I to však doznalo změn, a to především po estetické a funkční stránce. Styl s komponentou *BottomNavigationBar* zůstal. Kromě tří fragmentů jsou nyní použity fragmenty čtyři, a to z důvodu zjednodušení uživatelského rozhraní. K původnímu návrhu přibyl ještě fragment, ve kterém se zobrazují v databázi uložené textové řetězce. Největších změn doznal fragment pro přehrávání. Uživatel měl původně na výběr z většího množství modulů, tento návrh kvůli malému rozlišení displeje telefonu v praxi vypadal velmi nepřehledně, a proto byl předělán.

Za zmínku stojí i nové možnosti nástroje, jako je například přidání delaye, reverbu, stříhání nahrávek, pulzně šířková modulace pro vlnu obdelníkového průběhu nebo možnost amplitudově modulovat signál pomocí vlny, která prošla přes filtr typu dolní propusti s nastavitelnou rezonancí.

Dále byl návrh modifikován přidáním několika externích otevřených knihoven pro podporu práce se zvukem v prostředí Android Studia, jako je například knihovna *FFmpeg*, díky které je možné sříhat nahrávky přímo v aplikaci, nebo knihovna *WaveRecorder* umožňující záznam z mikrofonu zařízení ve formátu *wav*.

4.1 Android aplikace

V rámci použití čisté architektury byla část nástroje realizovaného ve vývojovém prostředí Android Studio rozdělena do tří vrstev.

První vrstva má za úkol zobrazovat a aktualizovat grafické uživatelské rozhraní nebo přijímat data od uživatele, jako je například stisknutí tlačítka. Je realizována pomocí fragmentů a jejich tříd.

```
1   package com.archonalabs.bakalarskaprace.fragments
2
3   import ...
20
21  class CapturingStringFragment : BaseFragment() {
22
23      private lateinit var cameraSource: CameraSource
24      private val viewModel : CapturingStringViewModel by viewModel()
25
26      override fun onCreateView(inflater: LayoutInflater,
27                                container: ViewGroup?,
28                                savedInstanceState: Bundle?
29      ): View? {
30          return inflater.inflate(R.layout.fragment_capturing_string
31              , container, attachToRoot: false)
32      }
33
34      @SuppressLint( ...value: "SourceLockedOrientationActivity")
35      override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
36          super.onViewCreated(view, savedInstanceState)
37          activity?.requestedOrientation = ActivityInfo.SCREEN_ORIENTATION_PORTRAIT
38          initObservers()
39          initCamera()
40          capturing_string_switch.setOnCheckedChangeListener { _, _ -> }
41          capturing_string_button.setOnClickListener {  it: View!
42              viewModel.captureString(capturing_string_switch.isChecked)
43          }
44      }
```

Obr. 4.1: Fragment pro zachycení textu

Ke každé třídě fragmentu je přiřazen *view model*, který zpracovává data od nebo pro uživatele. V případě nástroje se zde například nacházají prostředky pro ko-

munikaci s patchi napsanými v jazyce Pure Data za pomoci *pd-for-android*. Tato knihovna obsahuje metody, které na objekty typu *recieve* a *send* odesílají data ve tvaru koncového bodu (*string*) a hodnoty (*list*, *float*, *string*).

```
1  package com.archonalabs.bakalarskaprace.viewmodels
2
3  import ...
14
15 class CapturingStringViewModel(private val saveStringUseCase : SaveStringUseCase
16 ) : ViewModel() {
17
18     private var _string : MutableLiveData<String> = MutableLiveData()
19     val string : LiveData<String> = _string
20
21     private  var _lockDuringSave : MutableLiveData<Boolean> = MutableLiveData()
22     val lockDuringSave : LiveData<Boolean> = _lockDuringSave
23
24     fun captureString(checked: Boolean) {
25         viewModelScope.launch {  this: CoroutineScope
26             _lockDuringSave.postValue( value: true)
27             viewModelScope.launch {  this: CoroutineScope
28                 if (_string.value != null && _string.value!!.isNotEmpty()){
29                     Timber.d( message: "Check string: %s", _string.value)
30                     saveStringUseCase(params = SaveStringUseCase.
31                     Params(string = _string.value.toString(),
32                         isFirstMode = checked))
33                 }
34             }
35             _lockDuringSave.postValue( value: false)
36         }
37     }
```

Obr. 4.2: ViewModel pro fragment zachycení textu

Část zprostředkovávající komunikaci mezi první a třetí vrstvou aplikace je řešena pomocí konstrukcí *use case*.

Třetí vrstva aplikace obsahuje databázi, její objekty a s ní spojené operace zápisu dat.

4.1.1 Databáze

Databáze je nedílnou součástí nástroje. Jsou zde uchována data nejen o všech znacích, které aplikace umí a využívá, ale také o stupnicích, zaznamenaných textech, přednastevní pro sekvencer nebo souborech nahraných pomocí mikrofonu zařízení.

```kotlin
package com.archonalabs.domain.feature.strings

import com.archonalabs.domain.uscases.UseCase

class SaveStringUseCase(private val stringRepository: StringRepository
) : UseCase<Unit, SaveStringUseCase.Params>() {

    override suspend fun doWork(params: Params) = stringRepository
        .saveAndFilterString(params.string, params.isFirstMode)

    data class Params(val string : String, val isFirstMode : Boolean)
}
```

Obr. 4.3: Use case pro zachycení textu

Znaky

Aby bylo možné převést text na data, jak je zmíněno v návrhu aplikace, je potřeba mít k jednotlivým znakům, ze kterého se textový řetězec skládá, přiřazeny hodnoty MIDI čísel, popřípadě multiplikátoru.

```kotlin
@RealmClass
open class Character(
    @PrimaryKey
    var character : String = "",
    var value : Float = 0.0f,
    var multiplier : Float = 0.0f,
) : RealmObject()
```

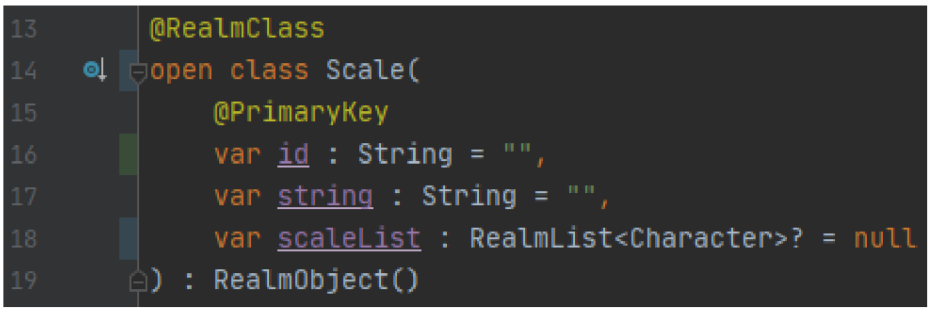
Obr. 4.4: Objekt databáze pro znak

Při prvním spuštění aplikace proběhne zápis znaků z předem připravených listů. Pomocí cyklů se databáze naplní sama, hodnota multiplikátoru pro speciální znaky je určovaná náhodně, čímž je alespoň částečně zajištěna originalita každého nástroje.

Nástroj umí rozpoznat 114 znaků v rozmezí MIDI čísel od 5 (subkontra F) do 118 (pětičárkované A#).

Stupnice

Stupnice tvoří další důležitou část nástroje, což přispívá tomu, aby převedený text byl melodický a netvořil jen změť náhodných tónů.

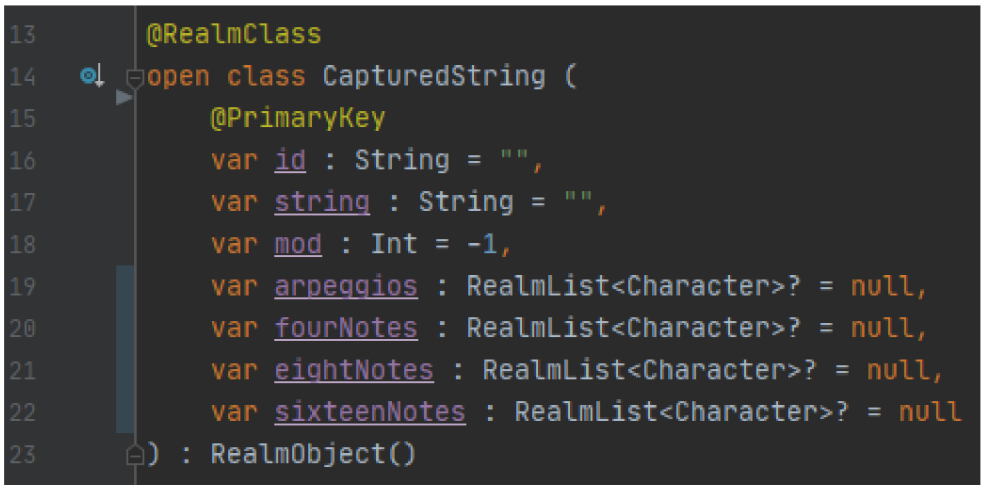


Obr. 4.5: Objekt databáze pro stupnici

Zápis stupnic proběhne opět při prvním spuštění aplikace hned po znacích. Každý znak lze považovat za základní tón stupnice, a jelikož jsou intervaly mezi tóny přesně dané, lze zápis značně automatizovat. Ze zvoleného rozsahu znaků je v databázi nástroje kolem 408 stupnic durových i diatonických, harmonických a melodických mollových.

Zachycený text

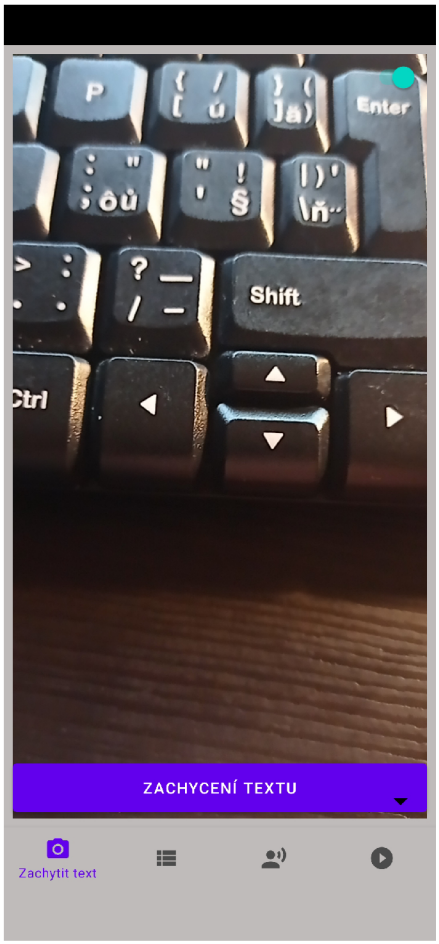
Databázový objekt pro zachycený text obsahuje kromě unikátního identifikátoru, znění textu a módu, ve kterém byl zachycen, i čtyři listy, ve kterých jsou uložené výše zmíněné znaky sloužící jako vstupní data pro Pure Data.



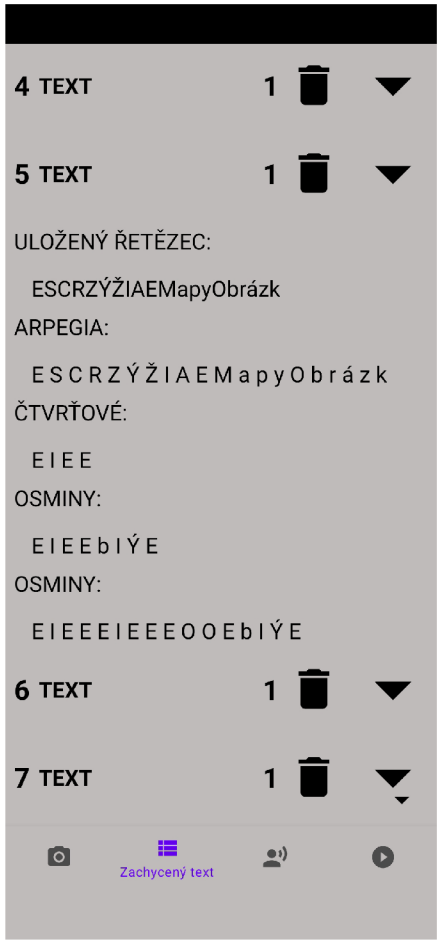
Obr. 4.6: Objekt databáze pro zachycený text

4.1.2 Zachycení textu a jeho převod na data

Grafické rozhraní pro zachycení textu zůstalo až na drobné estetické úpravy nezměněno. V pravém horním rohu je přepínač pro výběr, ve kterém módu se bude text zachycovat. Po stitsknutí tlačítka *zachytit text* je pomocí Google Vision API vytažen text a převeden na objekt typu *string*. Tento textový řetězec je přeposlán do další vrstvy aplikace, kde probíhá filtrování textu od mezer a znaků neobsažených v databázi. Dále je pomocí náhody vytažen jeden znak sloužící jako základní tón stupnice, která je pomocí něj opět nalezena v databázi. Následuje přiřazování MIDI hodnot podle algoritmu popsaného v návrhu aplikace a nakonec je celý textový řetězec spolu s hodnotami pro přehrávání uložen.



Obr. 4.7: Fragment pro zachycení textu



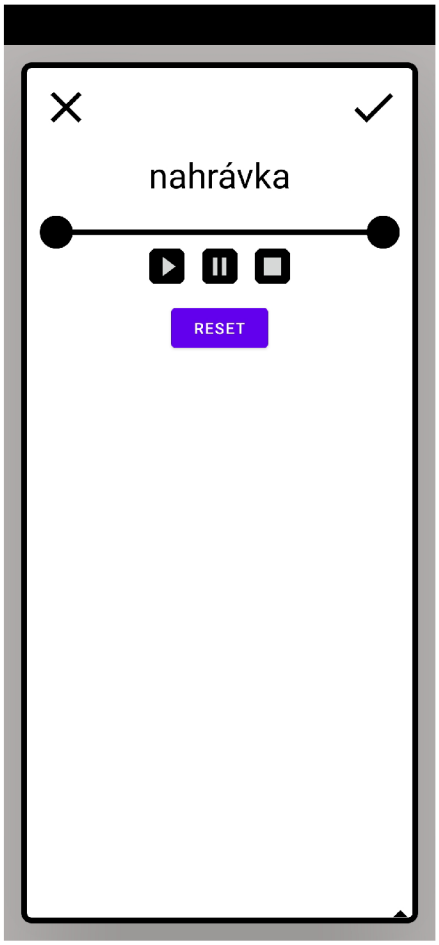
Obr. 4.8: Fragment pro zobrazení zachycených textů

Uživatel má možnost si zachycený text i s údaji prohlédnout na fragmentu pro zobrazení zachycených textů.

4.1.3 Nahrávání a úprava zvuku

Nezměněné, až na malé estetické a funkční prvky, zůstalo i grafické rozhraní pro záznam zvuku pomocí mikrofonu zařízení. V levém horním rohu je tlačítko pro začátek nahrávání. Po jeho stisknutí uživatel zadá jméno nahrávky a spustí se nahrávání pomocí mikrofonu zařízení. To je zprostředkováno knihovnou *WaveRecording*. Opětovné stisknutí tlačítka ukončí záznam zvuku a proběhne zápis dat do databáze a paměti telefonu. Dále je zde zobrazen seznam nahrávek. Stisknutím tlačítka pro přehrávání se nahrávka jednou přehraje.

Obr. 4.9: Fragment pro nahrávání



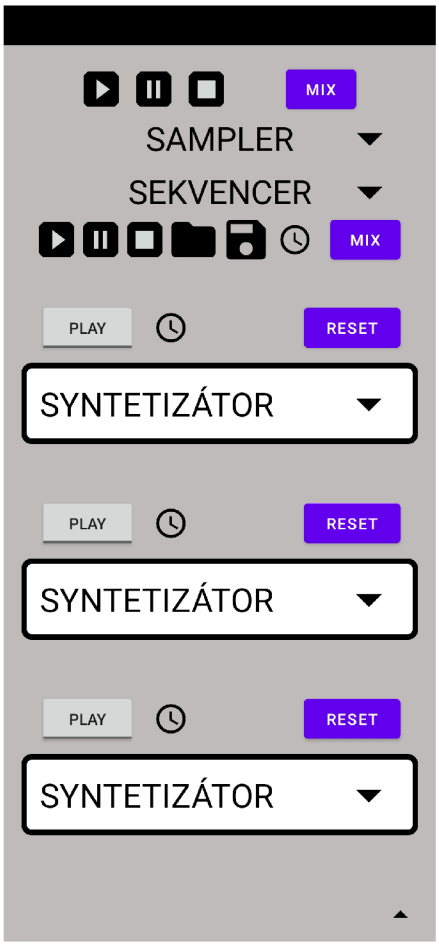
Obr. 4.10: Okno úpravy nahrávky

Při delším stisknutí jména nahrávky se otevře nové okno aplikace, kterou můžeme vidět na obrázku 4.10. Zde se nachází ovládání pro střih zvukových souborů. Pomocí táhel nastavených na počátečním, levém, bodu na hodnotě 0 a pravém bodu, tedy koncovém, který má proměnou hodnotu podle délky nahrávky, může uživatel určit nový začátek a konec. Pod táhly jsou tlačítka pro přehrávání. Původní zaznamenaný zvuk se zde přehrává ve smyčce a mění svoji délku podle nastavených bodů v reálném čase. Tlačítko *reset* slouží pro nastavení táhel do původní pozice. Pokud je uživatel

spokojen, může uložit upravenou nahrávku stisknutím tlačítka v pravém horním rohu. Nová nahrávka se potom pomocí knihovny *FFmpeg* upraví a uloží do paměti telefonu, název spolu s cestou k souboru se uloží do databáze. V levém horním rohu je tlačítko pro zrušení úprav.

4.1.4 Přehrávání

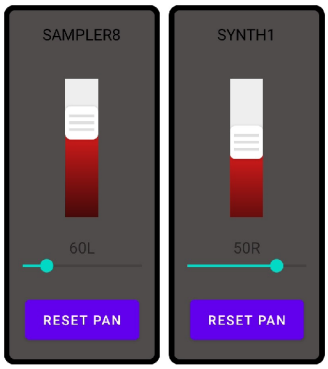
Jak již bylo zmíněno, největších úprav doznal fragment pro přehrávání, ve kterém může uživatel vytvářet kompozice. Oproti původnímu návrhu byl značně zjednodušen na pět komponent a obsahuje ovládání pro sampler, sekvencer a tři syntetizátory. Všechny jednotlivé části se dají schovat klepnutím na tlačítko s trojúhelníkem pro lepší přehlednost při tvorbě skladby.



Obr. 4.11: Fragment pro přehrávání

Mixážní pult

Mixážní pult nástroje slouží pro korekce hlasitosti a nastavení panoramatu při přehrávání. Grafické rozhraní je vcelku jednoduché a obsahuje pouze dvě táhla, název položky a tlačítko pro resetování panoramatu do výchozí pozice 0. Sekvencer nástroje má svůj vlastní mixážní pult, který je graficky totožný a slouží pro nastavení jednotlivých položek sekvenceru.



Obr. 4.12: Fragment pro přehrávání

Sampler

Komponenta pro sampler (obr. 4.13) se skládá z osmi tlačítek simulujících samplig pady. Pod nimi je tlačítko pro nastavení, po jehož stisknutí se zobrazí pod každým padem nové tlačítko. Stiskem se uživatel dostane do nastavení jednotlivých položek sampleru (obr. 4.14). Zde se nachází možnost pro výběr nahrávky, úpravu rychlosti přehrávání nebo zapnutí reverbu či delaye, popřípadě jejich kombinaci. Stisknutím tlačítka padu se začne přehrávat navolená skladba, pokud je podrženo delší dobu, spustí se přehrávání ve smyčce.

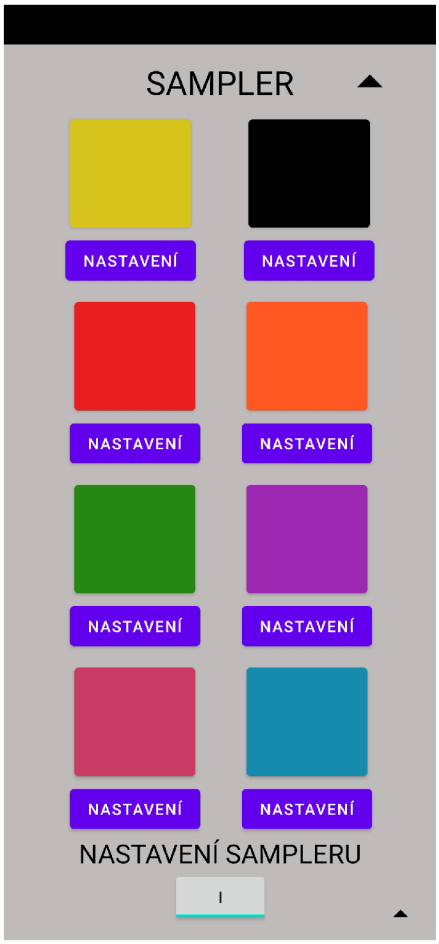
Sekvencer

V komponentě pro ovládání sekvenceru (obr. 4.15) můžeme nalézt tlačítka pro přehrávání, nastavení metronomu a mixu nebo možnosti uložit a načíst preset. Dále se skládá z deseti menších položek vlastní konstrukce sloužících pro ovládání jednotlivých částí bicí soupravy. Každá z položek obsahuje možnost vypnout přehrávaný řádek pomocí tlačítka *mute* nebo zapnout reverb či delay, popřípadě jejich kombinaci. Protože je šestnáct tlačítek pro ovládání přeci jen dost a rozměrově se nevejdou na displej telefonu, je možnost uvnitř komponenty rolovat do stran.

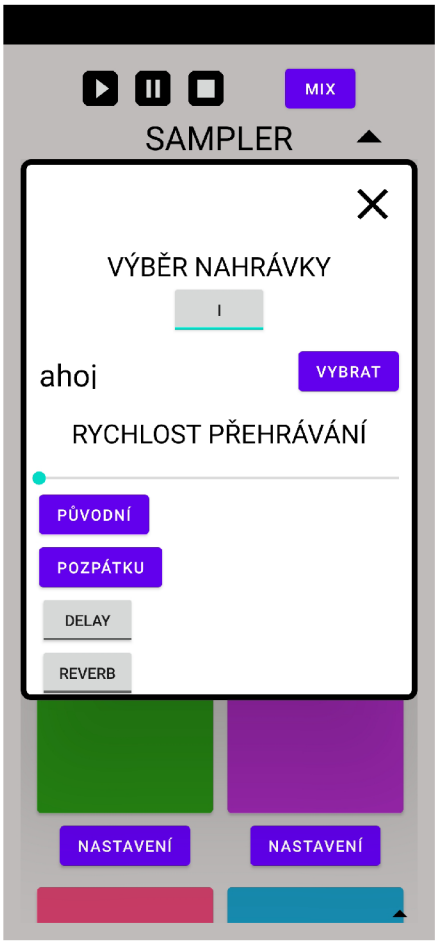
Z grafického rozhraní komponenty sekvenceru po zaškrtnutí tlačítka proběhne zapsání hodnot do výstupního listu, který je poté odeslán do Pure Data patche.

Syntetizátor

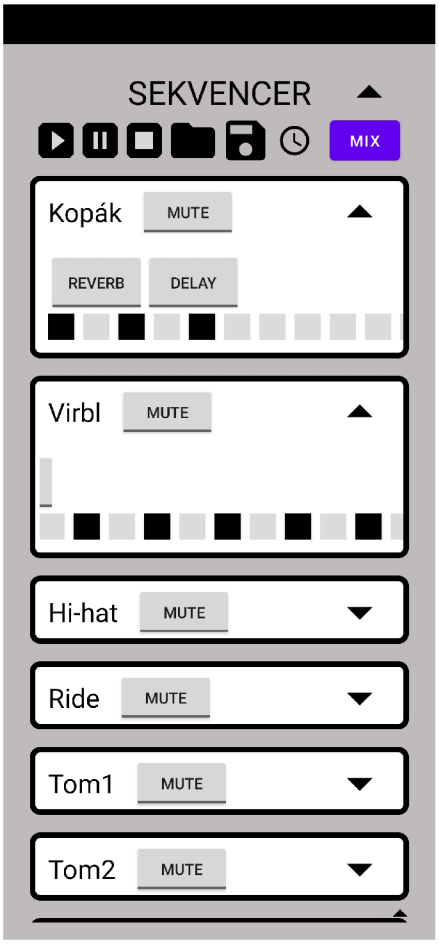
Za nejkomplexnější grafickou konstrukci celého nástroje lze považovat syntetizátor (obr. 4.16). Na horní liště komponenty jsou umístěny tlačítka pro přehrávání, nastavení metronomu nebo zresetování celé komponenty. Pod názvem je část určená pro výběr zaznamenaného textu a stylu přehrávání, následuje výběr tvaru vlny, zde je pro zjednodušení komponenty přiřazena i aditivní syntéza. Následuje možnost upravit obálku ADSR, výběr modulace a její nastavení nebo doplňující ovládání pro pulsní modulaci (pokud je navolen obdelníkový průběh), či aditivní syntézu.



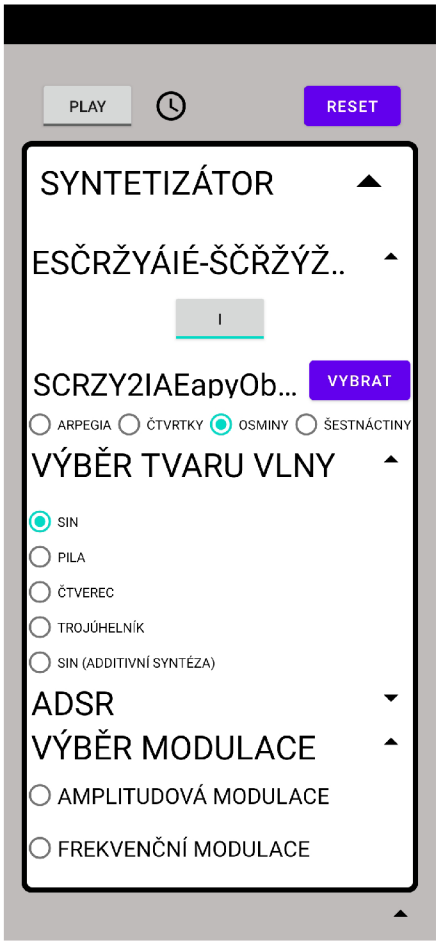
Obr. 4.13: Sampler



Obr. 4.14: Nastavení sampleru



Obr. 4.15: Komponenta sekvenceru



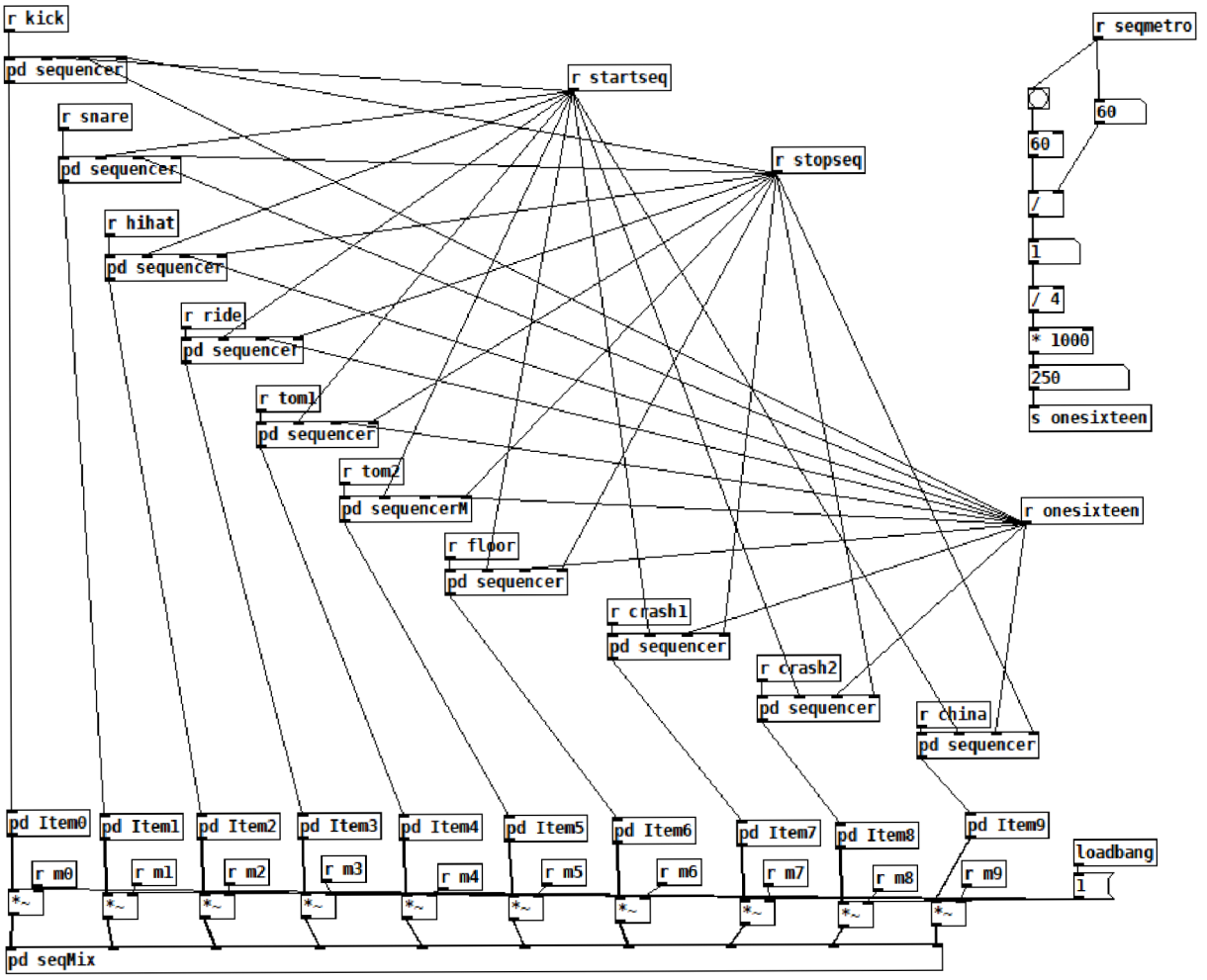
Obr. 4.16: Komponenta Syntetizátoru

Obr. 4.17: Komponenta Syntetizátoru

4.2 Pure Data

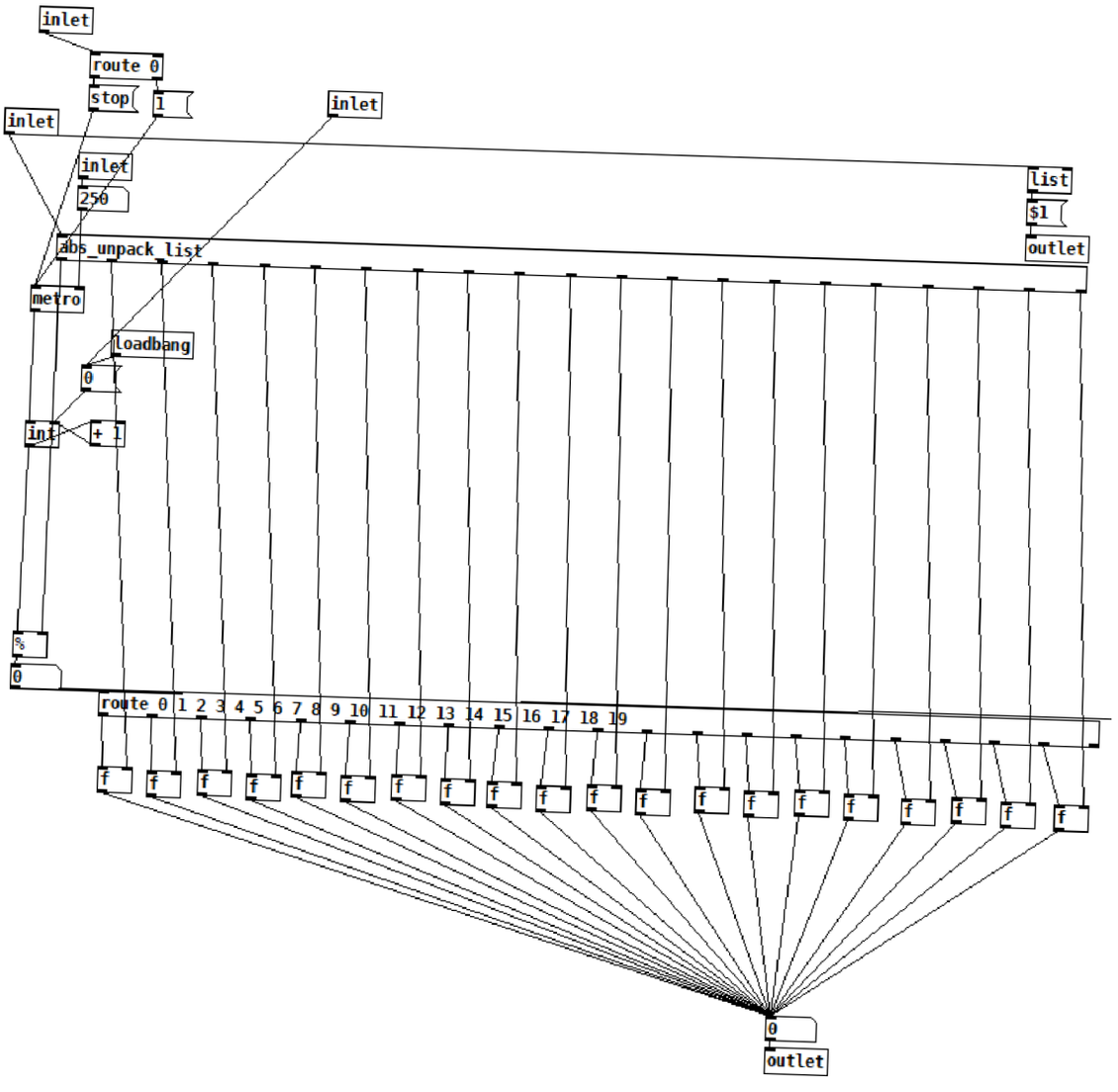
4.2.1 Sekvencer

Sekvencer je vytvořen vlastnoručně a složen ze subpatchů *sequencer, Item0 - 9, seqMix*. Na koncový bod *r seqmetro* je z Android části aplikace přijatá hodnota bpm a přepočítána na hodnotu jedné čtvrťové noty v milisekundách a předána do *sequencer*. Dále jsou zde recievery *r startseq* pro zapnutí nebo pozastavení přehrávání a *r stopseq* pro stopnutí sekvenceru. Vstupní body *m0 - 9* slouží pro funkci *mute*.



Obr. 4.18: Sekvencer

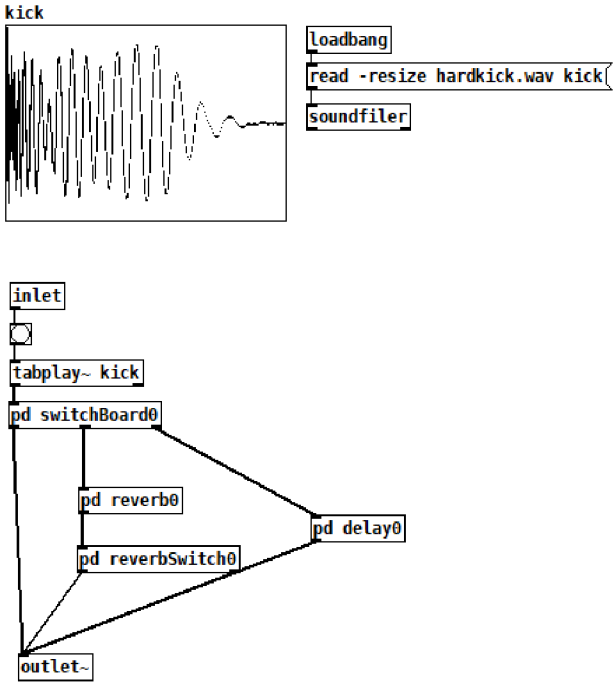
Po navolení a odeslání listu z aplikace, který obsahuje hodnoty 0 nebo 1, na receiver, jako je například *r kick*, je v *sequencer* pomocí abstrakce na obr. 4.19 list zpracován a uložen do proměnných. Při spuštění jsou hodnoty čteny postupně. Za abstrakci je ještě přiřazen objekt *route 1*, který propouští tok dat pouze, je-li hodnota 1. V subpatchi *Item* (obr. 4.20) je na základě propuštěných dat spuštěno přehrávání



Obr. 4.19: Abstrakce pro zpracování listů

stopy uložené v objektu *array* pomocí *tabplay~*. Uživatel má možnost k přehrávané stopě přiřadit delay nebo reverb, které jsou reprezentovány subpatchi *reverb0* a *delay0*. Po stisknutí příslušných tlačítek aplikace je v *switchBoard0* rozhodnuto, jakým výstupem tok dat zamíří. V komponentě *reverbSwitch0* se nachází další delay, který lze nasadit na výstup z *reverb0*. Data z *Item* přicházejí na vstup pro *mute* a dále na subpatch *seqMix*.

Protože nástroj nedovoluje změnu vzorku, je při spuštění patche pomocí objektů *loadbang*, *soundfiler* a zprávy *read -resize hardkick.wav kick* rovnou nahrávka nastavena a připravena k použití.



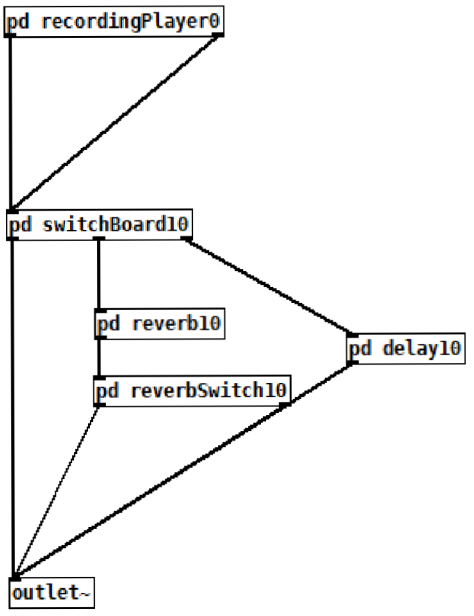
Obr. 4.20: Subpatch Item

4.2.2 Sampler

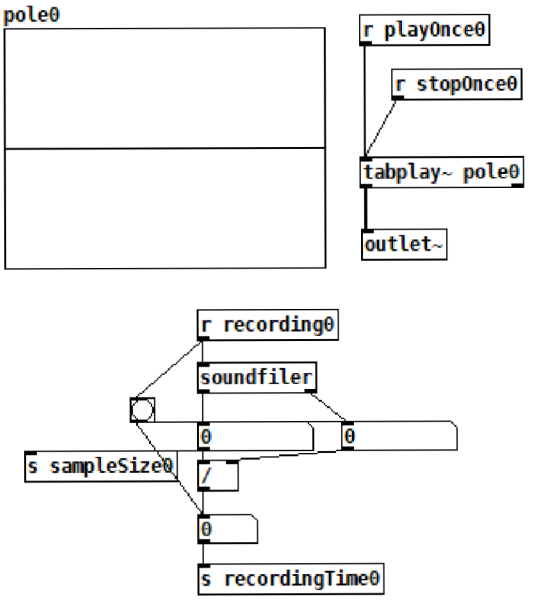
Patch pro sampler byl zbudován na základě návrhu zpracovaného v semestrální práci a upraven pro potřeby nástroje, jako je například přehrávání ve smyčce při úpravě nahrávek. I zde je možnost zapnout delay nebo reverb, popřípadě jejich kombinaci stejně jako u sekvenceru.

Po zvolení nahrávky je na přijímač *recording0* (obr. 4.22) vyslána zpráva ve tvaru *read -resize* s cestou k souboru. Objekt *soundfiler* zaznamená stopu do *pole0* a z výstupů je získána velikost vzorku spolu se vzorkovací frekvencí sloužící pro výpočet rychlosti přehrávání, tyto hodnoty jsou poté odeslány pomocí *sampleSize0* a *recordingTime0* dále do patche (obr. 4.23). Koncové body *playOnce0* a *stopOnce0* ovládají možnosti jednoduchého přehrátí vzorku.

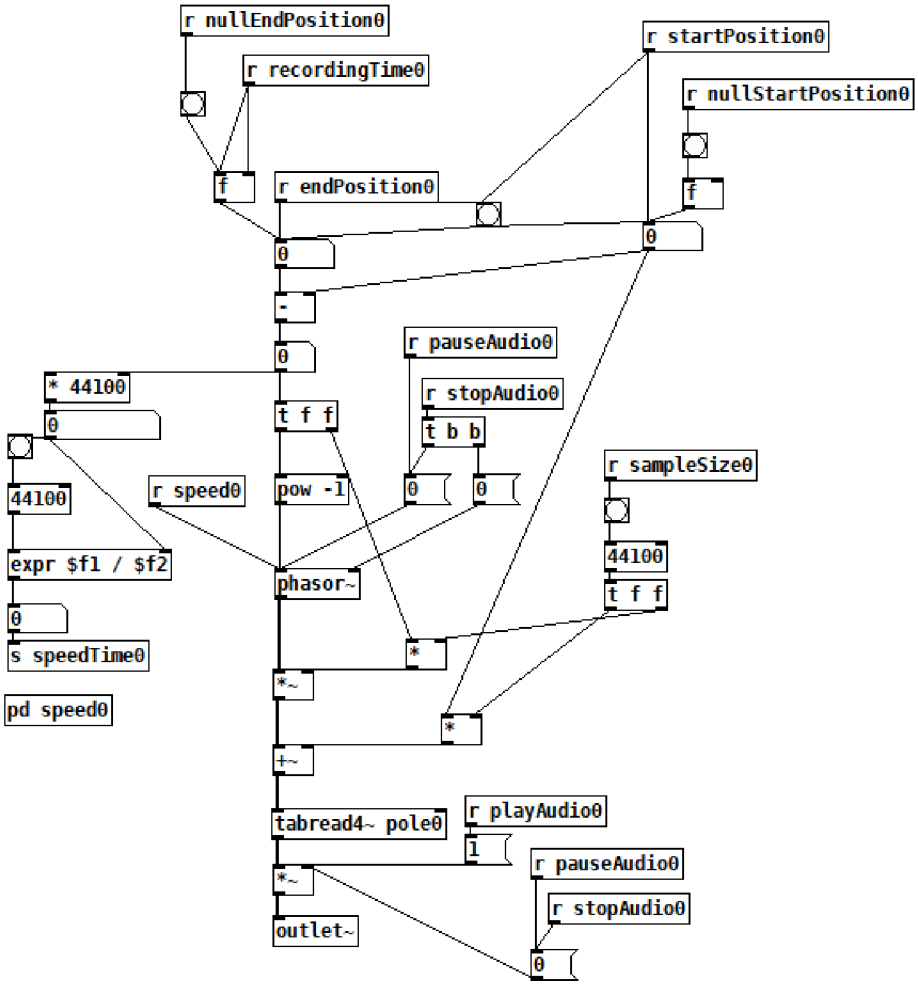
Pro nastavení nových koncových bodů slouží receivery *endPosition0* (obr. 4.23) a *startPosition0*, pro resetování zase *nullEndPosition0* a *nullStartPosition0*. Odečítáním počátečního bodu od koncového je v sekundách určena nová délka nahrávky, ze které je získána velikost vzorku a nová rychlost přehrávání, jejíž hodnota je pomocí *speedTime0* odeslaná do subpatche *pd speed0*. Zde probíhají výpočty nastavení rychlosti při jednotlivém přehrávání nebo smyčce. Výstup tohoto patche je přes *speed0* odeslán na *phasor~*.



Obr. 4.21: Patch pro práci s nahrávkami



Obr. 4.22: Subpatch pd recordingPlayer



Obr. 4.23: Subpatch pd recordingPlayer

4.2.3 Syntetizátor

Subpatch syntetizátoru (obr. 4.25) je částečně sestaven pomocí modifikovaných konstrukcí převzatých od Kavana[8] a jedná se o nejsložitější patch v celém nástroji.

V rámci možnosti jiného časového rozsahu přehrávané noty určené multiplikátorem je na vstup abstrakce pro zpracování listů posílán i list násobičů pomocí *synthMultipList0*. Výstupní hodnota je přes *multip0* poslána do subpatche *releaseCount* stejně jako informace o začátku nového tónu.

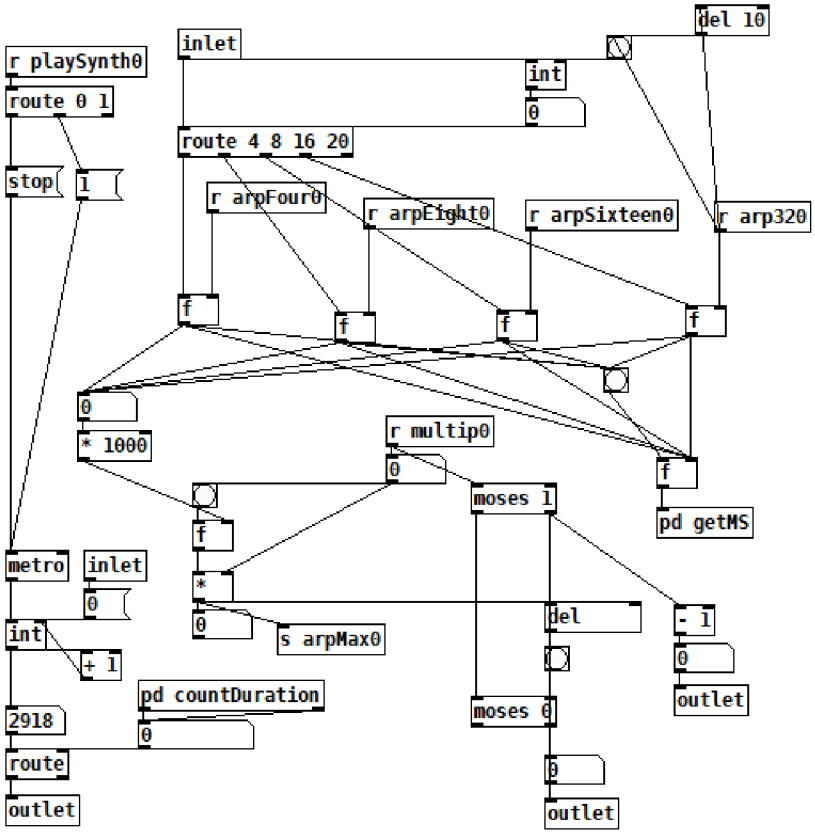
Subpatch *arpMetroCount0* slouží pro přepočet bpm na milisekundy, jednoduchým dělením jsou získávány délky trvání not pro čtvrťové, osminové, nebo šestnáctinové noty a posílány do *releaseCount*, kde se uloží do proměnných.

Objekt *releaseCount* (obr. 4.24) je vlastní konstrukce a slouží pro výpočet konce trvání tónu. List hodnot obsahující čísla MIDI not má k sobě přiřazenou hodnotu na-

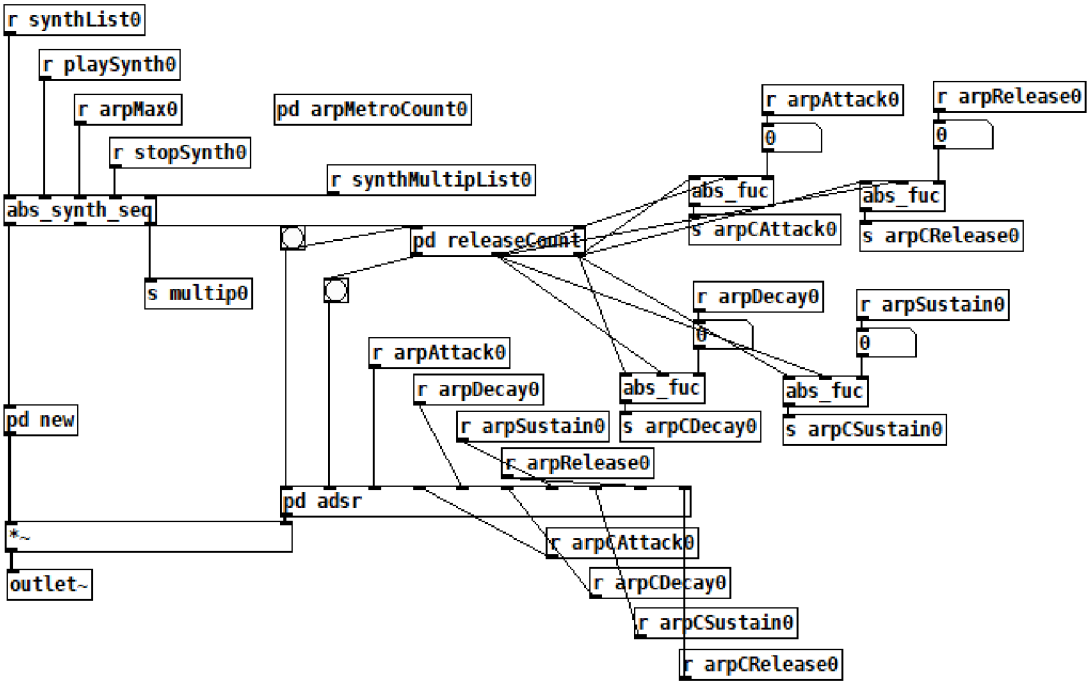
bývající 4, 8, 16 a 20, reprezentující arpeggia, čtvrtiny, osminy a šestnáctiny, pomocí kterých je vybrána základní délka trvání noty a odeslána na vstup *abs_synth_seq* přes *arpMax0*. Současně je nasteveno i ADSR. Z jeho prvních tří složek se v subpatchi *countDuration* spočítá čas, za který má být vyslán řídící signál, který spustí release. Při přehrávání každého nového tónu je délka jeho trvání násobena o hodnotu multiplikátoru a opět přeposlána přes *arpMax0*. Výstupy subpatche vedoucí do abstrakce *abs_fuc* odečítají nebo přičítají hodnoty k již nastaveným složkám ADSR, aby byl zachován jejich poměr.

V subpatchi *new* (obr. 4.25) jsou čtyři generátory průběhů: generátor sinusového, pilového, obdélníkového a trojúhelníkového průběhu. U obdélníku má uživatel možnost nastavit pulzně šířkovou modulaci, pomocí níž je upravena střída signálu.

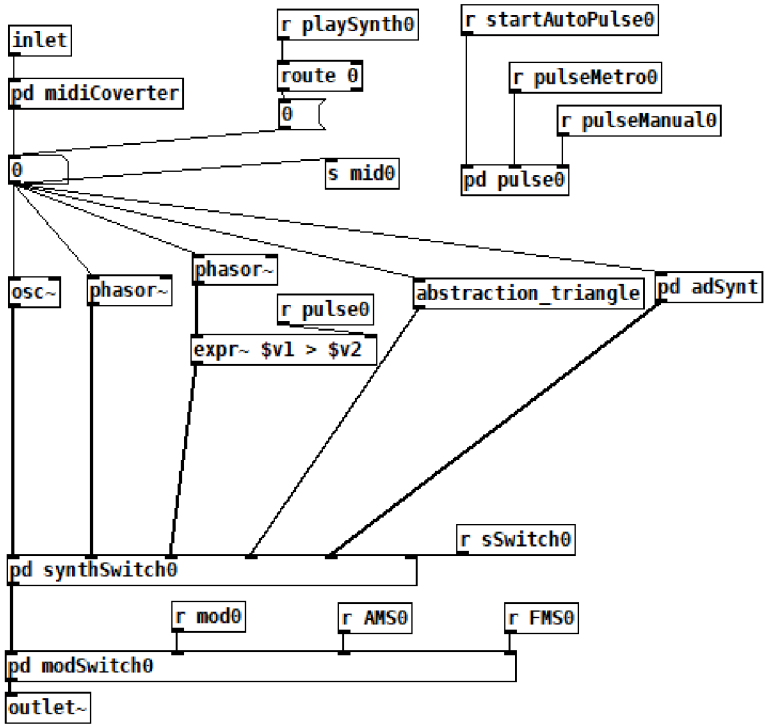
Přepínání mezi jednotlivými generátory je řešeno pomocí *synthSwitch0* do kterého je přes *sSwitch0* posílána hodnota navolená uživatelem v rozmezí 0 až 4. Vybraný průběh pokračuje do *modSwitch0*, který řeší přepínání mezi jednotlivými druhy modulací. Přijímač *mod0* slouží pro výběr frekvenční nebo amplitudové modulace, objekty *AMS0* a *FMS0* zase slouží pro výběr průběhu modulační vlny. Čistému nebo zmodulovanému zvuku je poté přiřazena obálka ADSR.



Obr. 4.24: Subpatch releaseCount



Obr. 4.25: Subpatch syntetizátoru



Obr. 4.26: Subpatch new

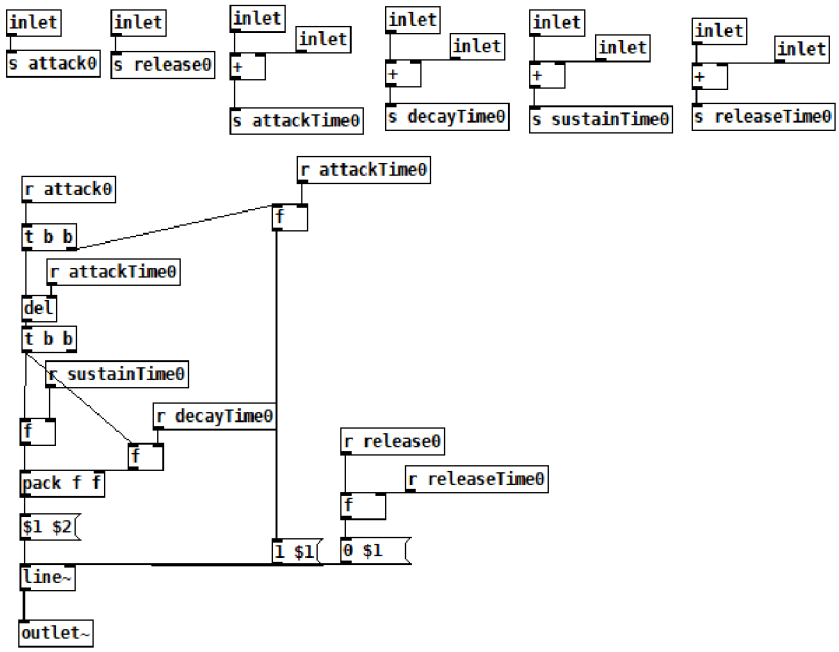
ADSR

Subpatch syntetizátoru *adsr* (obr. 4.27) určuje charakter průběhu tónu pomocí čtyř parametrů: Attack, což je doba, za kterou signál dosáhne svého maxima, decay - doba sestupu na ustálenou hodnotu, sustain - čas, po který je signál na ustálené hodnotě, a release, což je doba, za kterou signál dosáhne minima. S lehkou modifikací je subpatch převzat od Kavana[8].

Pomocí číselných hodnot je nastavená úroveň jednotlivých částí. Attack je potom spuštěn pomocí přijetí řídícího signálu, vysílaného při začátku nového tónu, na vstupu *attack0*.

Aditivní syntéza

Konstrukce aditivní syntézy je převzatá od Kavana[8] a upravena pro potřeby nástroje. Nachází se zde 17 sinusových oscilátorů vytvořených pomocí objektu *osc~* se vstupy pro ovládání poměru frekvence oproti fundamentu (reprezentován prvním oscilátorem) v rozmezí od 0 do 4 a možností měnit velikost amplitudy od 0 do 1. Pro



Obr. 4.27: Subpatch adsr

větší hudební možnosti je přidáno ještě dalších 34 zdrojů sinusového signálu kopírujících nastavení předešlých oscilátorů, u kterých má uživatel také možnost násobit frekvence, ale pouze pro celek v rozmezí od 0 do 2.

Frekvenční Modulace

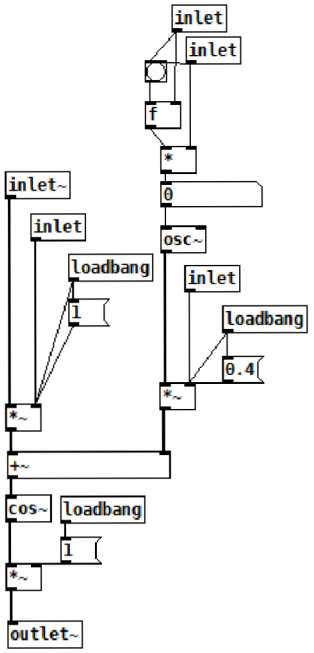
Patch pro frekvenční modulaci (4.28) je postaven na základě zapojení podle Kavana[8] a modifikován o možnost nastavení poměru frekvence modulační vlny pro zajištění stejného zvuku při přehrávání.

Amplitudová Modulace

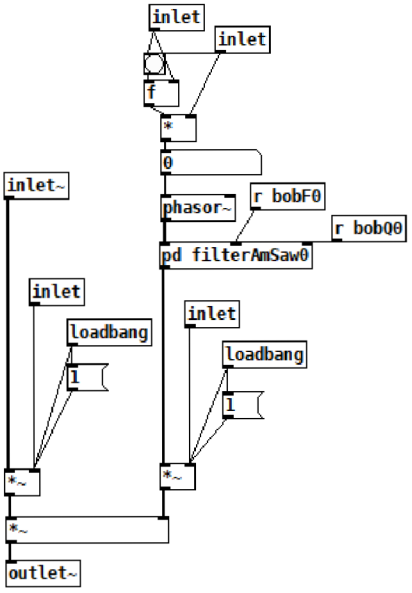
Struktura patche pro amplitudovou modulaci (4.29) je vytvořena podle Kavana[8] a modifikovaná stejným způsobem jako frekvenční modulace. V rámci zajímavějšího výsledného zvuku má uživatel možnost zapnout pro modulaci obdelníkem nebo pilou filtr typu dolní propust s rezonancí reprezentovaný objektem *bob~*, u kterého lze nastavit mezní kmitočet od 0 do 3000 Hz a činitel jakosti Q od 0 do 4.

4.2.4 Delay

Konstrukce delaye (obr. 4.30) je převzatá od Kavana[8]. Na vstupu je audio signál rozdělen na dva. Jeden pokračuje jako původní a druhý je pomocí *delread~* zpož-

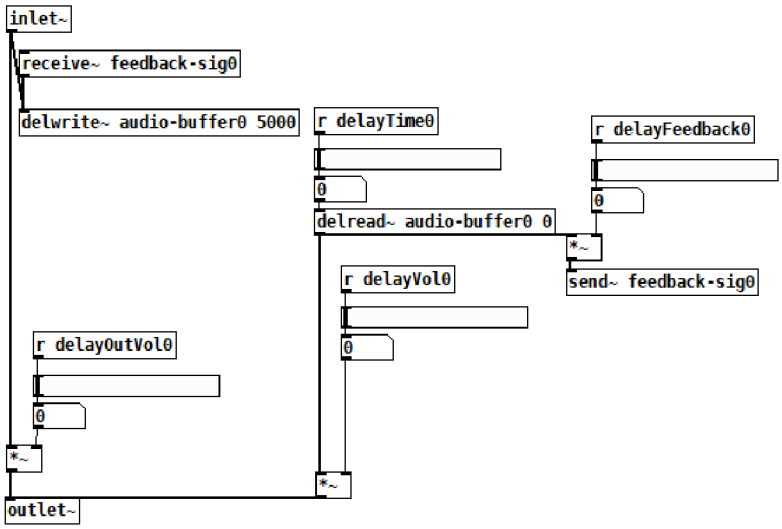


Obr. 4.28: Subpatch pro frekvenční modulaci



Obr. 4.29: Subpatch pro amplitudovou modulaci

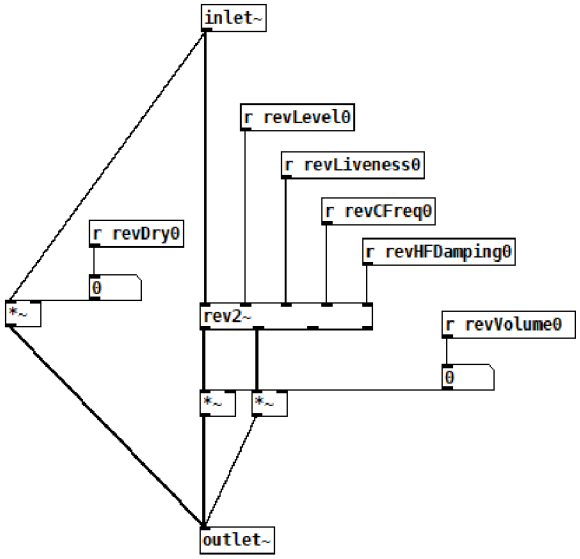
děn. Přijímač *delayTime0* nastavuje hodnotu zpoždění v ms, *delayFeedBack0* zase jeho intenzitu. Pomocí *delayVol10* je nastavena hlasitost zpožděného signálu a *delayOutVol0* slouží pro zesílení či zeslabení původního.



Obr. 4.30: Subpatch delay

4.2.5 Reverb

Patch reverbu je postaven za použití objektu *rev2~*, který je obsažen v prostředí Pure Data. Na vstupu se audio signál rozdělí na dva totožné audio signály. Jedna část reprezentující původní signál je zesílena hodnotou *revDry0*, druhá jeho část putuje do *rev2~*, na výstupu je potom zesílena o hodnotu *revVolume0*.



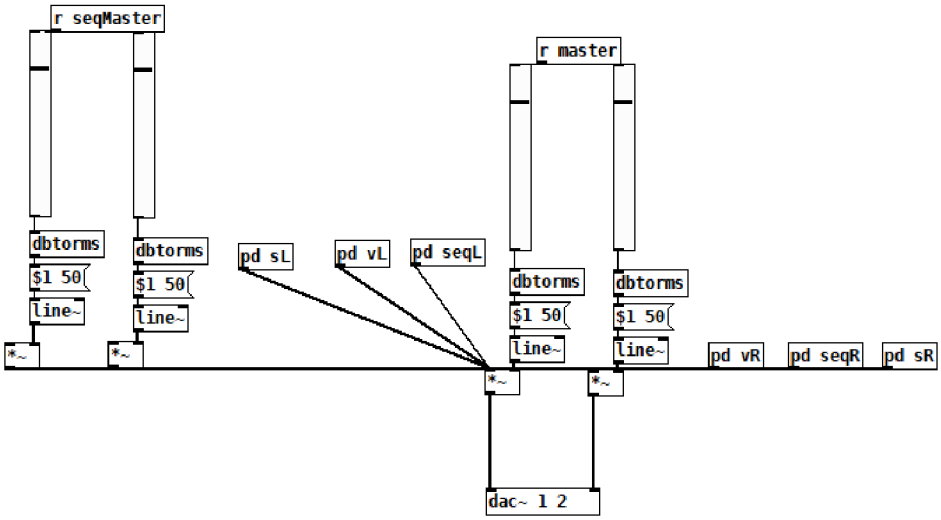
Obr. 4.31: Reverb

4.2.6 Mixážní pult

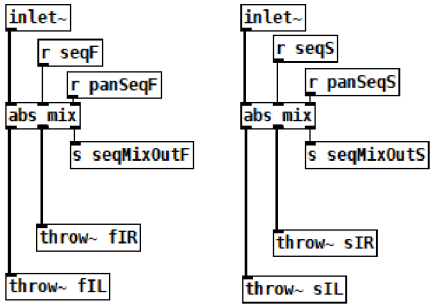
Mixážní pult aplikace je postaven za použití několika převzatých konstrukcí od Kavana[8], jako jsou abstrakce *abs_pan* a *abs_mix* (obr. 3.34), který pomocí objektu *dbtorms* převádí signál v dB na průměrnou hodnotu signálu RMS.

Pro jednotlivé části nástroje je vytvořen vlastní mixážní pult (*synthMix, seqMix, voiceMix*) se vstupy pro ovládání hlasitosti a panoramatem. Výstupy jsou pomocí objektů *throw~* a *catch~* přiváděny do subpatchů *sL, VL, seqL* pro levý a *sR, vR, seqR* pro pravý kanál abstrakce *abs_master*, kde se spojují dohromady a jsou přivedeny na objekt *dac~* sloužící jako audio výstup celého nástroje.

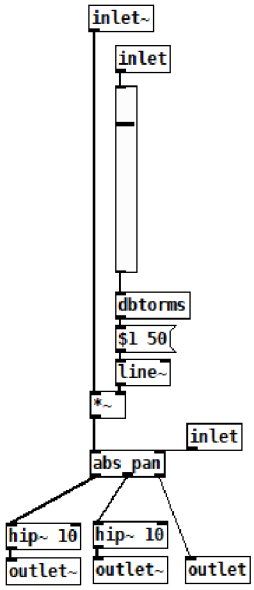
Protože je sekvencer složený z částí bicí soupravy tvořící celek, má svůj vlastní *master*.



Obr. 4.32: abs_master



Obr. 4.33: Subpatch seqMix



Obr. 4.34: Převzatá abstrakce abs_mix

4.3 Hudební a jiné možnosti nástroje

Možnosti nástroje jsou s ohledem jeho konstrukci omezené. Přesto lze nástroj v praxi použít několika způsoby.

Díky spojení sekvenceru, sampleru a syntetizéru je možné nástroj použít pro tvorbu minimalisticky laděných skladeb či jako zdroj inspirace pro hledání nových postupů při tvorbě hudby, které v jistých případech plynou z náhodného výběru hodnot pro přehrávání při převodu textu na data.

Další využití nástroj nalezne díky zabudovanému sekvenceru jako doprovodný rytmický nástroj nebo kombinací všech částí jako nevšedního zpestření kompozic realizovaných pomocí tradičních nástrojů.

Nástroj lze využít i čistě experimentálně. Tím je myšleno především prozkoumání možností elektronické hudby.

Jelikož se jedná o aplikaci pro platformu Android, po umístění do obchodu Google Play by měli uživatelé volný přístup k aplikaci. Tu by tím pádem bylo možné využít čistě jen pro zábavu.

4.4 Možnosti rozšíření nástroje

Protože je nástroj primárně zaměřen na zhudebnění textu, je jedna z možností rozšíření přidat do aplikace rozsáhlejší a komplexnější algoritmy přiřazování hodnot pro

přehrávání při zachycení textu a jeho převodu na data nebo možnosti, aby si tento algoritmus mohl uživatel upravovat sám. S tím souvisí i jiné formy záznamu textu než pomocí fotoaparátu zařízení, čímž je myšlen například záznam z klávesnice. V tomto případě by nebylo od věci pomocí zaznamenávaného textu, kromě hodnot pro přehrávání, měnit v reálném čase i parametry nástroje.

Jako zajímavý prostředek rozšíření se určitě jeví i možnost použití jiného než rovnoměrně temperovaného ladění.

Sampler i sekvencer nástroje by se mohly rozrůst o nové prvky ovládání, případně možnosti nahrávat a přehrávat smyčky.

Při tvorbě zvuku by bylo možné vyzkoušet i jiné druhy syntézy, jako je například granulární syntéza, nebo možností vytvarovat si výslednou vlnu pomocí grafického rozhraní. Jistě zajímavým rozšířením by mohlo být více kombinací pro modulaci použitím již zmodulované vlny nebo šumu, popřípadě více parametrizovat celé ovládání této části nástroje.

Jedním z možných směrů vylepšení aplikace by mohla být optimalizace kódů v jazyce Kotlin a Pure Data.

Závěr

Cílem práce bylo navrhnout a realizovat mobilní aplikaci představující experimentální softwarový hudební nástroj na platformě Android, který využívá pro tvorbu zvuku text zachycený pomocí vestavěného fotoaparátu zařízení v kombinaci za použití syntetizéru, sampleru a sekvenceru.

V části zabývající se teorií byly stručně rozebrány druhy syntéz, operační systém Android, jeho vývojové prostředí a programovací jazyky Kotlin a Pure Data, dále MIDI a Android knihovna pro ovládání Pure Data.

Ve třetí kapitole byl popsán návrh aplikace ve vývojovém prostředí Android Studio. Je zde obsažen návrh uživatelského rozhraní, senzorů, algoritmů pro převod a ukládání textu na data potřebná k přehrávání, nahrávání zvuku pomocí zařízení telefonu, záznam a načtení presetů, databáze a jejích objektů či komunikace této části aplikace s patchi psanými v jazyce Pure Data. Dále je zde popsán stručný návrh aplikace v jazyce Pure Data.

Při samotné realizaci byl shledán návrh aplikace, vycházející ze semestrální práce, jako nedostačující, a to především v části aplikace, která měla být naprogramována v jazyce Pure Data. Dále došlo oproti původnímu návrhu k několika změnám při ukládání textu, který slouží jako vstupní data pro tvorbu zvuku, úpravě a zjednodušení uživatelského rozhraní a rozšíření aplikace o nové funkcionality. Bylo také upuštěno od některých konceptů, jmenovitě od možnosti ukládání přednastavení, ovládání táhel a potenciometrů aplikace pomocí zabudovaných senzorů, či focení textu pomocí aplikace pro fotoaparát. Ve většině případů se tyto koncepce v praxi příliš neosvědčily.

V rámci realizace byly dále popsány hudební a jiné možnosti nástroje pramenící z praxe a možnosti, jak nástroj dále rozšiřovat a vylepšovat.

Výsledná aplikace je plně funkční a stabilní, pro běžného uživatele graficky přívětivá, snadno ovladatelná a intuitivní.

Literatura

[1] RUSS, Martin. *Sound synthesis and sampling* 3.rd ed. Boston: Elsevier/Focal Press, c2009. ISBN 978-0-240-52105-3

[2] SCHIMMEL, Jiří. *Studiová a hudební elektronika*, druhé vydání. Vysoké učení technické v Brně Fakulta elektrotechniky a komunikačních technologií Ústav telekomunikací, Brno 2015. 197 s. ISBN 978-80-214-4452-2

[3] *Oficiální stránka pro vývoj na platformě android* [online]. Dostupné z URL: <`https://developer.android.com/`>

[4] *Oficiální stránka PureData* [online]. Dostupné z URL: <`https://puredata.info/`>

[5] *Oficiální stránka Realm Databáze* [online]. Dostupné z URL: <`https://realm.io`>

[6] *Oficiální stránka JetBrains* [online]. Dostupné z URL: <`https://www.jetbrains.com/`>

[7] *Stránka aplikace LangoRythm* [online]. Dostupné z URL: <`https://kickthejetengine.com/langorhythm/`

[8] KAVAN, Jan. *Pure Data: platforma pro tvorbu interaktivního díla*, první vydání. Janáčkova akademie múzických umění v Brně, 2013. ISBN 978-80-7460-033-3.

5 Obsah elektronické přílohy

V příloze naleznete instalační soubor ve formátu .apk, složku projetku Android Studia a Pure Data.
rychtecky.apk
rychtecky_AS
rychtecky_PD