VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky
a komunikačních technologií

BAKALÁŘSKÁ PRÁCE

Brno, 2019 Filip Barák

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 POHYBEM TELEFONU

EXPERIMENTAL SOFTWARE MUSICAL INSTRUMENT ON ANDROID PLATFORM WITH ABILITY TO CONTROL PARAMETERS BY MOVING PHONE

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR
Filip Barák

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

BRNO 2019

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Bakalářská práce

bakalářský studijní obor **Audio inženýrství**
Ústav telekomunikací

Student: Filip Barák ***ID:*** 184181
Ročník: 3 ***Akademický rok:*** 2018/19

NÁZEV TÉMATU:

Experimentální softwarový hudební nástroj na platformě Android s možností ovládání některých parametrů pohybem telefonu

POKYNY PRO VYPRACOVÁNÍ:

Cílem práce je kompletně naprogramovat mobilní aplikaci - experimentální softwarový hudební nástroj pro platformu Android. Experimentálnost spočívá v kombinaci jednoduchého syntetizéru a sekvenceru v rámci mobilní platformy; hudební parametry lze ovládat mj. i pohybem telefonu.

DOPORUČENÁ LITERATURA:

[1] Electronic abd 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: 1.2.2019 ***Termín odevzdání:*** 27.5.2019

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

prof. Ing. Jiří Mišurec, CSc.
předseda oborové rady

Abstrakt

Cílem této práce je vytvořit aplikaci na platformě Android, která se vyznačuje netradičním způsobem ovládání hlavního melodického signálu – pohybem telefonu. Aplikace zároveň s tím umožňuje upravovat parametry zvuku na displeji telefonu. Další důležitou funkcí je možnost skládat a později přehrávat akordy, nebo jiné jednoduché souzvuky. Návrhu předchází teoretický úvod, kde stručně rozebírám hudební pojmy a parametry aplikace, kterou jsem vyvíjel v Android studiu v jazyce Java. Dále se zmiňuji o samotných způsobech realizace aplikace včetně ukázek kódů.

Klíčová slova

Hudební nástroj, Android, Java, pohyb, generátor, AudioTrack

Abstract

The purpose of this thesis is to create an application on the Android platform, which is characterized by an unconventional way of controlling the main melodic signal – by moving the phone. At the same time, the application allows you to edit the audio parameters on your phone's display. Another important feature is the ability to compose and later play chords, or other simple harmonies. The application design is preceded by a theoretical introduction, where I briefly analyze the musical terms and parameters of the application that I developed in the Android studio, Java language. Furthermore, I mention the actual methods of application implementation including code samples.

Keywords

Musical Instrument, Android, Java, Motion, Generator, AudioTrack

Bibliografická citace:

BARÁK, Filip. Experimentální softwarový hudební nástroj na platformě Android s možností ovládání některých parametrů pohybem telefonu. Brno, 2019. Dostupné také z: https://www.vutbr.cz/studenti/zav-prace/detail/116058. Bakalářská práce. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací. 53 stran, 2 přílohy. Vedoucí práce Dan Dlouhý.

Prohlášení

Prohlašuji, že svou závěrečnou práci na téma Experimentální softwarový hudební nástroj na platformě Android s možností ovládání některých parametrů pohybem telefonu 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 závěrečné práce dále prohlašuji, že v souvislosti s vytvořením této závěrečné práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

V Brně dne **20. května 2019** .....

podpis autora

Poděkování

Děkuji vedoucímu bakalářské práce doc. Ing. MgA. Mgr. Dan Dlouhý, Ph.D. za rady a velice rychlou komunikaci při zpracovávání mé bakalářské práce.

V Brně dne **20. května 2019** ……

podpis autora

Obsah

Seznam obrázků

Seznam tabulek

Seznam výpisu kódů

ÚVOD

V této práci se věnuji návrhu a zpracování softwarového hudebního nástroje na platformě Android. Operační systém Android jsem si vybral z důvodu jeho otevřenosti a snadného přístupu jednak k informacím, jednak k programovacím nástrojům a také díky tomu, že jsem sám aktivním uživatelem.

V mé aplikaci se chci věnovat tvorbě jednoduchého hudebního nástroje založeného na generování harmonického signálu a jeho následné úpravě. Právě úprava generovaného signálu bude probíhat nejen přes dotykový displej zařízení, ale také pomocí akcelerometru, tedy pohybem zařízení.

Aplikace bude generovat jednoduchý nebo složený harmonický signál, pomocí kterého si uživatel předvolí frekvenci tónů, se kterými bude chtít pracovat. Tyto tóny se mohou složit v akord nebo jiný souzvuk. Uživatel bude moci dále řídit frekvenci hlavní melodie pomocí náklonu zařízení vpravo nebo vlevo z horizontální polohy. Náklon doprava znamená zvýšení frekvence, náklon doleva frekvenci sníží.

Na začátku práce zmiňuji stručný popis nejdůležitějších komponentů aplikace pro Android a základní pojmy syntetické hudby. V návrhu aplikace se věnuji hlavně uživatelskému prostředí a způsobu ovládání aplikace. V kapitole Realizace aplikace popisuju jednotlivé funkční části aplikace a případně doplňuji o úryvky ze svého kódu. Na konci kapitoly popisuji také problémy, se kterými jsem se musel vypořádat v průběhu vývoje. V Závěru shrnuji dosažený výsledek a případné problémy nebo nedostatky.

1 TEORETICKÁ ČÁST: OS ANDROID

V Teoretické části stručně rozeberu OS Android a jeho historii, poté se zaměřím na popis nejdůležitějších komponentů aplikace pro Android a jeho vývojového prostředí včetně součástí usnadňující vývoj.

1.1 Stručná historie OS Android

Android je rozsáhlý open source operační systém postavený na linuxovém jádře a Android API (které komunikuje s jádrem) doplněný o grafické uživatelské rozhraní (GUI) a aplikace. Systém je navržen tak, aby pracoval na různém typu hardwaru (mobilní telefony, tablety, počítače, nositelná elektronika, chytré domácnosti, …).

Operační systém začal vznikat v roce 2003 založením společnosti Android Inc., nyní systém vyvíjí konsorcium OHA zaštítěné Googlem. První komerčně dostupný telefon byl HTC Dream představený v roce 2008 se kterým zároveň přišlo Android SDK 1.0 pro širokou veřejnost.

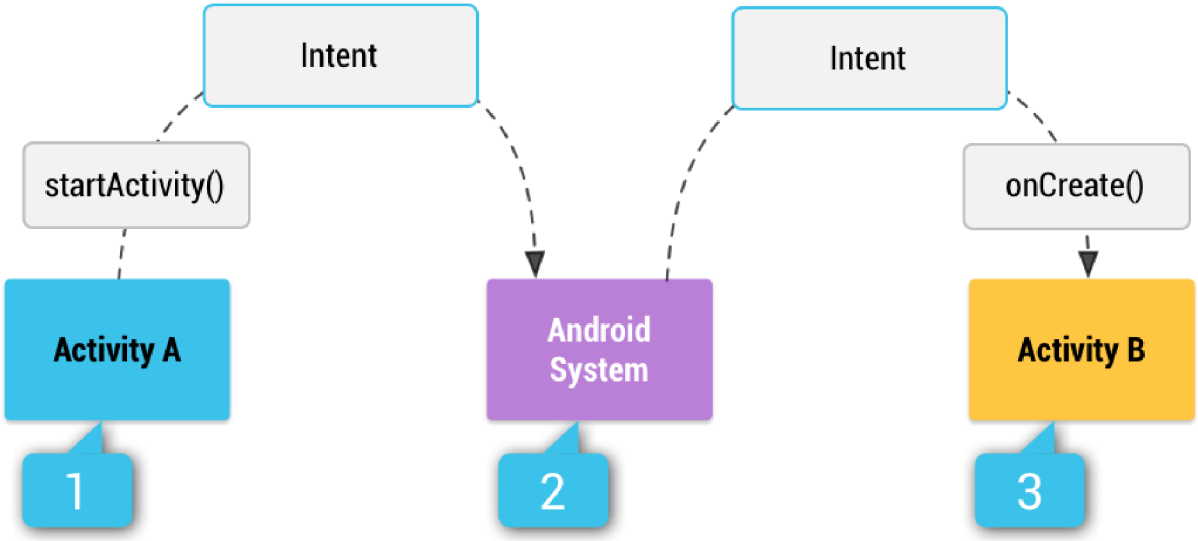
Aktuální verze androidu je 8.1 Oreo. Zásadní vlastnost tohoto systému je jeho ekosystém, který zahrnuje základní aplikace jako je kalendář, email, internetový prohlížeč, seznam kontaktů, … a nabízí jejich plnou synchronizaci s jiným zařízením podporující tento ekosystém nebo s webovou verzí. [1]

1.2 Komponenty Android aplikace

Vybral jsem tyto 4 nejdůležitější součásti každé Android aplikace, o kterých budu dále mluvit při samotné realizaci. [2]

1.2.1 Intent

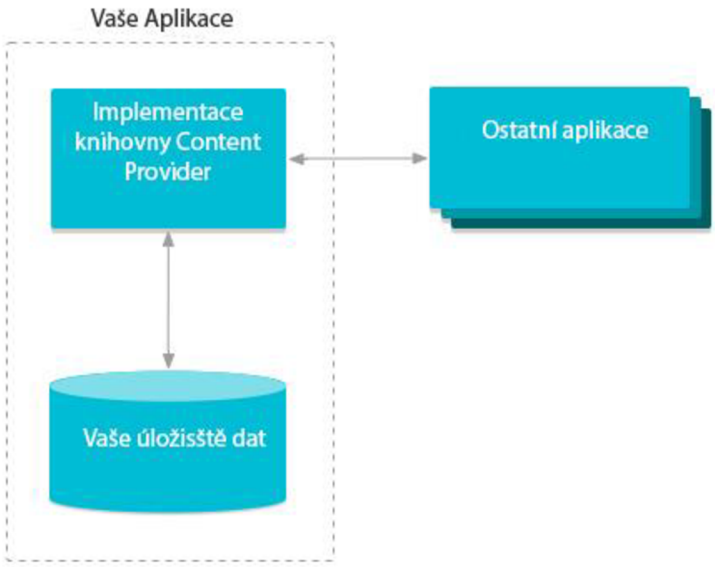
Intent zajišťuje předávání zpráv mezi komponentami aplikace pro provedení určité akce. Rozdělujeme intenty explicitní a implicitní. Explicitní Intent ví přímo jakou třídu nebo knihovnu spustit bez optání systému nebo uživatele, Implicitní intent dá na výběr systému, který provede na základě podnětu akci (spustí jinou aplikaci, vytvoří nový soubor, …). [3]



Obr. 1.1: Vytvoření Activity pomocí Intent [4]

1.2.1 Content providers

Content providers řídí přístup k datům pro aplikace (ukládání a čtení), což je jediný přístup k datům například z jiné aplikace. Důležité je, že pracuje s Permissions[1] a díky tomu získává přístup do databází jiných aplikací nebo systému. [5]



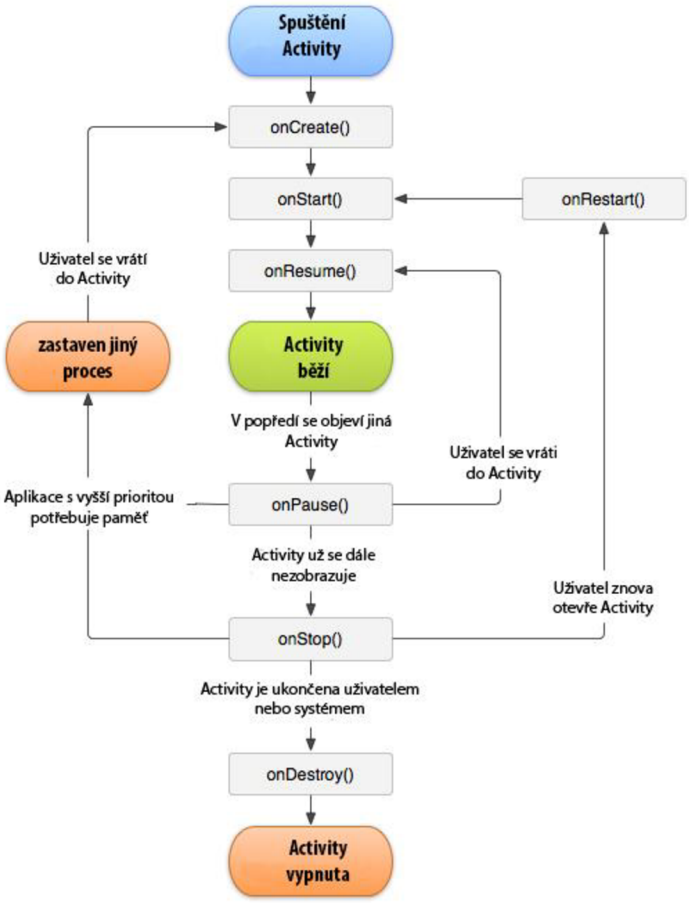
Obr. 1.2: Znázornění přístupu k datům pomocí Content providers [6]

[1] Permissions neboli oprávnění umožňuje aplikaci přistupovat do systému nebo databáze jiných systémů, oprávněná řídí a povoluje uživatel a souhlasí s nimi při instalaci aplikace či je může povolit zpětně v průběhu užívání aplikace.

1.2.2 Service

Tato komponenta provádí dlouhodobě trvající operace na pozadí bez zásahu uživatele, spouští se na pokyn vlastní či jiné aplikace a nemá žádné grafické ani jiné uživatelské rozhraní. Může zajišťovat například síťový provoz či přehrávání hudby. Správné fungování Service je vždy zásadní pro plynulý chod telefonu a životnost baterie, protože tuto komponentu většinou uživatel nemůže nijak ovlivnit a je závislý na dobré optimalizaci aplikace.

1.2.3 Activity



Obr. 1.3: Blokové schéma Activity popisující její vznik a zánik [7]

Activity v aplikaci představují GUI (grafické uživatelské prostředí) a obsahují standartně jednu stránku aplikace (aplikace má většinou více Activity, které jsou mezi sebou provázány a předávají si informace). Activity je právě to, co uživatel vidí a s čím pracuje, proto je důležité dodržovat různé standarty v rozmístění a vzhledu ovládacích prvků. Google se toto snaží udávat svým vlastním grafickým návrhem. [8]

1.2.4 Android manifest

Android Manifest je XML soubor, který musí obsahovat každá aplikace. Soubor sděluje systému základní informace o aplikaci jako jsou název, verze, jednotlivé komponenty, požadovaná systémová práva a další požadavky pro samotný chod aplikace.

1.3 Android SDK, API, AVD

1.3.1 SDK

Android software development kit neboli softwarové vývojové prostředí je nástroj pro vývoj aplikací na platformu android. V mém případě se jedná o Android studio, které bylo od začátku navrhováno přímo pro vývoj Android aplikací (na rozdíl od jiných vývojových platforem, například Eclipse), díky tomu je vývoj aplikací v Android studiu snadný a intuitivní. Android studio, protože je to oficiální vývojářský nástroj, obsahuje velké množství návodů usnadňující programování a také má zabudované různé nástroje, které se používají při vývoji jako je GUI editor, SDK manager (pomocí SDK manageru lze stahovat systém Android do PC pro emulování v AVD, umožňuje stahovat různé ovladače a knihovny, které se pak dále dají použít při vývoji), AVD manager, …

1.3.2 API

Application programming interface označuje rozhraní a určuje jaké nástroje je možné použít pro programování aplikací. Čím vyšší je verze systému, tím větší je API úroveň a tím více modernějších nástrojů při vývoji lze použít. Ne všechny jsou bohužel zpětně kompatibilní tzn. Některé speciální funkce vyvinuté pro android 7 (API 24) nebudou fungovat například na androidu 4 (API 14). viz tabulka 1.1.

Verze systému	název	Verze API
4.0	IceCreamSandwich	14
5.0	Lollipop	21
7.0	Nougat	24
8.1	Oreo	27
9.0	Pie	28

Tab. 1.1: Příklad názvů a verzí systému

1.3.3 AVD

Android Virtual Device je nástroj pro simulování Androidu v počítačovém systému pro vývoj aplikací. AVD supluje fyzické zařízení s OS Android. Nástroj umožňuje plnohodnotnou softvérovou konfiguraci Android zařízení, a to nejen telefonů, ale také tabletů, televizí nebo wearables (nositelná elektronika). Uživatel si zde může nastavit rozlišení displeje, velikost vyhrazené virtuální paměti, architekturu systému (ABI), která závisí na možnostech hardwarové akcelerace a určuje interakci počítačového procesoru

s požadovaným výpočetním výkonem, nebo požadovanou verzi systému a s tím související verzi API. [9]

2 TEORETICKÁ ČÁST: HUDEBNÍ POJMY

V této kapitole se budu věnovat hudebním pojmům od vytvoření tónu oscilátorem až po základní druhy syntézy. V této práci se nebudu zabývat inharmonickými nebo šumovými signály, protože můj syntetizér bude generovat jenom harmonická spektra signálů která popisuji v kapitole o oscilátorech. Základní pojmy z oblasti akustiky vysvětlovat nebudu.

2.1 Oscilátor

Zvuk vzniká díky kmitavému pohybu hmoty. Tento kmitavý pohyb získáme díky oscilátoru, který generuje periodický signál. Tento signál v konečném výsledku a za správných podmínek dokáže lidské ucho identifikovat jako zvuk.

U klasických hudebních nástrojů je oscilátor například struna nebo plátek, u analogových elektronických nástrojů je oscilátor tvořen LC obvody, u elektronických digitálních nástrojů je naprogramovaný virtuální oscilátor. Oscilátor může mít dvě funkce.

2.1.1 Oscilátor jako řídící prvek

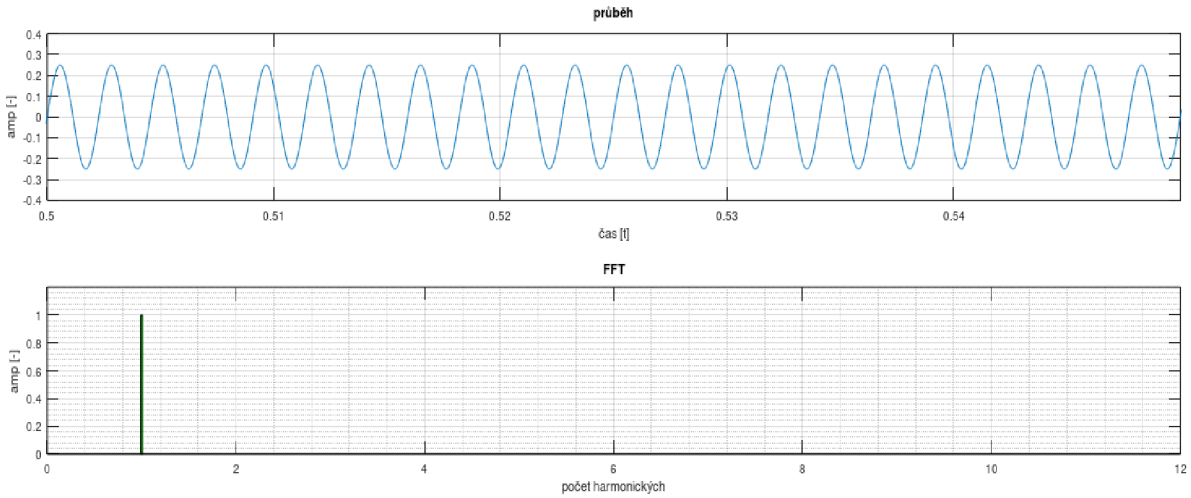
LFO – Nízkofrekvenční oscilátor v mém případě bude generovat sinusový signál v rozmezí 0,1 – 30 Hz. Jeho účel je periodicky ovlivňovat jiný signál, používá se například na tvorbu vibrata. Jeho základní parametry jsou míra modulace, frekvence a tvar vlny.

2.1.2 Oscilátor jako generátor zvuku

Oscilátor generuje harmonické kmity ve slyšitelném spektru. Můj digitálně řízený oscilátor (DCO) bude generovat 3 základní průběhy:

a) Sinus

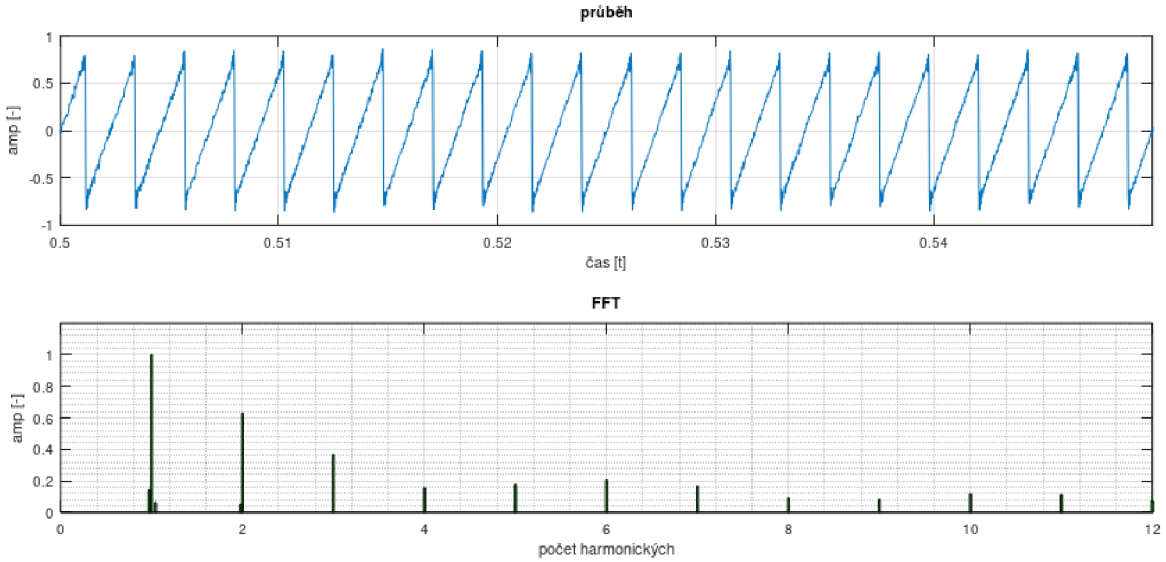
Základní harmonický signál obsahující jednu vlnu sinusového tvaru, má taky jen jednu harmonickou ve FFT spektru. Vlna sinus je základní stavební signál při tvorbě zvuku



Obr. 2.1: Sinus a jeho FFT spektrum

b) Pila

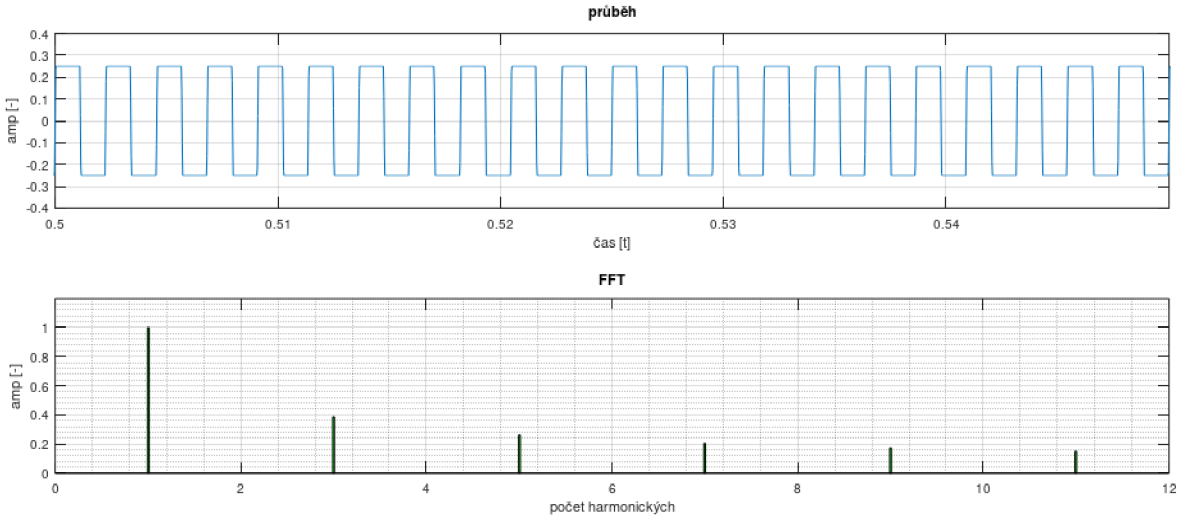
Pila obsahuje sudé i liché harmonické a je složena z teoreticky nekonečného množství sinusových průběhů, jejichž amplitudy klesají úměrně s jejich pořadovým číslem. Signál je harmonicky bohatý.



Obr. 2.2: Pila a její FFT spektrum

c) Čtverec

Čtverec obsahuje liché harmonické a je složen z teoreticky nekonečného množství sinusových průběhů. Pokud je jeho střída jiná, než 1:1, objevují se v jeho FFT spektru i sudé harmonické složky. [10]



Obr. 2.3: Čtverec a jeho FFT spektrum

2.2 Syntéza

Syntéza má za úkol napodobit již existující zvuk, například zvuk hudebního nástroje, nebo vytvářet zvuky nové, které nemohou být vytvořeny na (elektro)akustické nástroje. Syntéza tvoří určitý celek z jednotlivých složek signálu. Existuje v podstatě nekonečné množství způsobů, jak vytvořit výsledný signál – zvuk. Syntézu dělíme na několik metod, například:

- aditivní
- modulační
- subtraktivní
- tvarové
- granulární

Ve své aplikaci využívám syntézu aditivní (skládání souzvuků nebo akordů), frekvenčně modulační (Low Frequency Oscilator) a částečně subtraktivní (použití ekvalizéru). [11] [12] [13]

2.2.1 Aditivní syntéza

Aditivní neboli součtová syntéza umožňuje sčítat různé jednoduché průběhy (nejčastěji sinus). Podle Fourierova rozvoje víme, že v podstatě jakýkoliv periodický signál lze rozložit na určitý počet sinusových průběhů o různých frekvencích, amplitudách nebo vzájemných fázových posunech. Proto je sinus základní stavební kámen tohoto druhu syntézy. V praxi se nejčastěji používá na přidávání vyšších harmonických složek k základní frekvenci. Dají se takhle dokonce tvořit za určitých podmínek i další periodické signály jako pila, čtverec, …

Nejznámějším elektrickým hudebním nástrojem využívající aditivní syntézu jsou Hammondovy varhany.

2.2.2 Modulační syntéza

Základní modulační syntézy můžeme dělit na frekvenční, amplitudovou, kruhovou a parametrickou.

Amplitudová modulace mění amplitudu nosného signálu (na rozdíl od frekvence a fáze). Ve frekvenčním spektru mohou být dvě postranní pásma, nebo pouze jedno – horní nebo dolní. V rámci modulace hlavně v elektrotechnice mluvíme o plně modulovaném, podmodulovaném a přemodulovaném signálu.

Kruhová modulace vzniká opět násobením dvou harmonických signálů za vzniku součtové a rozdílové složky. Rozdíl je v tom, že je zde potlačen hlavní nosný signál. Vzniká plechový zvuk připomínající například elektrické výboje nebo cvrkot.

V mé aplikaci pracuji s jednoduchou variantou frekvenčně modulační syntézy. Tato syntéza se skládá z hlavního nosného a modulačního signálu. Zde zůstává amplituda stále stejná, ale přidávají se další frekvence – k hlavní a modulační frekvenci vznikají postranní, neharmonická pásma, které zásadně ovlivňují barvu zvuku. V této aplikaci využívám FM syntézu k modulování hlavní vlny nízkofrekvenčním oscilátorem (LFO). V praxi se LFO v rámci FM syntézy využívá například k vytvoření vibrata.

2.2.3 Subtraktivní syntéza

Subtraktivní neboli rozdílová syntéza je komerčně nejrozšířenější a nejpoužívanější. Principem fungování je opakem aditivní syntézy – z harmonicky bohatých signálů (sinus je tedy nepoužitelný) část spektra ubírá. Nevznikají tedy žádné nové frekvenční složky, pouze ty stávající jsou buďto potlačeny, nebo zesíleny. Ideální zdroj je tedy širokospektrální šum. Díky této syntéze mohou vznikat i všechny ostatní průběhy popsané v kapitole *2.1.2 Oscilátor jako generátor zvuku*.

2.2.4 Tvarová syntéza

Tvarová syntéza zastupuje několik technik na přetvarování jednoduchého vstupního signálu. Jedna z možností je například (náhodné) zamíchání vzorků v diskrétní podobě, nebo průchod vlny nelineárním obvodem, kde mohou vznikat nové harmonické složky. Další možnost je například graficky přímo měnit tvar vlny.

2.2.5 Granulární syntéza

Tento typ syntézy umí rozřezat vlnu na velice krátké kousky (jednotky až desítky milisekund) a pak je znova poskládat za sebe v různém pořadí i v různých vrstvách. Lze tedy vytvářet nové zvuky, nebo měnit barvu, výšku a rychlost daného zvuku ze začátku.

2.3 Soustava ladění

V této aplikaci se bude dát pracovat s tóny jak v jejich číselné hodnotě, vyjádřené v Hz, tak i pomocí názvu tónů v temperovaném ladění například pro nastavení akordů a jiných souzvuků (viz kapitola *4.8 Sestavování akordů a jiných souzvuků*).

Rovnoměrně temperované ladění se ladí podle referenčního tónu *komorní* a^1 o frekvenci 440 Hz dle dohody ISO v roce 1953. Toto ladění je názorně vidět například u klavíru, kde jedna oktáva je rozdělena na 12 stejných dílků. Díky tomu máme v temperovaném ladění enharmonické záměny, což v jiném ladění, například pythagorejském, není pravda.

Jednotlivé půltóny mají od základního tónu vzdálenost

$$v = \sqrt[12]{2^x} \tag{3.1}$$

kde x je požadovaná vzdálenost od základního tónu.

1. Příklad: čistá kvinta od tónu a^1 je e^2, tedy

$$v = \sqrt[12]{2^7} \tag{3.2}$$
$$v = 1{,}49830 \text{ Hz}$$

Výpočet frekvence tónu e^2 tedy bude 440 · 1,4983 = 659,2551 Hz

	c	**cis/des**	**d**	**dis/es**	**e**	**f**	**fis/ges**	**g**	**gis/as**	**a**	**ais/b**	**h**
Sub	16,35	17,32	18,35	19,44	20,60	21,82	23,12	24,50	25,95	27,50	29,13	30,86
kontra	32,70	34,65	36,71	38,89	41,20	43,65	46,24	48,99	51,91	55,00	58,26	61,73
velká	65,4	69,3	73,4	77,8	82,4	87,3	92,5	98,0	103,8	110,0	116,5	123,5
malá	131	139	147	156	165	175	185	196	208	220	233	247
Jedno	262	277	294	311	330	349	370	392	415	440	466	494
Dvou	523	554	587	622	659	698	740	784	831	880	932	988
Tří	1046	1109	1175	1244	1318	1397	1480	1568	1661	1760	1864	1975
Čtyř	2093	2217	2349	2489	2637	2793	2960	3136	3322	3520	3729	3951
Pěti	4186	4435	4698	4978	5273	5587	5919	6271	6644	7040	7458	7901
Šesti	8371	8869	9397	9955	10547	11174	11839	12543	13288	14080	14916	15802

Tab. 2.1: Přehled frekvencí jednotlivých tónů

[14] [15] [16]

Tónový rozsah mé aplikace bude od C do c^3 tedy 65,4 Hz až 1046 Hz podle rovnoměrného temperovaného ladění a^1 = 440 Hz. Viz kapitola *3.4 Zvukové možnosti a srovnání aplikace s jinými podobně řízenými aplikacemi*

3 NÁVRH APLIKACE

Tato kapitola obsahuje návrh aplikace pomocí stručného popisu ovládání a návrhu uživatelského rozhraní v Android Studiu za použití základních ovládacích i grafických prvků. Závěr kapitoly je věnovaný srovnáním s jinými podobně fungujícími aplikacemi.

Aplikace umožňuje generovat jednoduché periodické signály (k výběru jsou sinus, pila, čtverec), které lze dále vzájemně skládat, případně jinak upravovat a filtrovat. Uživatel zde může generovat zároveň hlavní signál a zároveň jeden z pěti uživatelsky přednastavených souzvuků. Generátor hlavního signálu bude měnit svojí frekvenci na základě pohybu zařízení (natočení vlevo a vpravo), bude ho možné ekvalizovat a případně modulovat LFO. Generátory akordů a jiných souzvuků si uživatel může nastavit před spuštěním hlavního okna. Zde uživatel nastavuje jak tvar vlny, tak tóny případně frekvence, ze kterých se souzvuky skládají. Aplikace má ve výchozím stavu 2 tlačítka pro vstup do nastavení generátorů a souzvuků, tlačítko nápovědy, kde jsou stručně popsané možnosti aplikace a jejího nastavení a tlačítko pro vstup do Hlavní Activity aplikace.

3.1 Android Studio

Při návrhu aplikace bylo použito Android Studio a možností jeho grafického editoru.

Při vytvoření Activity se vytvoří .java a .xml soubor a název Activity se automaticky zapíše do *AndroidManifest.xml* souboru.

.xml soubor obsahuje veškeré grafické i neviditelné ovládací prvky aplikace. Většina parametrů, jako je velikost, barva, umístění nebo název tlačítka či jiného ovladače, se dá nastavit z grafického editoru, pro pokročilejší možnosti nastavení se musí do textového editoru.

.java soubor dává všem grafickým prvkům nastaveným v .xml souboru funkci. Pro posunutí na další aktivity pomocí tlačítka, prvky Spinneru či funkce switche se nastavují právě zde. Nastavení probíhá v programovacím jazyce java a je částečně zjednodušeno automatickým doplněním slov, frází a funkcí.

3.2 Ovládání aplikace

Samotný pohyb v aplikaci, tedy její nastavení, bude stejný jako pohyb v jakékoliv mobilní aplikaci vyvíjené pro android. Základ jsou zde ovládací prvky v zobrazené Activity, tedy virtuální tlačítka na displeji. Tyto tlačítka mohou uživatele posunou o jednu Activity vpřed podle popisu tlačítek. pro pohyb o jednu Activity zpět se používá navigační lišta (tlačítko zpět, menu a domů) na displeji nebo hardwarová tlačítka jako součást zařízení. Dále se zde uživatel může setkat se SeekBarem neboli virtuálním posuvným potenciometrem. Při výběru tvaru signálu nebo výšky tónu se uživatel setká s ovládacím prvkem Spinner neboli vysouvací seznam. Aplikace si pamatuje naposledy zvolenou položku a někdy

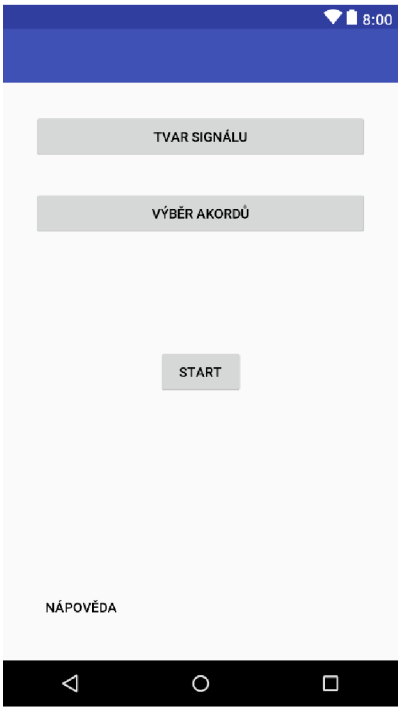
jí potvrdí toast zprávou. Z dalších ovládacích prvků ještě můžu zmínit textové pole pro zadávání číselné hodnoty frekvence a Switch neboli přepínač.

Za běhu Hlavní Activity aplikace se zařízení drží horizontálně displejem k sobě. Uživatel na hlavní obrazovce uvidí několik tlačítek, které ho neposunou o Activity dopředu, ale vyvolají akci v zobrazovaném okně, tedy většinou přehrají přednastavený zvuk. Frekvence melodie se ovládá nakloněním zařízení v ose Y, naklonění bude limitováno postavením telefonu to vertikální polohy.

3.3 Grafické uživatelské rozhraní

Aplikace je navrhována pro referenční zařízení Pixel 2, tedy pro 5“ obrazovku s Full HD rozlišením (1920x1080 pixelů, 420dpi). Podle mého se jedná o kompromis mezi velikostí ovládacích prvků, pohodlností ovládání a mírou zastoupení mezi uživateli. Aplikace byla testována na 4,7“ HD rozlišení (1280x720 pixelů, 312dpi), tedy i menší nebo starší zařízení by s ovládáním aplikace neměly mít problém.

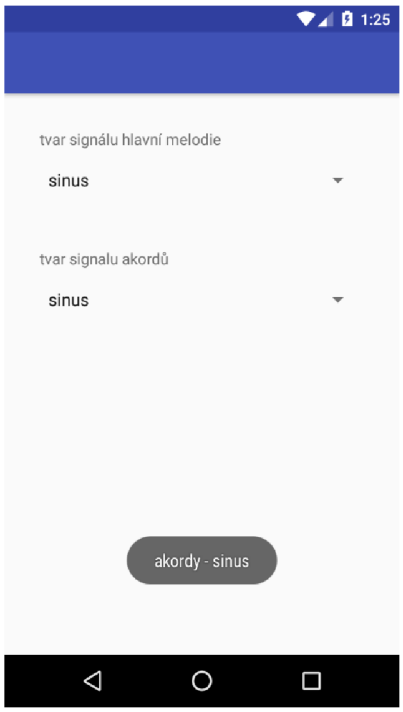
Výchozí obrazovka neboli Menu obsahuje 4 tlačítka. Každé tlačítko posune uživatele o jednu Activity vpřed. První dvě tlačítka obsahují prvky nastavení, které jsou po spuštění Hlavní Activity aplikace neměnné. První tlačítko obsahuje možnosti nastavení pro samotné generátory aplikace, druhé umožňuje nastavení tónů k akordům a jiným souzvukům. Třetí tlačítko spouští Hlavní Activity aplikace. Tlačítko ve spodním rohu displeje vyvolává obrazovku s nápovědou, kde bude stručný popis aplikace a funkcí. Při návrhu aplikace počítám se softwarovou navigační lištou.



Obr. 3.1: Návrh Activity Menu

3.3.1 Tvar signálu

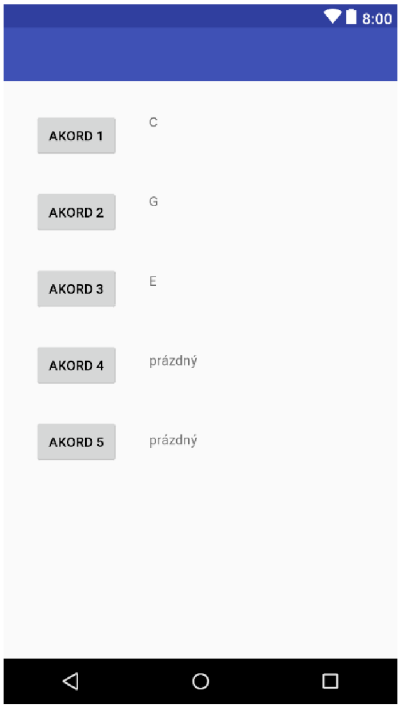
V této Activity uživatel vybere pomocí dvou Spinnerů výchozí tvar signálu. Toto je pokyn pro generátor, který následně podle dat uložených ve Spinneru bude generovat dané harmonické průběhy. Jako potvrzení se zobrazí Toast zpráva. V průběhu vývoje přibyla funkce na nastavení počtu tónů vložených mezi dva sousedící půltóny u hlavního signálu. Při nejvyšší možné volbě (6 dalších vložených tónů) se hlavní signál mění na základě dat ze senzoru téměř plynule. Zde už narážím na technické možnosti senzoru (rychlost čtení dat ze senzoru), viz *4.10.1 Generátor hlavní melodie*



Obr. 3.2: Návrh Activity TvarSignalu

3.3.2 Výběr akordů

Activity *VyberAkordu* zobrazuje pět tlačítek, které posouvají uživatele o jednu Activity odpředu. Vedle tlačítek je textové pole, které zobrazuje vybraný akord nebo jiný souzvuk, který uživatel nastaví v následující Activity.



Obr. 3.3: Návrh Activity VyberAkordu

3.3.3 Akord 1

Activity *Akord1*, stejně jako všechny Activity v této rovině, dává uživateli možnost výběru akordu či jiné soustavy tónů, které se pak použijí v Hlavní Activity aplikace. Uživatel zde má na výběr buď přednastavený durový akord, nebo použije následující spinnery pro sestavení vlastního seskupení tónů. Dále během vývoje přibyla funkce na sestavování vlastních akordů pomocí uživatelsky zadané frekvence jednotlivých tónů. Generátor bude generovat jednotlivé tóny akordu zvlášť.

Obr. 3.4: Návrh Activity Akord1

3.3.4 Hlavní Activity aplikace

Activity je navrhována jako FullScreenActivity, tedy Activity přes celou obrazovku, aby bylo možné použít ušetřené místo, které zabírá navigační lišta a aby nedocházelo k nechtěným akcím vyvolané právě kliknutím na navigační nebo notifikační lištu. Od tohoto záměru jsem později upustil kvůli pohodlnějšímu ovládání aplikace. Zároveň je počítáno s držením zařízení horizontálně – horizontální rozložení této Activity je zde vynuceno.

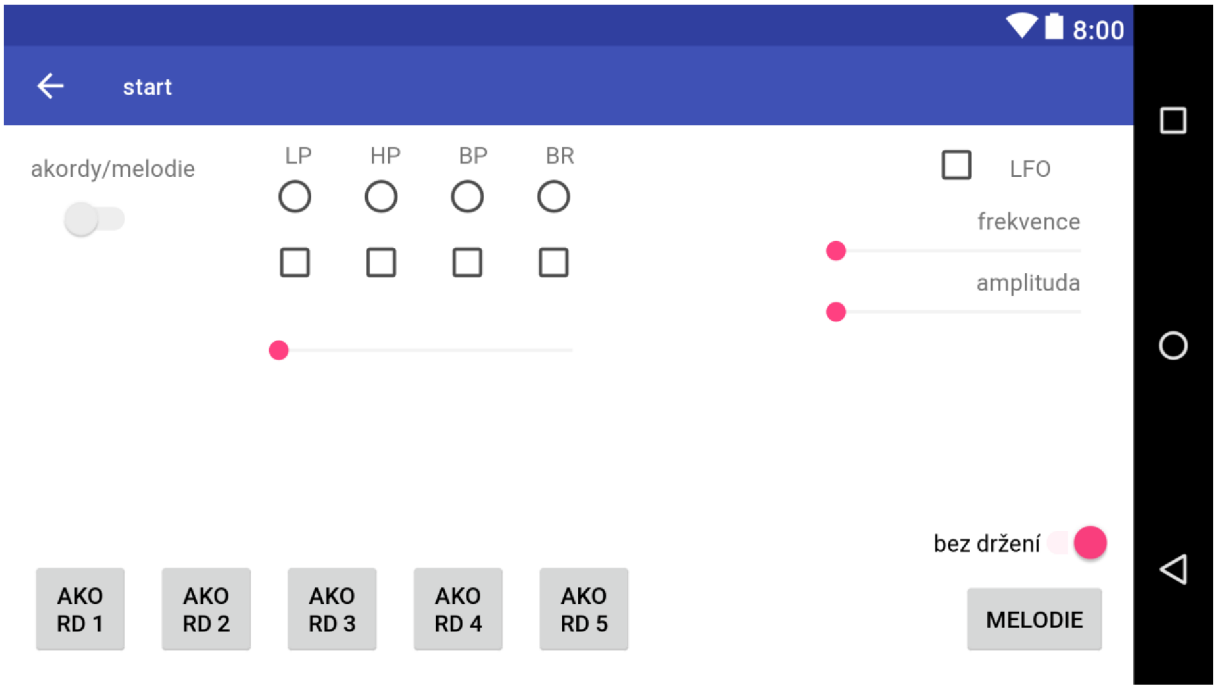
Základ této Activity je 6 tlačítek umístěných ve spodní části. Tlačítka po dobu držení generují zvuk. Tlačítka Akord 1 až Akord 5 generují signál zvolený při vybírání akordů. Tlačítko melodie obsahuje ještě Switch nad tlačítkem, který umožňuje přepínání mezi generováním signálu nepřetržitě (bez držení tlačítka) ve své zapnuté poloze, nebo bude hlavní signál generovaný pouze po dobu držení tlačítka Melodie.

Asi uprostřed obrazovky je nastavení filtrů. Pod textovými poli je RadioButton které slouží pro výběr jednoho filtru. Na Vybraný filtr bude uplatňovaný SeekBar neboli nastavení mezní frekvence. Mezi SeekBar a RadioButton je CheckButton, které slouží pro aktivování daného filtru. Filtr může mít nastavenou hodnotu, ale vypnutím CheckButton se filtr vypne. CheckButtonů může být samozřejmě zapnutých více najednou.

Vpravo nahoře je CheckButton pro aktivování LFO. Dva SeekBary pod ním potom nastavují parametry LFO podle popisku. Frekvence LFO bude pracovat od 0,1 do 40 Hz, protože je to podle mě rozpětí, které dokáže prezentovat všechny podstatné možnosti tohoto generátoru. SeekBar Amplituda nastavuje úroveň modulace hlavního signálu.

Vlevo nahoře je switch, který po zapnutí umožňuje aplikovat filtry a LFO na hlavní signál, pokud je Switch vypnutý, filtry a LFO ovlivňují signál akordů.

Během vývoje se několik ovládacích prvků změnilo, byly odebrány, nebo naopak vznikly nové. Tuto změnu popisuji v kapitole *4.1 Změny v aplikaci oproti návrhu.*



Obr. 3.5: Návrh Hlavní Activity aplikace

3.4 Zvukové možnosti a srovnání aplikace s jinými podobně řízenými aplikacemi

Díky filtrům jsou možnosti nastavení zvuku poměrně bohaté s ohledem na jejich jednoduché nastavení. Ovládání filtrů bylo navrhováno záměrně pouze s použitím jednoho SeekBaru ze dvou důvodů: aby mohl aplikaci pohodlně ovládat i laik bez předchozích zkušeností a pro demonstrativní účely funkčnosti, protože na mobilním zařízení nikdy nebude uživatel nastavovat parametry zvuku a filtrů tak podrobně jako u počítačových plug-inů/syntezátorů například z důvodu volby výstupů (reprodukce zvuku), nebo použití pro další práci s tímto zvukem. To samé platí i pro LFO, který umožňuje jen jednoduché nastavení frekvence a amplitudy, opět bez možnosti podrobného zadávání přesných hodnot, protože by to bylo v mobilní aplikaci tohohle typu kontraproduktivní na úkor ovladatelnosti.

S ovládáním zvuku pomocí pohybu zařízení musím dalším testováním přijít na správný způsob implementace a nastavení citlivosti. Gyroskopický senzor v každém moderním zařízení je poměrně citlivý i na nepatrné pohyby způsobené například třesem ruky, a právě toto se pokusím správným nastavením citlivostí odfiltrovat. Detekce pohybu musí být zároveň dostatečně citlivá, aby byl uživatel schopný pohodlným pohybem zařízení obsáhnout celé spektrum nastavené frekvence. Právě z tohoto důvodu bude vrchní frekvence omezená na hodnotu 1046 Hz neboli c^3. Spodní frekvence bude začínat na 65,4 Hz, tedy velké C. Vycházím z konstrukčních důvodů reproduktorů v mobilním telefonu.

Výpočet naklonění vychází z hodnot zařízení v service menu[2], kde, při poloze telefonu horizontálně, je hodnota osy Y 0. Nakloněním o 90° vpravo, tedy do standardní vertikální polohy, se dostaneme na hodnotu osy Y 10, nakloněním o 90° vlevo, tedy vertikálně vzhůru nohama, dostáváme hodnotu −10.

Mezi tóny C a c^3 jsou 4 oktávy neboli 4·12 půltónů. Při startu, kdy bude zařízení v horizontální poloze, bude hodnota Y = 0, tedy c^1 261,6 Hz. Na každou stranu otočení zbyde 24 půltónů. Pro hodnotu půltónu na ose Y vycházíme ze vztahu (4.1)

$$\frac{10}{24} \cong 0{,}4167 \tag{4.1}$$

Pokud tedy budeme chtít vypočítat například hodnotu tónu a^1 440 Hz, zjistíme, že a^1 je 10. půltón nad středním tónem c^1 a počítáme

$$10 \cdot \frac{10}{24} \cong 4{,}167 \tag{4.2}$$

$$\frac{4{,}167}{10} \cdot 90 \cong 37{,}503 \tag{4.3}$$

A pomocí trojčlenky (4.3) zjistíme, že je potřeba zařízení naklonit o 37,5° doprava. Hodnota osy Y je v tomto případě vyjádřena rovnicí (4.2)

3.4.1 Aplikace pro Android

Aplikace se vyznačují hlavně zjednodušeným ovládáním přizpůsobeným pro dotykový displej při zachování dostatečného množství ovládacích prvků.

a) DRC – Polyphonic Synthesizer [17]

Jako zástupce syntetizérů pro Android jsem si vybral aplikaci s názvem DRC – Polyphonic Syntesizer. Aplikace má jednoduché uživatelské rozhraní, přitom nabízí opravdu hodně možností nastavování. Aplikace vychází ze dvou oscilátorů, které doplňují šumový oscilátor a filtry jako je LFO nebo obálka ADSR. Aplikace také umí pracovat s připojenými midi zařízeními a stereo funkcemi. Na této aplikace se mi líbí jednoduché, a přitom propracované uživatelské prostředí schovávající velké množství funkcí.

b) Music Synthesizer for Android [18]

Tato jednoduchá aplikace byla vytvořená pro demonstraci „High Performance Audio for Android“ na Google I/O 2013 jako příklad toho, jak lze dosáhnout nízké odezvy pro kvalitní přehrávání zvuku. Aplikace emuluje zvuky z legendární Yamahy DX7. Aplikace rovněž podporuje USB midi zařízení a obsahuje resonanční filtr s nastavitelným mezním kmitočtem a úrovní rezonance. Třetí ovládací prvek je nastavení úrovně overdrive. S Těmito třemi ovládacími prvky v kombinaci se zvuky z DX7 jdou zahrát velice zajímavé zvuky.

[2] Service menu umožňuje testovat tlačítka, displej nebo senzory v zařízení i s výpisem jejich hodnot.

3.4.2 Aplikace pro PC

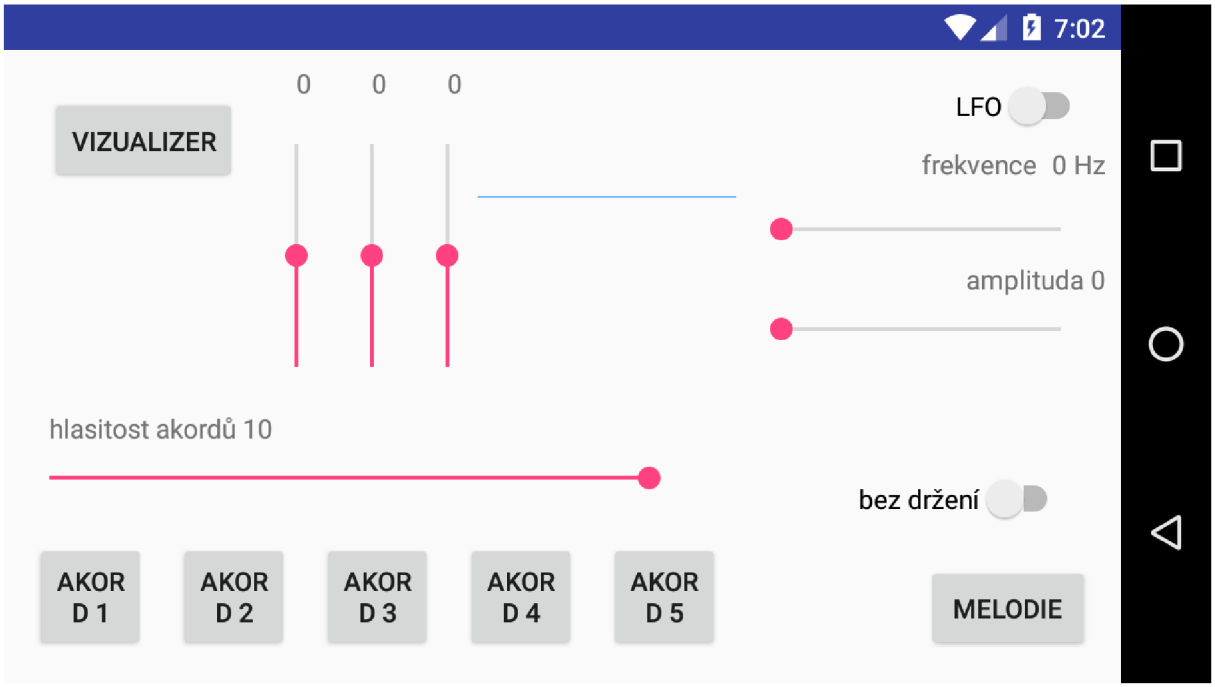
a) MiniMogueVA [19]

Jako zástupce aplikace pro PC zmíním MiniMogueVA jako softwarový syntezátor založený na analogovým nástroji Minimoog. Tato aplikace obsahuje 3 oscilátory a všechny základní filtry a jejich pokročilé nastavení. MiniMoog je neznámnější analogový syntezátor a velice významný nástroj pro popovou a rockovou hudbu 70. let.

4 REALIZACE APLIKACE

4.1 Změny v aplikaci oproti návrhu

Oproti návrhu GUI, který popisuji v kapitole *3.3* prošlo největší změnou *Hlavní Activity aplikace.* Zde přibyl SeekBar pro ovládání hlasitosti akordů (hlasitost hlavní melodie je nastavována systémovou hlasitostí multimédií), přibylo tlačítko vizualizéru, kterým se uděluje oprávnění pro užití mikrofonu právě kvůli nově přidanému vizualizéru. Toto dále popisuji v kapitole *4.7 Vizualizér.* Dále byl přepracován systém ekvalizéru, který nyní funguje pro zjednodušení jako klasický 3 pásmový ekvalizér se třemi SeekBary *low, mid, high*, viz *4.6 Ekvalizér.* Drobnou, hlavně grafickou, změnou prošly i prvky pro nastavení LFO, který se nyní zapíná Switchem a uživatel může nad SeekBarem kontrolovat zadanou číselnou hodnotu. Aktuální číselnou hodnotu frekvence hlavní melodie nyní vidí uživatel i na tlačítku spouštění hlavní melodie.



Obr. 4.1: Nové rozložení hlavní Activity aplikace.

Další změnou prošla Activity pro sestavení akordů, kde si nyní může uživatel vybrat mezi přednastaveným durovým akordem, může si sestavit libovolný jiný akord z jednotlivých přednastavených tónů ve třech oktávách, nebo může zadat ručně frekvenci jednotlivých tónů, viz *4.8 Sestavování akordů.*

Poslední hlavní změna je přidání funkce pro nastavení poměru rychlosti změny frekvence u hlavní melodie, kterou popisuji v kapitole *3.3.1 Tvar signálu.*

4.2 Ukládání dat

V aplikaci je žádoucí, aby si pamatovala některá nastavení, která uživatel provede. Tato nastavení zůstanou v paměti i poté co uživatel aplikaci zavře. Po znovuotevření aplikace

se hodnoty načtou z paměti a jednotlivé ovládací prvky se tomu přizpůsobí. Pro vymazání těchto dat a uvedení aplikace do továrního nastavení je nutné, aby uživatel vymazal data aplikace v nastavení systému.

Vzhledem k tomu, že potřebuji ukládat pouze primitivní data typu int, boolean nebo string, vybral jsem si funkci Shared Preferences. Tato funkce umožňuje ukládat právě tento jednoduchý typ dat pomocí klíčového slova a pak k nim kdekoliv jinde v aplikaci zase přistoupit.

Do paměti jsou mimo jiné uložené hodnoty ze všech možností nastavení aplikace z Menu, aby je uživatel nemusel při každém zapnutí znovu zadávat.

4.3 Hlavní Activity aplikace

V tomto okně a v tomto .java souboru se schází veškerá data, která aplikace umí vygenerovat, byla uložena do paměti nebo vytvořena při vývoji.

Hned v metodě *onCreate* jsem definoval implementaci akcelerometru a jeho listeneru, dále je zde definováno čtení z paměti SharedPreferences a inicializace vizualizéru za podmínky, že má aplikace získané systémové oprávnění. Samozřejmě se zde volají všechny grafické ovládací prvky aplikace a metody, které k nim náleží.

Naproti metodě *onCreate* jsou metody *onPause, onStop, onDestroy* (viz. *1.2.3 Activity*), kde je důležité čistit paměť a zastavovat běžící procesy. V rámci optimalizace jsem se rozhodl ve všech těchto metodách vypouštět listener pro akcelerometr (kdyby dále běžel na pozadí, mohlo by to mít vliv například na výdrž baterie zařízení, pokud by ho systém včas sám nezastavil), samozřejmě zastavovat generátory hlavní melodie, kdyby uživatel opustil Hlavní Activity bez zastavení (zpravidla Switch pro generování bez držení tlačítka) a v neposlední řadě odinicializovat ekvalizér a vizualizér.

V samotném těle této Classy jsou nejdůležitější prvky, které ovládají samotné generátory. Tlačítka na spouštění souzvuků jsou vypodmínkované tak, aby nebylo možné spustit 2 souzvuky najednou, stejně tak tlačítko a Switch hlavní melodie. U ovladačů hlavní melodie jsem byl nucen nastavit zpoždění spouštění generátoru kvůli předávání *AudioSessionID*. Aktuální zpoždění mám 20ms (*Thread.sleep(20*)) a dle testování je to jedno z nejnižších možných. Toto zpoždění čeká na generátor (*AudioTrack*), který mezitím vygeneruje svoje unikátní *AudioSessionID* a předá ho do Hlavní Activity aplikace pro správnou implementaci ekvalizéru.

4.4 Čtení dat ze senzoru

Akcelerometr je v Android Javě zpracováván SenzorManagerem [20], který vytvoří Listener. Tento Listener umožňuje nastavit jaký typ dat a jak rychle bude aplikace ze senzoru zpracovávat. Pro potřebu téhle aplikace jsem Listener nastavil na nejrychlejší možnou obnovovací frekvenci a největší přesnost.

```
SensorManager.SENSOR_DELAY_FASTEST,
SensorManager.SENSOR_STATUS_ACCURACY_HIGH
```

Výpis kódu 4.1: Nastavení rychlosti čtení dat ze senzoru

Tento SenzorEventListener poté vytvoří 2 metody *onSensorChanged* a *onAccuracyChanged*. Vzhledem k tomu, že přesnost senzoru v průběhu neměním, přijímám data pouze v metodě *onSensorChanged.* Tato metoda se volá s každou změnou na senzoru v rychlosti, jaký jsem si nastavil výše, je to proto ideální metoda na výpočet frekvence hlavní melodie. Výpočet provádím podle rovnice 3.1 s tím, že telefon v horizontální poloze (osa y = 0) má 440 Hz. Na každou stranu jde poté 16 tónů (185 Hz fis až 1047 Hz c´´´). Výstup z této metody je poté int s frekvencí, kterou předávám generátoru hlavního signálu. Kód výpočtu frekvence hlavní melodie včetně nastavení poměru změny frekvence uvádím v příloze A, část 3. Výpočet frekvence hlavní melodie.

4.5 Generátory signálů

V Aplikaci mám 2 druhy generátoru signálu. Oba druhy generuji pomocí Android knihovny AudioTrack. Oba generátory vychází ze stejné rovnice [21]. Generátory se liší hlavně metodou ukládání dat do mezipaměti.

4.5.1 Generátor akordů

První druh signálu používám pro generování v čase frekvenčně neměnných tónů – pro akordy. Tento AudioTrack je nastaven v MODE_STATIC což znamená, že AudioTrack počká, až se naplní buffer vzorky a poté přehraje celý buffer. Ten následně opakuji po celou dobu držení tlačítka. Používám standartní vzorkovací frekvenci 44100 Hz, tedy pro výpočet vzorků v jedné vlně (kterou opakuji) použiju vzorec (4.1)

$$sc = \frac{sr}{f} \tag{4.1}$$

Kdy sc je počet vzorků, sr je samplovací frekvence, f je frekvence tónu.

Celý generátor poté vypadá takto:

```
private final int SAMPLE_RATE = 44100;
private AudioTrack audioTrack1;
private int sampleCount;
public int amplitude = 32767; //max int hodnota

public void setSinus1(int frekvence) {
    int buffsize = AudioTrack.getMinBufferSize(SAMPLE_RATE,
AudioFormat.CHANNEL_OUT_MONO, AudioFormat.ENCODING_PCM_16BIT);
    audioTrack1 = new AudioTrack(AudioManager.STREAM_MUSIC, SAMPLE_RATE,
AudioFormat.CHANNEL_OUT_MONO, AudioFormat.ENCODING_PCM_16BIT, buffsize,
AudioTrack.MODE_STATIC);
    if (audioTrack1.getPlayState() == 1) {
        sampleCount = (int) ((float) SAMPLE_RATE / frekvence);
        short[] sample = new short[sampleCount];

        double TwoPi = 8. * Math.atan(1.);
        double phase = 0.0;

        for (int i = 0; i < sampleCount; i++) {
            sample[i] = (short) (AMPLITUDE * Math.sin(phase));
            phase += TwoPi * frekvence / SAMPLE_RATE;
        }
        audioTrack1.write(sample, 0, sampleCount);
    }
}
public void startvoid1() {
    if (audioTrack1.getPlayState() == 1) { //Jestli je zastaven
        audioTrack1.reloadStaticData();
        audioTrack1.setLoopPoints(0, sampleCount, -1); //nekonečné opakování
        audioTrack1.play();
    }
}
```

Výpis kódu 4.2: Generátor sinusového průběhu MODE_STATIC

Tento příklad z mého kódu je pro generování sinusového signálu prvního tónu akordů.

V tomto generátoru kromě frekvence dále měním amplitudu a tím řídím hlasitost akordů, který si uživatel může nastavit pomocí SeekBaru v hlavní Activity aplikace. SeekBar má pro rychlejší nastavení 10 kroků. Číslo 32767 vyšlo díky 16bitovému kódování a je to 0 dB pro integrovaný DA převodník.

4.5.2 Generátor hlavní melodie

Tento generátor běží jako další Thread (krom hlavního) a je nastaven v MODE_STREAM. Toto nastavení se hodí právě pro signály, jejichž frekvence není stálá – generátor přehrává vzorky ihned tak jak jsou vypočítány a nečeká na naplnění bufferu. Výpočet je téměř stejný jako v případě generátoru u akordů, ovšem je zde možnost ještě hlavní signál modulovat LFO. Všechny typy průběhů mohou být modulovány LFO ve tvaru sinus. Toto dále popisuji v kapitole *4.10.5 LFO.*

```
public void run() {

    super.run();
    isRunning = true;

    int sibuffsize = AudioTrack.getMinBufferSize(sr,
            AudioFormat.CHANNEL_OUT_MONO,
AudioFormat.ENCODING_PCM_16BIT);
    AudioTrack siaudioTrack = new AudioTrack(AudioManager.STREAM_MUSIC,
            sr, AudioFormat.CHANNEL_OUT_MONO,
            AudioFormat.ENCODING_PCM_16BIT, sibuffsize,
            AudioTrack.MODE_STREAM);

    short[] samples = new short[sibuffsize];
    int amp = 32767;
    double twopi = 8. * Math.atan(1.);
    double ph = 0.0;
    double ph2 = 0.0;
    double PrubehLFO;

    try {
        siaudioTrack.play();
    } catch (IllegalStateException e) {
        System.out.println("chyba play " + e);
    }
    ID = siaudioTrack.getAudioSessionId();
    while (isRunning) {
        double f = tuneFreqSI;
        double fLFO = frLFO;
        for (int i = 0; i < sibuffsize; i++) {
                PrubehLFO = ampLFO * Math.sin(ph2);
            samples[i] = (short) (amp * Math.sin(ph + PrubehLFO));
            ph += twopi * f / sr;
            ph2 += twopi * fLFO / sr;
        }
        siaudioTrack.write(samples, 0, sibuffsize);
    }
    siaudioTrack.stop();
    siaudioTrack.release();
}
```

Výpis kódu 4.3: Generátor sinusového průběhu MODE_STREAM

Takto vypadá generátor Hlavní melodie ve tvaru sinus. Tento Thread tedy přijímá jak frekvenci hlavní melodie na základě dat ze senzoru, tak frekvenci a úroveň modulace LFO. Thread jsem se snažil optimalizovat tak, aby měl nejmenší možnou latenci a velikost bufferu.

4.6 Ekvalizér

Ekvalizér realizuji za využití originální android knihovny [22]. Ve výchozím stavu má ekvalizér 5 pásem s pevným středem na 60 Hz, 230 Hz, 910 Hz, 3600 Hz a 14000 Hz. Pro potřeby této aplikace je zbytečné upravovat krajní pásma, tudíž jsem implementoval 3pásmový ekvalizér. Jednotlivá pásma může uživatel nastavit Seekbarem v rozsahu 30 dB (od -15 dB do +15 dB), což je maximální rozsah ekvalizéru z této knihovny při základním systémovým nastavení. Kód je přizpůsobený na variantu,

že výrobce telefonu zasáhne do knihovny equalizéru a změní její parametry (například rozsah ekvalizéru).

Ekvalizér umí v základním nastavení ekvalizace pomocí SeekBarů ekvalizovat zvuk pouze při jeho přehrávání. To znamená po dobu běhu generátoru se uplatňuje ekvalizace pouze pro tento generátor (na toto jedno AudioSessionID) a při dalším spuštění generátoru (a změně AudioSessionID) už se logicky vyvolá ekvalizér nový bez ohledu na nastavené parametry na SeekBaech. V metodě *afterInitEQ* se věnuji tomu, aby si uživatel mohl prvně nastavit ekvalizér (hodnoty na SeekBarech) a po spuštění generátoru se toto nastavení ekvalizéru projevilo. Tato metoda se uplatňuje zároveň pro znovuspuštění generátoru, kdy vyvolá ekvalizér s hodnotami podle nastavení SeekBaru, který se nemění až do dalšího zavolání metody *setupEQ,* tedy zavolání onCreate metody v Hlavní Activity aplikace. Viz příloha A, část 1. Ekvalizér.

4.7 Vizualizér

Vizualizér zobrazuje aktuální průběh vlny a jiných složených signálů, které aktuálně telefon generuje. Opět zde využívám originální Android knihovnu [23]. Pro správnou funkci potřebuje, bohužel, systémové oprávnění k přístupu k mikrofonu. Vizualizér k mikrofonu fakticky nepřistupuje, ale onu grafiku vykresluje na základě výstupu zvuku ze zařízení a tato knihovna je nastavena tak, že ten zvuk v podstatě nahrává zpátky z výstupu do vstupu zařízení (všechno samozřejmě softvérově). Systémové oprávnění přístupu k mikrofonu je považováno jako citlivé a uživatel s ním musí výslovně souhlasit (od API 21). Proto jsem v hlavním okně aplikace vytvořil tlačítko s funkcí udělování systémového oprávnění kdy si uživatel může vybrat, zdali oprávnění udělí, nebo jestli vizualizér používat nechce.

Vizualizér je zobrazen v nové Linear Layout, který má pevnou výšku 140pixelů a variabilní šířku, která se mění na základě rozlišení jednotlivého zařízení. V závislosti na frekvenci by měl uživatel vidět od jedné do několika jednotek vln. Standartně se vizualizéru nastaví AudioSessionID (ID zdroje zvuku) pro jeden konkrétní zdroj zvuku (přehrávač médií, generátor signálu, …), zde jsem nastavil ID na hodnotu 0, což je master mix output z celého zařízení. Vizualizér bude tedy zobrazovat průběhy jakéhokoliv zvuku, který telefon bude v dané době přehrávat. Toto řešení jsem zvolil proto, aby se zobrazovaly průběhy ze všech generátorů současně (tedy i při přehrávání souzvuků), protože vizualizér neumí nastavit více ID najednou. Kód vizualizéru uvádím v příloze A, část 2. Vizualizér.

Bohužel, v průběhu testování jsem narazil na situace, kdy především starší zařízení nebyla schopna Vizualizér vykreslit. Se snižující se verzí androidu klesá i úspěšnost vykreslení. Vizualizér plně funguje v androidu ve verzi 9.0, 8.1 a 8.0, tzn. API 26 až 28. viz *4.10.4 Vizualizér.*

4.8 Sestavování akordů a jiných souzvuků

Jak jsem již popsal v návrhu aplikace, uživatel si bude moct sestavit 5 různých akordů či jiných souzvuků, které se uloží do paměti a které si poté bude moct přehrát v hlavní Activity aplikace. Každý souzvuk se skládá ze 4 tónů. Sestavování souzvuků probíhá třemi způsoby.

První možnost je vybrat si pomocí Spinneru jeden z již přednastavených durových akordů. Akordy začínají na tónu c´ (262 Hz) a jsou ve složení Cdur, Ddur, Edur, Fdur, Gdur, Adur, Hdur. Výběr není nutné nijak potvrzovat, aplikace si hodnotu sama uloží ve formě int (pořadí vybrané možnosti začínající od 0) do Shared Preferences pod určitým klíčovým slovem a vyvolává jí nejen při nastavení frekvence souzvuku, ale také při zobrazení vybrané možnosti v Activity *VyberAkordu* jako String.

Druhou možností je sestavit souzvuk z jednotlivých tónů a půltónů. Uživatel zde má na výběr pro každý ze čtyř tónů souzvuku určit si oktávu (malá, jednočárkovaná, dvojčárkovaná) a pak samostatný tón (c, cis, d, dis, e, f, fis, g, gis, a, b, h). Podobně jako při výběru přednastavených akordů není nutné výběr potvrzovat, aplikace si hodnoty automaticky ukládá do Shared Preferences tak, jak jsem popsal výše.

Třetí možností je zadání čtyř frekvencí, které reprezentují 4 tóny požadovaného souzvuku. Tyto číselné hodnoty se musí pohybovat v rozsahu 1 Hz až 1000 Hz a potvrzují se tlačítkem, které hlídá správnost zadaných hodnot a ukládá je do paměti. Horní limit 1000 Hz je nastaven z důvodu optimalizace a zabrané velikosti použité paměti s ohledem na reálné využití. Spodní limit logicky vychází z podstaty generátoru akordů, aby se nesnažil generovat signál o nulové případně záporné frekvenci. Zadávat jdou tedy pouze celá kladná čísla.

Tyto 3 způsoby zadávání hodnot se zapínají nebo vypínají pomocí Switche, který hlídá to, aby byla vybraná vždy pouze jedna možnost a zároveň je graficky odděluje. Druhý Switch může být zapnutý až poté, co je zapnutý první přepínač. Vypnutím druhého Switche dojde ke zneviditelnění nabídky, která se nachází pod ním. První přepínač také řídí načítání *ArrayAdapteru,* tedy zobrazení a práci s položkami ve Spinneru, v závislosti na jeho poloze. I tyto polohy Switchů se ukládají jako int do Shared Preferences pod vlastním klíčovým slovem.

Obr. 4.2: Nová podoba Activity Akord1

4.9 Možnosti dalšího rozšíření

V aplikaci se v případném budoucím vývoji dá změnit například způsob ovládání ekvalizéru nebo jiných filtrů tak, jak bylo popsané v návrhu aplikace.

Dále se mi nepodařilo implementovat jiné průběhy LFO než sinus. Zde by se dalo uvažovat o čtverci, pile případně jiném náhodném signálu.

Možnost pro rozšíření je i u volby tvaru periodického signálu. Daly by se přidat i jiné tvary jako trojúhelník, případně šumové signály.

U volby pro výběr akordů a jiných souzvuků by se dalo zapracovat na uživatelsky přívětivější variantě, tento způsob ovládání mi přijde trochu kostrbatý, bohužel jsem na žádný jiný takto efektivní způsob nepřišel.

Dala by se přidat také možnost změny výchozí frekvence pro hlavní melodii (Y = 0, 440 Hz).

4.10 Problémy při realizaci

V této kapitole se budu věnovat několika problémům, se kterými jsem se setkal během vývoje aplikace.

Vývoj aplikace mi trval dlouho jednak proto, že jsem měl málo zkušeností s programováním a také proto, že jsem se setkal s mnohými problémy, některé z nich nejsem schopen vyřešit doteď.

4.10.1 Generátor hlavní melodie

První a nejdůležitější problém jsem řešil se samotným generátorem zvuku. Výběr knihovny AudioTrack byl celkem jasný, ale správná implementace, a hlavně optimalizace byla náročná. Když jsem měl funkční generátor, vytvořil jsem rovnici pro výpočet výsledné frekvence na základě dat získané ze senzoru a snažil se jí dostat do generátoru v reálném čase. Problém byl v tom, že jsem používal *AudioTrack* v *MODE_STATIC* a s každou změnou frekvence, tedy s každou novou obnovou dat pocházející z akcelerometru, začal generátor generovat novou vlnu na nové frekvenci zase od začátku, tedy od amplitudy 0. Vzhledem k tomu, že obnovovací frekvence akcelerometru není ani vzdáleně periodická, vznikaly na výstupu ve zvuku „lupance“. Ty byly právě způsobeny tím, že ten starý zvuk se zastavil díky obnově dat ze senzoru a získání nové frekvence okamžitě, tedy na náhodném místě v průběhu vlny. Tento rozdíl v amplitudě oproti počátku způsoboval právě ony nechtěné zvuky. Malá zajímavost, když jsem v tomto stavu naklonil telefon do úrovně $\pm 2,\overline{2}$ na ose Y, tedy 349 Hz nebo 698 Hz (tón f), zvuk byl čistý bez jakýchkoliv „lupanců“.

Tento problém jsem vyřešil tak, že jsem převedl *AudioTrack* do *MODE_STREAM* a zároveň do jeho vlastního *Threadu,* aby nebyl ovlivněn ničím jiným. Toto opatření způsobilo jiný problém – odezvu. Předávání informací mezi *Thready* není okamžité, tudíž nastává prodleva i na tom nejdůležitějším – změna frekvence, a to odhadem 100 až 300 ms. Tento problém je bohužel neřešitelný, protože těch míst, kde se může nějaký proces opozdit od náklonu zařízení až po změnu generované frekvence, je mnoho a největší mírou do toho přispívá reakce samotného senzoru. Tento problém jsem musel řešit i opačným směrem – při předávání *AudioSessionID* jak jsem již popsal v kapitole *4.3 Hlavní Activity aplikace.*

S akcelerometrem se pojí ještě další 2 problémy. První z nich je hardwarový – Při prudkém pohybu telefonu, tedy při skokové změně frekvence mají zabudované akcelerometry svou setrvačnost, která se projevuje nepřesně zahranou frekvencí dalšího tónu po prudkém pohybu – zpravidla o tón vyšší nebo nižší ve směru pohybu. Tohle je, bohužel, věc, se kterou se nedá nic moc dělat.

Druhý problém je kombinace hardwaru a softwaru. Tak jak jsem uvedl v kapitole *3.3.1 Tvar signálu*, při vložení dalších tónů mezi dva sousedící, se změna hlavní frekvence provádí po mnohem menších úsecích, tedy se stává citlivější i na drobný pohyb telefonu. Cílem tohoto nastavení bylo umožnit měnit frekvenci hlavní melodie glissandovitě. To se částečně povedlo – při pomalém pohybu telefonu je skok mezi dvěma sousedícími

frekvencemi sice menší, ale stále je patrný a při rychlejším pohybu telefonem musí generátor zareagovat na rychlou změnu frekvence, tudíž dojde ke skokové změně. Glissando jako takové se tedy nekoná, pouze se k tomu může uživatel přiblížit pomalým pohybem telefonu. Toto je problém především rychlostí čtení dat ze senzoru a rychlostí výpočtů prováděných při změně frekvence.

4.10.2 Generování souzvuků

V původním návrhu jsem počítal s tím, že 5 přednastavených akordů se bude moct dát spustit zároveň. To by obnášelo, že by v aplikaci muselo být přítomno mnohem víc generátorů najednou, konkrétně 20 (4 generátory na 5 akordů) plus generátor hlavní melodie. Teoreticky to možné je, v praxi jsem se setkal s obrovským zpomalením zařízení, které se projevovalo hlavně velice pomalými reakcemi akcelerometru až za míru použitelnosti. Toto zpomalení si vysvětluji hlavně naplněním všech Bufferů, tedy špatný přístup k paměti ze strany knihovny, nebo špatnou optimalizací z mojí strany. Nyní je tedy možné přehrát pouze jeden z pěti souzvuků současně, který generují 4 generátory *AudioTrack*. Myslím si, že tento stav není nijak omezující vzhledem k použitelnosti aplikace.

4.10.3 Ekvalizér

U této funkce jsem se opět odchýlil od mého původního návrhu. Původně jsem zamýšlel vytvořit ekvalizér, kde by si uživatel mohl vybrat středovou frekvenci a poté pomocí pásmové propusti nebo pásmové zádrže tuto frekvenci (a její okolí) filtrovat. Bohužel při hlubším zkoumání originální Android knihovny ekvalizéru jsem zjistil, že je k ní, mírně řečeno, nedostatečná dokumentace a myslím si, že to tato knihovna ani neumožňuje. Rozhodl jsem se tedy pojmout ekvalizér jinak (viz *4.6 Ekvalizér*) a zjistil jsem, že toto řešení je dostatečné. Originální Android ekvalizér je případně možný rozšířit až na 14 pásem, ale myslím si, že pro potřeby této aplikace za prvé není nutný takto podrobný ekvalizér, za druhé by se to ani nevešlo v tomto stavu do GUI Hlavní Activity aplikace.

4.10.4 Vizualizér

Jak jsem již popsal výše, ne na všech zařízeních se vizualizér vykresluje. Tato chyba může mít mnoho důvodů. Jedna z možností je málo vyhrazené paměti pro vykreslování na starších zařízeních, nebo (nepravděpodobně) špatné předávání systémového oprávnění k použití mikrofonu. Může to být i tím, že každé zařízení má jiné AudioSessionID pro master output, opět mi to však nepřijde moc pravděpodobné. Zde bohužel není způsob (nebo jsem ho nenašel) jak toto master output ID získat. V každém případě ani jedno zařízení, na kterém vizualizér nefungoval, nevypsalo žádnou chybovou hlášku, nebo jinou zprávu o chybě či nefunkčnosti. V AVD na API 25 vizualizér fungoval bez problémů, na testovacím telefonu (stejné API) nefunguje, na jiném telefonu (API 23) funguje jen někdy, v podstatě náhodně. Na telefonech od API 26 funguje bez problémů. Z tohoto důvodu nevím, kde mám hledat chybu, nebo co opravit, uvedená knihovna opět nemá moc dobrou dokumentaci a ani jinde na internetových fórech jsem žádný podobný problém nenašel.

4.10.5 LFO

Zde jsem řešil (a nevyřešil) problém spíše matematický než programátorský. Dalším logickým krokem kolem LFO bylo přidat i jiné harmonické průběhy než sinus. Ze začátku jsem se snažil implementovat alespoň stejné tvary jako hlavní melodie, tedy čtverec a pilu. V podstatě jsem vycházel z rovnice pro výpočet okamžité hodnoty napětí, tedy:

$$u = U_{\mathrm{n}} \sin(\Omega t + m_{FM} \cdot \sin \omega t) \tag{4.2}$$

Kdy U_n je amplituda hlavního signálu, v mém případě 32767 a Ωt v mém případě zastupuje rovnice pro výpočet fáze, viz *Výpis kódu 4.3: Generátor sinusového průběhu MODE_STREAM*. m_{FM} je standartně modulační index, v mém případě amplituda signálu LFO, ωt opět zastupuje rovnice pro výpočet fáze LFO.

Vzhledem k tomu, že kód pro výpočet čtvercového průběhu v mém případě vychází z rovnice pro výpočet sinusového průběhu, pouze upravuji tvar výsledné vlny hodnotou ±maximální amplitudy na základě hodnoty fáze, nevěděl jsem, jakým způsobem mám čtverec do této rovnice implementovat. Ve výsledku tento LFO vždy zněl stejně jako sinus, protože to v podstatě byla sinusovka, na kterou neplatily žádné pokusy o přetvarování vlny do tvaru čtverce.

U pily je situace trochu odlišná, protože v rovnici pro generování pily u hlavní melodie nevycházím z už daného harmonického průběhu *Math.Sin()*. Zde jsem řešil, jak to tedy vložit do rovnice 4.2, protože všechny moje pokusy buď nefungovaly vůbec, nebo opět zněly úplně stejně jako sinus, protože, pravděpodobně díky špatnému kódu, rovnice pro výpočet hlavní vlny nějakým způsobem neregistrovala podmínky, jaké jsem tam nastavil.

Tak jako tak je to možnost pro případný další budoucí vývoj aplikace a v tuto dobu může být frekvence hlavní vlny modulována pouze LFO ve tvaru sinus.

4.11 Hudební potenciál a jiné využití aplikace

Dle mého názoru existují 3 možnosti využití aplikace.

První možnost je zkusit pomocí této aplikace zahrát (napodobit) již existující melodii nebo písničku. Toto řešení je technicky možné, ale prakticky naráží na několik problémů. Za prvé má uživatel k dispozici pouze 5 akordů, což by nemusel být zas takový problém pro jednoduché melodie, ale hlavně se musí naučit správně korigovat frekvenci hlavní melodie. Z vlastní zkušenosti vím, že (i když jsou rozestupy v pohybu mezi tóny poměrně velké) není tak jednoduché najít konkrétní tóny „naslepo“. Chce to jistý cvik a také držení každého zařízení je trochu odlišné stejně tak jako rychlost reakce senzoru. Prakticky není možné zahrát konkrétní melodii bez chyb ve frekvenci hlavní melodie. Za druhé musím vzít v potaz to, že aplikace generuje pouze omezený počet typů vln, které se budou zvukově (barevně) jen velice málo podobat nějaké známé melodi.

Druhá možnost je, jak již název této práce napovídá, čistě experimentální. Uživatel si může nastavit naprosto libovolnou kombinaci souzvuků a generovat hlavní vlnu naprosto náhodně a libovolně. Zde lze podotknout, že frekvence hlavní melodie jde rytmizovat na základě dostatečně přesného pohybu se zařízením. V kombinaci s jednoduchým

modulováním hlavní vlny pomocí LFO lze vytvořit opravdu hodně alternativní typy melodie. Tento druh generování zvuků je opět jen pro zábavu případně inspiraci pro další jinou hudební tvorbu.

Třetí možnost je edukační. Zde bych vyzdvihl hlavně způsob tvoření souzvuků. Uživatel si zde může navolit libovolný souzvuk a vyzkoušet si, jak to zní dohromady s jiným souzvukem případně frekvencí hlavní melodie. Díky tomu, že v dnešní době je možné tuto aplikaci mít neustále při sobě, lze si takové věci nastavit a vyzkoušet prakticky kdykoliv bez toho, aniž by uživatel musel čekat až přijde k počítači a vyzkouší si to na jiném, pokročilejším, syntetizátoru. Dle mého názoru to lze to využít jak při tvorbě jiné hudby, tak při výuce například hudební teorie.

Tvoření souzvuků tímto způsobem je zároveň věc, která činí tuto aplikaci jedinečnou. Nenašel jsem žádnou jinou veřejně přístupnou aplikaci, kde si může uživatel navolit akordy nebo jiné souzvuky podobným způsobem a k tomu zároveň přehrávat jinou melodii. Jak jsem psal výše, vzhledem k tomu, že se aplikace ani nesnaží napodobovat jiné hudební nástroje, jde tedy o čistě syntetický zvukový generátor a cílí na jiný typ uživatelů než většina aplikací v obchodě Google Play.

Nad samotným skládáním souzvuků může uživatel strávit opravdu hodně času vzhledem k tomu, že možností je téměř neomezené množství. Trochu uživatelsky nepřívětivé je to, pokud uživatel do tohoto nastavování frekvencí souzvuků chodí často – hledá nové zvuky, nebo neustále experimentuje s nastavením, protože poté je to relativně zdlouhavý proces se nastavením aplikace proklikat. Když jdu například změnit frekvenci 2. tónu u akordu 1, musím asi 8 – 9x kliknout, abych změnu provedl a vrátil se zpět do Hlavní Activity. Pro změnu jenom jednoho parametru je to relativně dlouhá doba.

Celkově mě překvapilo, že na první poslech ne moc libozvučný souzvuk může znít úplně jinak zároveň se zvukem hlavní melodie. Tato kombinace se dá využít například pro vytváření jednoduchých návrhů beatů, tedy základní melodický rytmus do moderní elektronické hudby. Nejdůležitější je se naučit relativně přesně pohyby se zařízením, pak už jenom stačí dodržovat zvolené tempo tohoto pohybu. Právě z tohoto důvodu by se dalo do budoucna uvažovat o možnosti změny výchozí frekvence (440 Hz) výš nebo níž, aby tyto pohyby nebyly tak krkolomé. Hodnotu této středové frekvence by si mohl uživatel sám zvolit.

5 ZÁVĚR

Cílem práce bylo kompletně navrhnout originální experimentální hudební nástroj pro mobilní zařízení. Originalita a experiment tohoto nástroje tkví ve způsobu ovládání – hlavní melodie se ovládá pohybem zařízení, a přitom zůstává interakce s displejem, kde může uživatel signál dále doplňovat nebo upravovat v reálném čase za zvuku jednoho z pěti akordů nebo jiných souzvuků.

V teoretickém úvodu práce rozebírám základní teoretické předpoklady k vytvoření aplikace a jejich součástí. Dále stručně zmiňuji nejdůležitější základní hudební pojmy, ze kterých vycházím při návrhu aplikace.

V samotném návrhu aplikace se podrobně zabývám grafickým návrhem a popisem ovládacích prvků. Při vývoji jsem narazil na nedostatky v grafickém návrhu i popisu funkcí, které jsem shrnul v kapitole *4.1.*

Zatím jsem nenašel žádnou jinou mobilní aplikaci stejného zaměření. Hudební programy ovládané pohybem zařízení sice existují, ale neumožňují se zvukem dále pracovat, naopak aplikace podporující úpravu zvuku neumožňují ovládání melodie pohybem zařízení.

Během vývoje aplikace jsem se setkal s problémy spojené hlavně se samotným generováním zvuku, většina z nich souvisela se špatnou optimalizací. Tyto problémy pramenily mimo jiné i s mým nedostatkem zkušeností s vývojem aplikací pro OS Android, kterých jsem zde získal skutečně hodně. Aplikace jako taková je nyní plně funkční a hotová s tím, že v kapitole *4.9* jsem navrhnul směr možného dalšího vývoje. Jako hlavní nedostatek mé aplikace považuji nemožnost si zvolit jiné průběhy u LFO.

LITERATURA

[1] Android (Operační systém). [Online]
https://cs.wikipedia.org/wiki/Android_(operační_systém)#Historie.

[2] Inc., Google. Developer Guides. [Online]
https://developer.android.com/guide/index.html.

[3] Konečný, Matěj. Vyvíjíme pro Android: Intenty, intent filtry a permissions. [Online]
https://www.zdrojak.cz/clanky/vyvijime-pro-android-intenty-intent-filtry-a-permissions/.

[4] Inc., Google. Intent. [Online]
https://developer.android.com/images/components/intent-filters@2x.png.

[5] Konečný, Matěj. Content Providery. [Online]

[6] Inc., Google. Content Providers. [Online]
https://developer.android.com/guide/topics/providers/images/content-provider-overview.png.

[7] Inc., Google. Activity lifecycle. [Online]
https://developer.android.com/images/activity_lifecycle.png.

[8] Inc., Google. Activity. [Online]
https://developer.android.com/reference/android/app/Activity.html.

[9] Inc., Google. Android Studio. [Online]
https://developer.android.com/studio/features.html.

[10] GUŠTAR, Milan. *Elektrofony: historie, principy, souvislosti. Část II, Elektrické nástroje.* Praha : Uvnitř, 2008. ISBN 978-80-239-8447-7.

[11] Troszok, Daniel. Zvuková syntéza. [Online] http://www.muzikus.cz/pro-muzikanty-clanky/Zvukova-synteza-tema-mesice~24~leden~2013/.

[12] SYROVÝ, Václav. *Hudební akustika.* v Praze : Akademie muzických umění, 2013. ISBN ISBN978-80-7331-297-8.

[13] Materiály do předmětu JZT - tvorba umělého zvuku.

[14] Uher, Zbyněk. Historie VT: Historie, současnost a budoucnost určování frekvencí tónů v hudebním SW. . [Online]
https://www.fi.muni.cz/usr/jkucera/pv109/2001/xuher/xuher.html.

[15] Doc. RNDr. Jan Obdržálek, CSc., MUStr. Teorie ladění. [Online]
http://utf.mff.cuni.cz/~jobdr/download/Vyklad%204-2g.pdf.

[16] ZENKL, Luděk. *ABC hudební nauky.* Praha : Editio Barenreiter, 2000. ISBN 80-86385-01-9.

[17] Lda, Imaginando. DRC - Polyphonic Synthesizer. [Online]
https://play.google.com/store/apps/details?id=com.imaginando.drc.

[18] Raphlinus. MusicSynthesizer for Andorid. [Online]
https://github.com/google/music-synthesizer-for-android.

[19] Stegner, Glen. MiniMogueVA. [Online]
http://www.vst4free.com/free_vst.php?id=405.

[20] Inc., Google. SenzorManager. [Online]
https://developer.android.com/reference/android/hardware/SensorManager.

[21] Mitchell, Daniel R. *BasicSynth.* místo neznámé : Lulu.com, 2016. ISBN: 978-0-557-02212-0.

[22] Inc., Google. Equalizer. [Online]
https://developer.android.com/reference/android/media/audiofx/Equalizer.

[23] Inc.,Google. Visualizer. [Online]
https://developer.android.com/reference/android/media/audiofx/Visualizer.

SEZNAM SYMBOLŮ, VELIČIN A ZKRATEK

ABI – Application Binary Interface – Nízko úrovňové aplikační rozhraní

ADSR – attack, decay, sustain, release

API – Application Programming Interface – rozhraní pro programování aplikací

AVD – Andorid Virtual Device – Virtuální zařízení Android

DCO – Digitally Controlled Oscillator – digitálně řízený oscilátor

DPI – dots per inch – body na palec

FFT – Fast Fourier transform – rychlá Fourierova transformace

GUI – Graphical User Interface – grafické uživatelské rozhraní

ISO – International Organization for Standardization – Mezinárodní organizace pro normalizaci

LC – obvody z cívky a kondenzátoru

LFO – low frequency oscilator – nízkofrekvenční oscilátor

OHA – Open Handset Alliance – uskupení výrobců mobilních telefonů

SDK – Software development kit – softvérový vývojářský nástroj

XML – eXtensible Markup Language – rozšiřitelný značkovací jazyk

SEZNAM PŘÍLOH

Příloha A

Části Kódu

1. Ekvalizér
2. Vizualizér
3. Výpočet frekvence hlavní melodie

Příloha B

CD

Příloha A

Části kódu

1. Ekvalizér

```
public void InitEQ() {
    eq = new Equalizer(0, ID);
    eq.setEnabled(true);

}
private void releaseEQ() {
    if (eq != null) {
        eq.release();
        eq = null;
    }
}

public void setupEQ() {

    final short lowerEqualizerBandLevel = eq.getBandLevelRange()[0];
    final short upperEqualizerBandLevel = eq.getBandLevelRange()[1];

    for (short i = 1; i < 4; i++) {
        final short equalizerBandIndex = i;

        bar = new SeekBar(this);
        TextView text = new TextView(this);

        if (equalizerBandIndex == 1) {
            bar = findViewById(R.id.sb_low);
            text = findViewById(R.id.tv_low);
        } else if (equalizerBandIndex == 2) {
            bar = findViewById(R.id.sb_mid);
            text = findViewById(R.id.tv_mid);
        } else if (equalizerBandIndex == 3) {
            bar = findViewById(R.id.sb_high);
            text = findViewById(R.id.tv_high);
        }

        int setmax = (upperEqualizerBandLevel - lowerEqualizerBandLevel);

        bar.setMax(setmax);

        int progress = (eq.getBandLevel(equalizerBandIndex)) +
upperEqualizerBandLevel;
        bar.setProgress(progress);
        text.setText(String.valueOf(eq.getBandLevel(equalizerBandIndex)));

        final TextView finalText = text;
        bar.setOnSeekBarChangeListener(new
SeekBar.OnSeekBarChangeListener() {
            public void onProgressChanged(SeekBar seekBar, int progress,
                                          boolean fromUser) {
```

```
 InitEQ();
                if (equalizerBandIndex == 1) {
                    inProgress1 = progress;
                } else if (equalizerBandIndex == 2) {
                    inProgress2 = progress;
                } else if (equalizerBandIndex == 3) {
                    inProgress3 = progress;
                }
                eq.setBandLevel(equalizerBandIndex,
                        (short) (progress + lowerEqualizerBandLevel));
                prog = (progress + lowerEqualizerBandLevel) / 100;
                finalText.setText(String.valueOf(prog));
            }

            public void onStartTrackingTouch(SeekBar seekBar) {
            }

            public void onStopTrackingTouch(SeekBar seekBar) {
            }
        });
    }
}

private void afterInitEQ() {
    final short lowerEqualizerBandLevel = eq.getBandLevelRange()[0];

    for (short i = 1; i < 4; i++) {

        if (i == 1) {
            bar = findViewById(R.id.sb_low);
            eq.setBandLevel(i, (short) (inProgress1 +
lowerEqualizerBandLevel));
        } else if (i == 2) {
            bar = findViewById(R.id.sb_mid);
            eq.setBandLevel(i, (short) (inProgress2 +
lowerEqualizerBandLevel));
        } else if (i == 3) {
            bar = findViewById(R.id.sb_high);
            eq.setBandLevel(i, (short) (inProgress3 +
lowerEqualizerBandLevel));

        }


    }
}
```

2. Vizualizér

```
public void InitVis() {
    mVisualizer = new Visualizer(0);
}

private void setupVis() {

    mLinearLayout = findViewById(R.id.LinLay);
    mVisualizerView = new VisualizerView(fullscreen_start.context_start);
    mLinearLayout.addView(mVisualizerView);
    InitVis();
    System.out.println(Arrays.toString(Visualizer.getCaptureSizeRange()));
    mVisualizer.setCaptureSize(Visualizer.getCaptureSizeRange()[1] / 2);
    mVisualizer.setDataCaptureListener(new
Visualizer.OnDataCaptureListener() {
        public void onWaveFormDataCapture(Visualizer visualizer, byte[]
bytes,
                                          int samplingRate) {

            mVisualizerView.updateVisualizer(bytes);
        }

        public void onFftDataCapture(Visualizer visualizer, byte[] bytes,
int samplingRate) {
        }
    }, Visualizer.getMaxCaptureRate() / 2, true, false);
}

private void releaseVis() {
    if (mVisualizer != null) {
        mVisualizer.setEnabled(false);
        mVisualizer.release();
        mVisualizer = null;
        mLinearLayout.removeView(mVisualizerView);
    }
}
```

3. Výpočet frekvence hlavní melodie

```
public void onSensorChanged(SensorEvent event) {

    //float x = event.values[0];
    float y = event.values[1];
    //float z = event.values[2];

    double tony = 10.0 / (16 * NovyPomer);
    double PoradiTonu = (y / tony);
    int ZaokrouhlenePoradi = (int) PoradiTonu;

    double mocnina1 = Math.pow(2, ZaokrouhlenePoradi);
    double odmocnina1 = (Math.pow(mocnina1, (1.0 / (pomer * NovyPomer))));
    double senzorFreq = 440 * odmocnina1;

    konecnafrekvence = (int) Math.round(senzorFreq);
```

```
    PTSQ.setTuneFreq(konecnafrekvence, lfo_freq, AmplitudaLFO);
    PTSI.setTuneFreq(konecnafrekvence, lfo_freq, AmplitudaLFO);
    PTSW.setTuneFreq(konecnafrekvence, lfo_freq, AmplitudaLFO);

}


@Override
public void onAccuracyChanged(Sensor sensor, int accuracy) {

}
int pomerInt = preferencesPomer.getInt("Pomer", HlSignalInt);


if (pomerInt == 0) { //kvůli mozne jine zmene pomeru zvoleno reseni if/else
    NovyPomer = 1;
} else if (pomerInt == 1) {
    NovyPomer = 2;
} else if (pomerInt == 2) {
    NovyPomer = 3;
} else if (pomerInt == 3) {
    NovyPomer = 4;
} else if (pomerInt == 4) {
    NovyPomer = 5;
} else if (pomerInt == 5) {
    NovyPomer = 6;
} else if (pomerInt == 6) {
    NovyPomer = 7;
}
```

Příloha B

CD

Přiložené CD obsahuje:

Zkompilovanou aplikaci ve formátu .apk, složku celého projektu z Android Studia.

Barak_BP.apk

Barak_BP (složka)