

Vysoká škola ekonomická v Praze

Fakulta informatiky a statistiky

Katedra informačních technologií

Studijní program: Aplikovaná informatika

Obor: Informatika

Vývoj mobilní aplikace pro hudebníky

BAKALÁŘSKÁ PRÁCE

Student : Miroslav Novotný

Vedoucí : Ing. Jarmila Pavlíčková, Ph.D.

Oponent : Ing. Magda Kutíšová

2015

Prohlášení

Prohlašuji, že jsem bakalářskou práci zpracoval samostatně a že jsem uvedl všechny použité prameny a literaturu, ze které jsem čerpal.

V Praze dne 6. května 2015

.....
Miroslav Novotný

Poděkování

Rád bych poděkoval paní Ing. Jarmile Pavlíčkové, Ph.D. za trpělivost, ochotu a možnost výběru vlastního tématu. Dále bych chtěl poděkovat své rodině a kamarádům, kteří mě podporovali a pomáhali mi po celou dobu studia.

Abstrakt

Cílem této bakalářské práce je návrh a vývoj aplikace pro hudebníky na platformě Android. Teoretická část práce je koncipovaná jako úvod do teorie zvuku. Nejprve se ovšem věnuji popisu architektury systému Android a jednotlivým komponentám, které ji tvoří. Následují kapitoly zkoumající zvuk od jeho vzniku, přes proces převodu do digitální podoby, až k popisu jednotlivých rozhraní pro jeho zaznamenávání a zpracovávání v prostředí systému Android. Praktická část je pak věnovaná návrhu a vývoji aplikace, která slouží jako pomůcka při cvičení hry na hudební nástroj. Tvoří ji komponenty: ladička, metronom a EarTrainer. Pro realizaci ladičky bylo nutné nastudovat potřebnou teorii, nutnou pro podporu hlavní funkcionality výpočtu frekvence ze zaznamenaného audiosignálu. Z tohoto důvodu se v práci nachází i teoretický úvod do Fourierových transformací. Poslední část je věnovaná srovnání přesnosti výpočtu komponenty ladička s konkurencí. Výsledná aplikace byla publikována na Google Play.

Klíčová slova

Vývoj pro Android, Android aplikace, Analýza zvuku, Ladička, Metronom, Fourierova transformace, FFT

Abstract

The aim of this bachelor's thesis is to design and develop an Android application for musicians. The theoretical part, is conceived as an introduction to the theory of sound. Firstly, I describe the architecture of Android and the individual components that make it up. There are sub-sequent chapters exploring the sound from its inception, through the process of converting to digital form, to a description of the individual interfaces for the recording and processing from within the environment of an Android system. The practical part, is devoted to designing and developing an application that serves as a tool for practicing playing a musical instrument. The application consists of various features such as: tuner, metronome and an EarTrainer. To understand the component of a tuner it has been necessary to study a theory needed to support the main functionality of calculating the frequency of the recorded audio signal. For this reason, this thesis is also a theoretical introduction to the Fourier transformation. The last part of this thesis is focusing on how the tuner compares with the calculation accuracy of other competitors. Finally, The application was published for the Google Play store.

Keywords

Android Development, Android application, Sound Analysis, Tuner, Metronome, Fourier transformation, FFT

Obsah

Prohlášení.....	ii
Poděkování	iii
Abstrakt.....	ii
Klíčová slova	ii
Abstract.....	ii
Keywords	ii

Obsah iii

1 Úvod	1
1.1 Vymezení tématu.....	1
1.2 Důvod výběru tématu	2
1.3 Cíle práce.....	2
1.4 Způsob dosažení cíle.....	2
1.5 Předpoklady a omezení práce.....	3
1.6 Struktura práce	3
2 Komentovaná rešerše prací	4
3 Seznámení s Androidem.....	6
3.1 Obecně.....	6
3.2 Historie	7
3.3 Architektura.....	7
3.3.1 Linux Kernel	8
3.3.2 Knihovny.....	8
3.3.3 Android Runtime	8
3.3.4 Aplikační Framework	9
3.3.5 Aplikace	9
4 Digitalizace audiosignálu	10
4.1 Teorie.....	10
4.2 Analogový vs. digitální zvuk	10
4.3 Analogový zvuk.....	11
4.4 Digitální zvuk.....	12
4.5 Proces digitalizace.....	13
4.5.1 Pulzně kódová modulace	13
4.5.2 Vzorkování.....	14
Nyquistův (Shannonův, Kotělnikovův) teorém	15
4.5.3 Kvantizace	16
4.5.4 Kódování.....	17

5	Android a zvuk	18
5.1	MediaPlayer	20
5.2	SoundPool	21
5.3	AudioTrack	22
5.4	AudioRecord	24
5.5	MediaRecorder	24
6	Praktická část - vývoj aplikace	26
6.1.1	Analýza a požadavky	26
6.1.2	Rešerše aplikací ve světě	26
6.1.3	Use case diagram	28
6.1.4	Struktura projektu	28
6.1.5	Obrazovka základního menu	30
6.2	Komponenta metronom	30
6.2.1	Analýza a požadavky	31
6.2.2	Příklad užití	31
6.2.3	Problémy při realizaci	31
6.2.4	Popis	32
6.2.5	Class Diagram	36
6.2.6	Obrazovka	37
6.3	Komponenta ladička	37
6.3.1	Analýza a požadavky	38
6.3.2	Příklad užití	41
6.3.3	Problémy při realizaci	41
6.3.4	Popis	42
6.3.5	Obrazovka	51
6.3.6	Porovnání	51
6.4	Komponenta EarTrainer	53
6.4.1	Analýza a požadavky	53
6.4.2	Příklad užití	53
6.4.3	Problémy při realizaci	54
6.4.4	Popis	54
6.4.5	Class Diagram	61
6.4.6	Obrazovka	62
6.5	Publikace	63
7	Závěr	64
	Seznam literatury	65
	Terminologický slovník	68
	Seznam obrázků, tabulek a výpisů kódu	69

Příloha 1. Kompletní Class Diagram	71
---	-----------

1 Úvod

Pro mnoho lidí znamená hudba prostředek k vyjádření sebe sama. Pokud jí budeme chtít porozumět a prožít hlouběji, nabízí se možnost posunout se z pozice posluchače dál a získat naprosto nový prožitek skrze hru na hudební nástroj. Sám již několik let hraji na kytaru a za tu dobu jsem měl možnost se seznámit a zahrát si s řadou dalších muzikantů. Ti se samozřejmě nejvíce liší ve svých hráčských dovednostech. Jsou mezi nimi lidé s hlubším hudebním vzděláním, absolventi lidových škol umění, ale úplně nejčastěji samouci.

Do poslední zmíněné skupiny se řadím i já, a proto mohu velmi přesně charakterizovat zásadní problém, který tuto skupinu spojuje. Pokud se člověk rozhodne učit na kytaru svépomoci, většinou to dopadá tak, že z počátku vynechá hudební teorii a rovnou se začne učit skladby a melodie svých oblíbených interpretů.

To je v dnešní době možné hlavně díky velkému množství výukových materiálů dostupných na internetu. Nachází se zde videa, kde vás lektor učí, kam přesně pokládat prsty na hmatníku a dokonale kopíruje původní skladbu. Dalším velmi oblíbeným materiálem jsou tabulatury. Jedná se o hudební zápis, ve kterém jsou jednotlivé tóny a akordy zapsány ve formě čísel, reprezentující konkrétní místo na hmatníku.

Tato forma výuky může bohužel lehce sklouznout k tomu, že se hráč nesoustředí na hudební projev jako takový, ale na mechanické přehrávání. Dokáže sice zahrát part správně, ale nerozumí mu. Velmi dobře se to ilustruje na příkladu, kdy hráči praskne struna a on i když má teoreticky ještě hodně možností, jak figuru zahrát, nemůže, umí pouze jeden postup. Další problém může nastat v situaci, kdy se samouk snaží hrát s jiným muzikantem. Pokud není zvyklý hrát podle metronomu, je velmi pravděpodobné, že nedokáže udržet tempo skladby.

1.1 Vymezení tématu

Práce se zabývá návrhem a vývojem aplikace na platformu Android určenou pro začínající i pokročilé hudebníky. U čtenáře se předpokládá alespoň základní znalost objektového programování, jazyka Java a XML.

1.2 Důvod výběru tématu

Při výběru pro mě hrálo velkou roli, aby se téma zabývalo něčím, co mě opravdu zajímá. Rozhodl jsem se tedy pro oblast programování, která se mi stala velmi blízkou po nástupu na Vysokou školu ekonomickou v Praze. Zde jsem mimo jiné absolvoval předměty spojené s jazykem Java, o který jsem vzbudil velký zájem a rozhodl se mu věnovat nejen ve studijním, ale také v pracovním životě. Jelikož se ve svém volném čase aktivně věnuji hře na kytaru, přišlo mi zajímavé propojit tyto dva odlišné světy skrze mobilní aplikaci, která bude sloužit jako cvičební pomůcka pro hudebníky.

1.3 Cíle práce

Hlavním cílem této bakalářské práce je vyvinout funkční aplikaci pro mobilní platformu Android. Tato aplikace by měla sloužit jako cvičební pomůcka pro začínající i pokročilé hudebníky. Bude tvořena třemi komponentami: ladička, metronom a EarTrainer.

Vedlejším cílem je přiblížit problematiku oblasti zaznamenávání a zpracovávání digitálního zvuku, a to jak v obecné rovině, tak v prostředí systému Android. Tato část je využita jako teoretický základ, potřebný pro porozumění dané oblasti, který je následně využit k návrhu a implementaci výše zmíněné aplikace.

Práce se zabývá konkrétní problematikou, proto jejím cílem není hlubší obecný popis platformy Android.

1.4 Způsob dosažení cíle

Pro dosažení cíle nejprve provádím rešerši prací zabývajících se podobným tématem. V dalším kroku popisuji literaturu potřebnou pro porozumění teoretickému základu spojenému se zaznamenáváním a zpracováváním audiosignálu na platformě Android.

V praktické části se věnuji samotné analýze a návrhu aplikace. Následně se zabývám výběrem vhodné technologie pro podporu funkcionalit. V posledním kroku se zaměřuji na samotnou implementaci.

1.5 Předpoklady a omezení práce

Jak již bylo zmíněno výše, u čtenáře se předpokládá alespoň základní znalost objektového programování, jazyka Java a XML. Při vývoji bylo použito prostředí Eclipse Juno, rozšířené o plugin ADT (Android Developer Tools), který usnadňuje vývoj. Součástí je i emulátor zařízení s operačním systémem. Ten ovšem nemohl být použit, jelikož ani ve své poslední verzi stále nepodporuje emulaci mikrofonního vstupu, která je nezbytná pro funkčnost komponenty ladička. Místo něj byl pro testovací účely použit reálný chytrý telefon. Konkrétně Xiaomi Mi2 s Androidem ve verzi 4.1.1.

V práci se objevují anglické názvy, jelikož pro některé z nich neexistují v českém jazyce ekvivalenty nebo se v softwarovém inženýrství nepoužívají.

1.6 Struktura práce

Práce je rozdělena do dvou hlavních částí. Prvních pět kapitol je teoretických, zbytek práce je věnován části praktické.

První kapitolu práce tvoří úvod. Následuje komentovaná rešerše obdobných prací. V třetí kapitole se zaměřuji na popis platformy Android. Na začátku se nachází jen obecný úvod a zbytek kapitoly je pak věnován jednotlivým prvkům architektury operačního systému. Čtvrtá kapitola se zabývá procesem digitalizace zvukového signálu. Nejprve je definováno několik teoretických pojmů nutných k porozumění další části, kterou je samotný popis procesu převodu analogového signálu na digitální. Pátá kapitola zkoumá možnosti zaznamenávání a zpracovávání zvuku na platformě Android.

V šesté kapitole se nachází praktická část práce. Po obecné analýze požadavků na aplikaci následuje rešerše podobných dostupných řešení. Každé komponentě aplikace je věnována zvláštní podkapitola, přičemž její stavba je u všech komponent stejná. V prvním kroku je provedena analýza požadavků, které musí komponenta splňovat. Dále je modelován případ užití. Následuje výpis problémů, ke kterým docházelo při vývoji a testování, a jejich řešení. Velká část každé podkapitoly je věnována popisu implementace funkcionalit. Dalším prvkem je pak zobrazení Class Diagramu a řešení grafického uživatelského rozhraní. Následuje poslední podkapitola věnována publikaci aplikace na Google Play.

2 Komentovaná rešerše prací

PLEVKA, Ondřej. *Vývoj a publikace aplikace pro platformu Android*. Praha, 2014. Bakalářská práce. Vysoká škola ekonomická v Praze. Vedoucí práce Ing. Jarmila Pavlíčková, Ph.D.

V první části se autor zaměřil na popis jednotlivých verzí systému Android. Následuje krátká teoretická pasáž o QR kódu. Zbytek práce je pak věnován návrhu a vývoji aplikace pro dynamickou správu sbírek na základě čárových či QR kódů. Poslední pasáž pojednává o publikaci a možnostech propagace aplikace na internetu.

UCHYTIL, Petr. *Vývoj mobilních aplikací pro platformu Android v programovacím jazyce Java*. Praha, 2014. Bakalářská práce. Vysoká škola ekonomická v Praze. Vedoucí Ing. Jarmila Pavlíčková, Ph.D.

Tato práce je opět věnována popisu operačního systému Android. Z počátku se autor zaměřuje na popis struktury systému. Následuje kapitola o jednotlivých komponentách, kterými je tvořena téměř každá Android aplikace. Další pasáž je pak věnována vytváření grafického uživatelského rozhraní. Praktickou část tvoří návrh a vývoj aplikace, umožňující ovládat hudební soubory umístěné na počítači prostřednictvím mobilního zařízení. V práci je často uváděno srovnání s platformou Java SE.

VACULA, Josef. *Vývoj aplikací pro Google Android*. Praha, 2011. Bakalářská práce. Vysoká škola ekonomická v Praze. Vedoucí práce doc. Ing. Alena Buchalcegová, Ph.D.

Stejně jako obě předešlé byla i tato práce napsána a obhájena pod Katedrou informačních technologií na Vysoké škole ekonomické v Praze. Struktura je koncipována jako obecný úvod do problematiky programování aplikací na platformě Android. Nachází se zde popis architektury systému, jednotlivých komponent, životního cyklu aplikace atd. V praktické části se pak autor zaměřil na návrh a vývoj aplikace pro správu výrobků, zákazníků a objednávek v obchodě.

ONDRUŠKA, Juraj. *Ozvučený kalendář pro OS Android*. Brno, 2012. Bakalářská práce. Masarykova univerzita v Brně. Vedoucí práce doc. RNDr. Ivan Kopeček, CSc.

Autor se zde zaměřil na implementaci aplikace pro osoby se zrakovým postižením. V první části se věnuje úvodu do problematiky spojené s tímto hendikepem. Následující kapitola řeší

teorii okolo hlasové syntézy a digitalizace zvuku. Ve zbytku práce se pak nachází popis návrhu a implementace aplikace. Výsledná aplikace využívá pro ukládání službu Google Kalendář a dokáže uživateli připomenout za pomoci hlasového výstupu blížící se událost.

CIBIRA, Jakub. *Webová aplikace pro výuku notového zápisu a trénink sluchového vnímání*. Brno, 2012. Diplomová práce. Masarykova univerzita v Brně. Vedoucí práce doc. Mgr. Radek Pelánek, Ph.D.

Na rozdíl od výše zmíněných prací se právě tato nezabývá vývojem aplikace pro systém Android. Cílem práce je vytvořit webovou aplikaci, která pomůže žákům základní školy při učení notového zápisu a tréninku hudebního sluchu. První kapitoly jsou věnovány úvodu do teorie vnímání a analýzy zvuku. Následuje komentovaná rešerše obdobných aplikací. Zbytek práce pak sleduje návrh a řešení zadání. Výsledná aplikace je napsána za pomoci jazyka JavaScript.

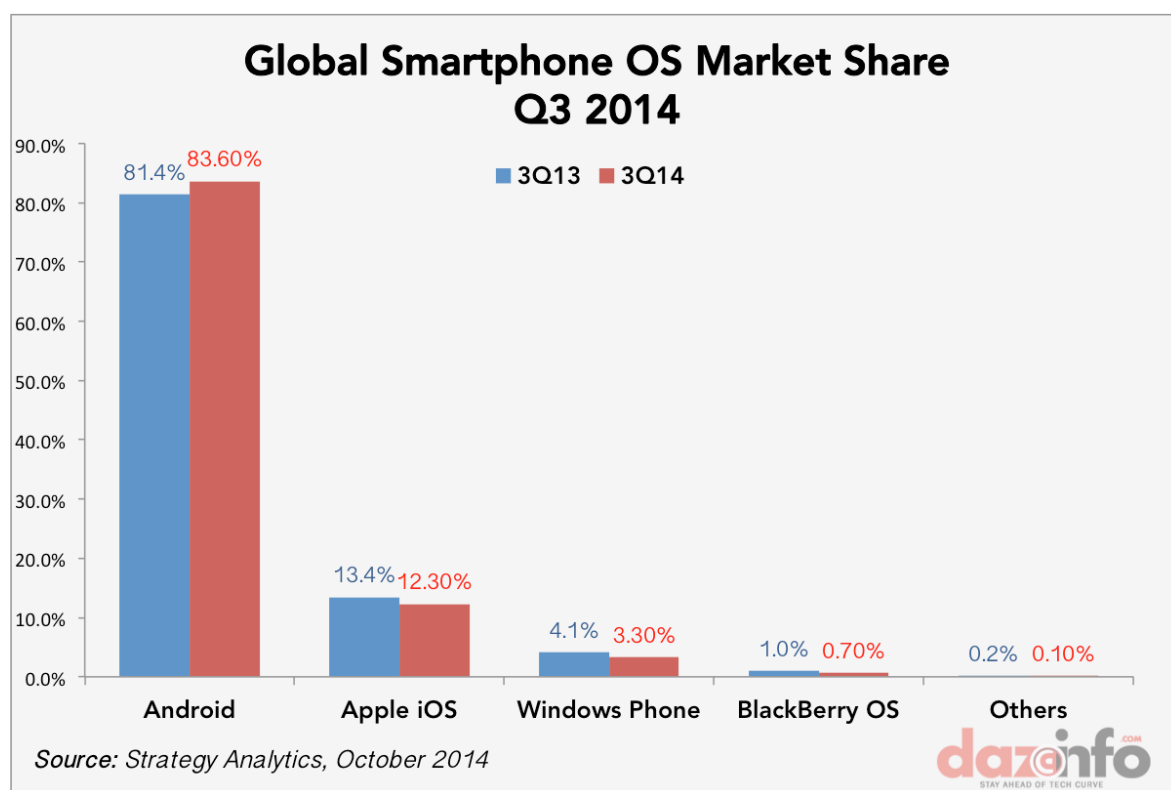
DOSTÁL, Filip. *Porovnání audio vzorků*. Zlín, 2014. Diplomová práce. Univerzita Tomáše Bati ve Zlíně. Vedoucí práce Ing. Viliam Dolinay, Ph.D.

Výstupem této práce je desktopová aplikace, která slouží pro porovnávání krátkých zvukových vzorků, jejíž jádro je napsané v jazyce Java. Autor se také zaměřil na popis teoretického kontextu včetně rešerše dostupných technologií. Aplikace má propracované grafické uživatelské rozhraní, na kterém dokáže zobrazit spektra porovnávaných vzorků.

3 Seznámení s Androidem

3.1 Obecně

Android je operační systém určený především pro chytré telefony a tablety. Díky licenci open-source, pod kterou je distribuovaný, mohla brzy vzniknout početná komunita vývojářů. Od roku 2010 je Android nejrozšířenějším operačním systémem na světě a zaujímá téměř 85 % podíl na trhu. V současné době se společnost Google rozhodla následovat trend chytrých televizí a připravuje vlastní platformu ve formě chytrého set-top boxu ([2], [3]). Jak je vidět na obrázku č. 1, ve třetím kvartále roku 2014 zaujímal systém Android téměř 84% podíl na trhu.



Obrázek 1:

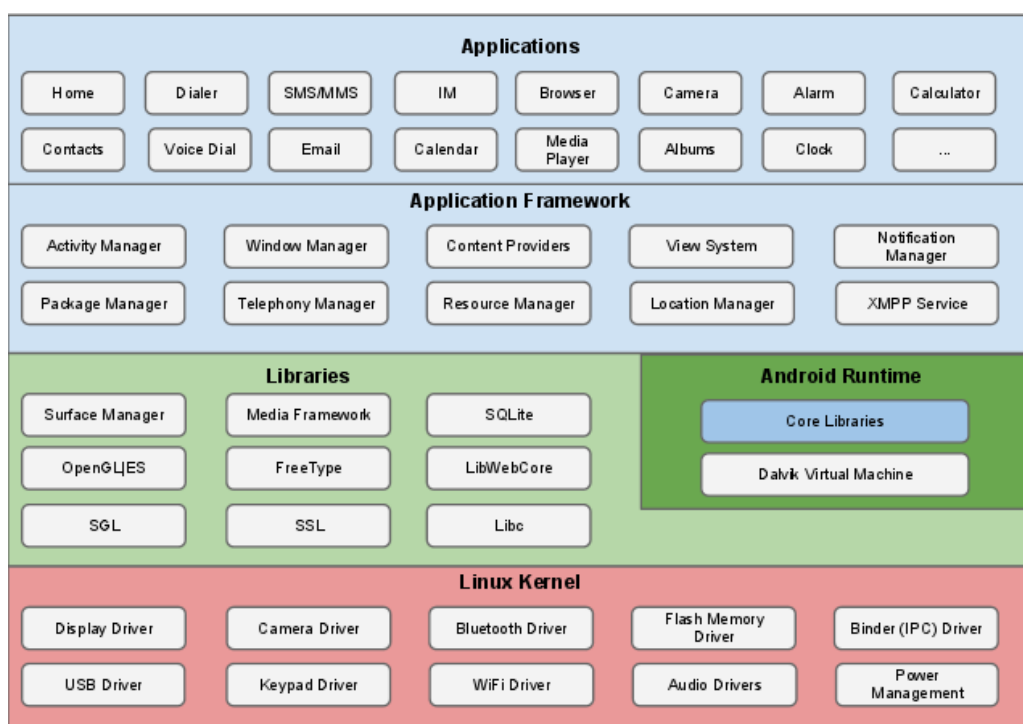
Zastoupení mobilních OS na trhu v třetím kvartále roku 2014 (Zdroj: [1])

3.2 Historie

Android je mobilní operační systém původně vyvíjený stejnojmennou firmou. V roce 2005 tuto nepříliš známou společnost koupil americký gigant Google Inc. V listopadu roku 2007 vzniklo konsorcium Open Handset Alliance. Toto uskupení zahrnuje přední výrobce mobilních telefonů, hardwaru a mobilních aplikací. Mezi členy patří kromě Google například NVIDIA, Qualcomm, Samsung nebo Intel. Konsorcium si kladlo za cíl vyvinout otevřený standard pro mobilní zařízení. Krátce po představení nového produktu s názvem Android byl pro vývojáře uvolněn první Android SDK licencovaný jako open-source. Necelý rok na to byl oficiálně uveden na trh první telefon s první verzí operačního systému. V současnosti je nejnovější verze 5.0 Lollipop. Ta oproti dřívějším nabízí zcela nové rychlejší Runtime ART a podporu 64 bitových aplikací [2].

3.3 Architektura

Architektura operačního systému Android se skládá z 5 vrstev. Tyto vrstvy primárně pracují samostatně, ale mohou zasahovat i do vrstev ostatních [4].



Obrázek 2:

Architektura systému Android (Zdroj: [4])

3.3.1 Linux Kernel

Nejnižší vrstva je tvořena jádrem operačního systému. Jedná se konkrétně o velmi upravenou verzi Linuxu 2.6. Ten zprostředkovává základní systémové funkcionality, jako jsou správa procesů a paměti. Dále také správu nad klávesnicí, displejem, fotoaparátem a dalšími zařízeními. Právě díky Linuxovému jádru zvládá operační systém Android velmi dobře síťování a podporu ovladačů nejrůznějších hardwarových zařízení [5].

3.3.2 Knihovny

Nad Linuxovým jádrem se nachází vrstva knihoven, které jsou napsané v jazyce C/C++. S těmi pracují různé komponenty systému. Ke knihovnám přistupují vývojáři skrze Android Application Framework. Pro ilustraci uvádím několik příkladů knihoven [6].

- OpenGL – 3D a 2D grafika
- SQLite – databáze pro ukládání a sdílení dat aplikací
- SSL – síťová bezpečnost
- WebKit – engine pro webový prohlížeč
- Media Framework – nahrávání a přehrávání zvuku a videa

3.3.3 Android Runtime

Třetí vrstvu architektury tvoří Android Runtime. Zde se nachází klíčová komponenta systému Dalvik Virtual Machine. Jak je již z názvu patrné, jedná se o virtuální stroj, který bychom mohli přirovnat k Java Virtual Machine. Tento virtuální stroj je ovšem speciálně navržený a optimalizovaný pro systém Android. Dalvik slouží ke spuštění aplikací v systému a využívá vlastností Linuxového jádra, jako jsou správa paměti nebo práce s vlákny. Navzdory tomu, že je Java volně šiřitelná, Java Virtual Machine není. Z tohoto důvodu, a také z důvodu optimalizace pro mobilní zařízení, se rozhodla společnost Google pro vývoj vlastního virtuálního stroje. Ten vytvořené *.class* soubory znovu zkompile do *.dex* (Dalvik Executable) souborů ([6], [7]).

Konečný, spustitelný kód tedy není Java bajtkód, ale jsou to *.dex* soubory, které se dále balí

do archivů *.apk*. V doposud poslední verzi Androidu (Android 5.0 Lollipop) byl Dalvik nahrazen novým virtuálním strojem Android Runtime (ART) [7].

3.3.4 Aplikační Framework

Čtvrtá vrstva poskytuje rozhraní API (Application Programming Interface) v různých oblastech, jako je síťování, multimédia, grafické uživatelské rozhraní, řízení spotřeby energie a přístup k úložišti. Knihovny, ze kterých je Framework tvořen jsou napsány v Javě a běží nad základními knihovnami Android Runtime. Aplikační Framework poskytuje různé správce pro různé účely, jako je řízení spotřeby energie, notifikace v rámci celého systému, správa oken atd. Aplikace by měly využívat služeb manažerů a nepoužívat základní knihovny přímo. To umožňuje správcům vyžadovat po aplikaci oprávnění. Tímto je předejito situaci, kdy aplikace nemá například povolení zahájit telefonní hovor nebo odeslat data po síti ([4], [6])

3.3.5 Aplikace

Poslední vrstvu tvoří samotné aplikace. Ty jsou buď nainstalované jako součást operačního systému nebo lze využít softwaru třetích stran. Pro tyto účely slouží Google Play (do češtiny překládána jako Google Obchod) - online distribuční služba, odkud je možné digitální obsah, buď zdarma, nebo za poplatek stáhnout [8].

4 Digitalizace audiosignálu

4.1 Teorie

Zvuk je obecně definován jako mechanické kmitání, které se šíří v látkovém prostředí všemi směry. Velmi důležitou vlastností je, že v určitém pásmu frekvencí dokáže vyvolat sluchový vjem. Zvuk v rozsahu této frekvence označujeme jako slyšitelný zvuk. Stanovení frekvenčního pásma slyšitelného zvuku pro lidské ucho je silně individuální. Jen málo lidí je schopno slyšet v celém rozsahu (problematická je především horní hranice, která je velmi proměnlivá a závislá například na věku). Obecně se ale jedná o interval přibližně mezi 16 Hz až 20 kHz. Jak již bylo zmíněno výše, zvuky mimo toto pásmo nejsou slyšitelné, přesto je lidské tělo dokáže vnímat. Ty mohou mít i nepříznivý dopad na fyzické zdraví člověka či jeho psychiku. Zvuk pod spodní hranicí slyšitelného spektra s velmi nízkou frekvencí (do 16 Hz) se označuje jako infrazvuk. Při větší intenzitě může u člověka vyvolat například závrať nebo i infarkt. Zvuk nad horní hranicí slyšitelného spektra (nad 20 kHz) se pak označuje jako ultrazvuk [9].

4.2 Analogový vs. digitální zvuk

Přibližně od sedmdesátých let minulého století existuje stále se omílající spor o to, jakým technologickým postupem zvuk zaznamenávat. Prvním z nich je použití klasičtějšího analogového přístupu, který s sebou ovšem přináší spoustu problémů, s nimiž je nutné se potýkat. Druhým je využití novějšího postupu digitalizace zvuku.

¹Jak již bylo zmíněno výše, zvuk je forma vlnění, která se dokáže šířit vzduchem. Při kontaktu s membránou mikrofону ji rozechvívá a dochází k pohybu cívky umístěné v magnetickém poli. Takto dochází k vytvoření elektrického signálu. Tento signál v podstatě představuje jakýsi věrný obraz původního zvuku. Jeho kvalita je ovlivněna pouze částmi audio řetězce a jejich nastaveními. Do této chvíle je postup nahrávání u obou přístupů shodný [10].

4.3 Analogový zvuk

Z důvodu dalšího zpracování je potřeba signál nějak zaznamenat. A zde se objevuje klíčový rozdíl. V analogovém světě se nabízí možnosti uložit signál například na magnetofonovou pásku či vinylovou desku.

V případě pásky je signál uložen do magnetické vrstvy. Při procesu přehrávání je pak magnetickou hlavou převeden zpět na elektrický signál. Ten je dále zesílen a za pomoci reproduktoru převeden na kinetickou energii, která rozechvívá vzduch.

U gramofonové desky je proces obdobný, až na to, že je záznam uložen jako zvlnění v drážce. Jehla gramofonu pak funguje na podobném principu jako výše zmíněný mikrofón. Při přehrávání se chvěje a pohybuje cívkou v magnetickém poli, která vytváří elektrický signál. Dále proces pokračuje stejně jako u magnetofonové pásky.

Pro práci s analogovým signálem se využívá elektronických zařízení, která ovlivňují signál o šum a zkreslení. Tyto dvě veličiny pak tvoří jakýsi přidaný charakteristický prvek velmi oblíbený mezi muzikanty. Přenáší se totiž nejen zvuková nahrávka, ale i informace o použité technice a tedy potažmo i době, kdy byla nahrána. Velké nevýhody analogového záznamu, jako jsou krátká životnost a komplikovaná editace, nakonec vedly k hledání nové technologie, kterou se stalo digitální audio ([10], [11], [12]).

¹ Pro jednodušší ilustraci principu je popisována situace, kdy je v řetězci umístěn dynamický mikrofón.

4.4 Digitální zvuk

Digitální audio je forma reprezentace zvuku za pomoci hodnot dvojkové soustavy. Pro nahrávání nebo poslech je nutné použít příslušný převodník (analogově digitální/digitálně analogový). Při procesu digitalizace zvuku dochází ke vzniku zkreslení a šumu jen v místech konverze, ať už z analogové formy do digitální nebo naopak [12].

Hlavní výhody digitálního audia jsou jeho snadná editace, přenositelnost a životnost, díky nimž se stalo velmi oblíbené v nahrávacích studiích. Právě snadná přenositelnost značně snížila distribuční náklady a umožnila vzniknout novým odvětvím, specializujícím se na distribuci hudby přes internet [12].

Princip digitalizace analogového signálu je popsán v další části kapitoly.

Na níže uvedeném obrázku č. 3 je zobrazeno srovnání analogového a digitálního zvuku.

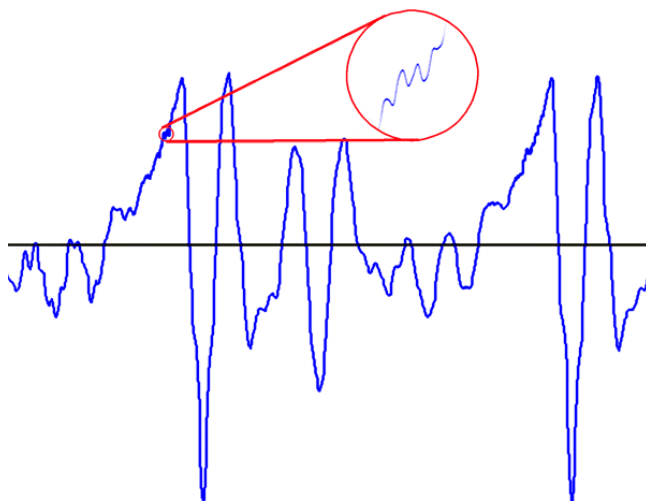
Analogový a digitální zvuk	
Analogový zvuk	Digitální zvuk
spojitý	nespojité
"nekonečné" rozlišení	konečný počet hodnot
elektrický signál	binární kód
při přebuzení postupný nárůst distorze	při přebuzení prudký nástup distorze
zkreslení bývá lineární	zkreslení bývá nelineární
subj. přirozený, "teplý"	subj. nepřirozený, sterilní
větší riziko šumu a ztráty kvality	větší odstup signálu od šumu
trpí při přehrávání a kopírování	kopírování bez ztráty kvality
omezené možnosti editace	široké možnosti editace a zobrazení
magnet. pásek, LP deska	CD, DVD, Blu-ray, HDD

Obrázek 3:

Srovnání analogového a digitálního zvuku (Zdroj: [33])

4.5 Proces digitalizace

Digitalizace audiosignálu představuje převedení nekonečného počtu hodnot spojitého analogového signálu na konečný počet hodnot diskrétního digitálního signálu. Jelikož se jedná o nelineární proces, tak není možné zpětnou konverzí získat původní signál [14]. Obrázek č. 4 níže zobrazuje detail spojitého signálu.



Obrázek 4:
Detail spojitého signálu (Zdroj: [13])

4.5.1 Pulzně kódová modulace

V roce 1937 přišel brit Alec Reeves s principem pulzně kódové modulace (PCM), která umožňuje převod spojitého analogového signálu na diskrétní digitální. Později se z této metody stal nejpoužívanější typ modulace v oblasti digitální audio techniky, převážně díky jednoduchému principu, na kterém je postavena. Dalším důvodem se stal fakt, že se ji společnost Sony na začátku osmdesátých let rozhodla použít pro svůj tehdy nový typ média Audio CD, který se brzy stal nejrozšířenějším záznamovým médiem. PCM je tak v dnešní době multiplatformním standardem v oblasti digitálního záznamu [15].

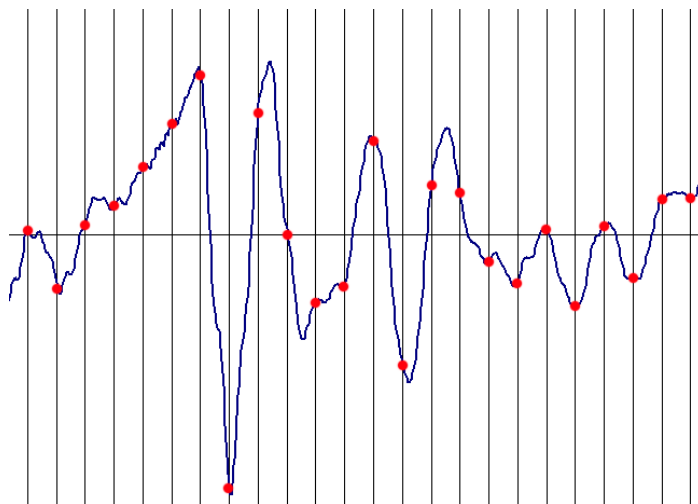
Některé třídy systému Android určené ať už pro zaznamenávání či přehrávání zvuku pracují s touto modulací. Proto je následující část práce věnována popisu jednotlivých kroků.

4.5.2 Vzorkování

První část digitalizace tvoří proces vzorkování signálu. Jak již bylo zmíněno výše, analogový signál se skládá z nekonečného počtu hodnot, a proto zde nastává problém. Jelikož jsme v digitálním světě omezeni velikostí paměti a výpočetním výkonem, je nutné omezit i počet vzorků, které za časovou jednotku uděláme. Prvním parametrem je tedy vzorkovací frekvence.

Samotný proces, zachycený na obrázku č. 5, probíhá tak, že se v první fázi rozdělí vodorovná osa signálu dle vzorkovací frekvence (počet vzorků za sekundu) na rovnoměrné úseky. Následně se z každého úseku odebere jeden vzorek. Výsledek tvoří množina hodnot získaných v pevně stanovených časových intervalech. Jelikož původní spojitý signál je tvořen nekonečným počtem hodnot, je patrné, že vzorkovaný signál je oproti původnímu ochuzen o spoustu detailů [18].

Čím vyšší je vzorkovací frekvence, tím se digitalizovaný signál méně odlišuje od původního analogového. S tím je ovšem spojen i problém množství dat, které jsou zaznamenávány. Vyšší počet vzorků znamená i vyšší nároky na výpočetní výkon a paměť.[18]



Obrázek 5:

Proces vzorkování signálu (Zdroj: [13])

Je-li vzorkovací frekvence příliš nízká, dochází ke zkreslení digitalizovaného signálu a ztrátě kvality. Této problematice se věnuje Nyquistův (Shanonův, Kotělnikovův) teorém popsany níže.[16]

Nyquistův (Shannonův, Kotělnikovův) teorém

$$^2 \quad F_{vz} \geq 2F_{max}$$

Jedná se o vztah mezi vzorkovací frekvencí a nejvyšší amplitudou digitalizovaného signálu. Teorém říká, že pro dokonalou rekonstrukci signálu je potřeba minimálně dvakrát vyšší vzorkovací frekvence, než je maximální frekvence digitalizovaného signálu. Jinými slovy, je nutné, aby polovina vzorkovací frekvence (tzv. Nyquistova frekvence) byla vyšší než je nejvyšší frekvence vzorkovaného signálu [19].

Při nedodržení tohoto předpokladu dochází k jevu, který se nazývá Aliasing (česky: falšování). V důsledku toho se mohou ve výsledném signálu objevit frekvence, které se v původním signálu nenacházely [16].

Pokud je tedy maximální frekvence signálu například 150Hz, je třeba použít vzorkovací frekvenci nejméně 300Hz, aby se eliminovalo riziko Aliasingu.

V praxi je ovšem nutné vzniku Aliasingu předcházet, proto se ve většině případů před převodník analogového signálu na digitální zařazuje tzv. antialiasingový filtr, který dokáže ze spektra odfiltrovat frekvence, tak aby byla splněna podmínka pro Nyquistův (Shannonův, Kotělnikovův) teorém [18].

Jelikož je rozsah lidského sluchu často udáván od 16Hz až 20kHz, je minimální vzorkovací frekvence splňující teorém 40kHz. Z tohoto důvodu byla jako standard pro zaznamenávání zvuku na CD stanovena vzorkovací frekvence, doplněná o rezervu, na 44,1 kHz. Je tak možno zaznamenat slyšitelné spektrum v celé šíři [20]. Tabulka 1 umístěná níže uvádí hodnoty nejpoužívanějších vzorkovacích frekvencí.

² Vzorec popisující Nyquistův teorém

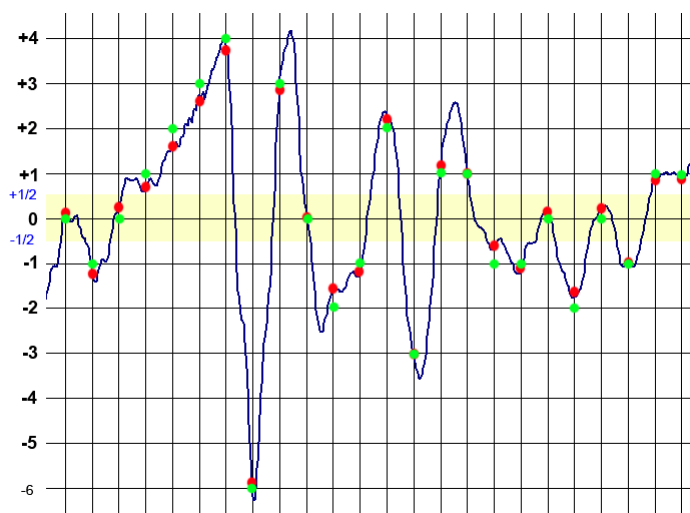
Tabulka 1: Příklady nejpoužívanějších frekvencí, zdroj[34]

Frekvence	Popis
8.000 Hz	vzorkovací frekvence používaná v klasické digitální telefonii. Vychází z toho, že těžiště informací v lidském hlase se nachází ve spektru do 4 kHz, dnešní komunikační formáty (např. VOIP) používají i vyšší v závislosti na použitém kodeku.
44.100 Hz	nejběžnější hodnota využívaná v CD Digital Audio a u většiny hudby uložené ve formátu MP3
48.000 Hz	vzorkovací frekvence využívaná pro záznam zvuku do video hardwaru, používá se také v digitální TV, DVD i v moderních video rozhraních jako je SDI.
96.000 Hz	vzorkování využívané v DVD Audio, Bluray a HD DVD, běžně se využívá i při nahrávání.
192.000 Hz	opět formáty s vysokým rozlišením, často nejvyšší frekvence, na které je ještě schopna pracovat většina profesionálního audio hardwaru a softwaru.

4.5.3 Kvantizace

Druhou část digitalizace tvoří proces kvantizace. Zde je vzorkovanému signálu přiřazena odpovídající hodnota na svislé ose. Osa amplitudy vstupního signálu je rozdělena na jednotlivé hladiny. Tyto hladiny se označují jako kvantizační úrovně a udávají, o kolik se mohou minimálně lišit amplitudy dvou vzorků signálu. Jelikož je velmi pravděpodobné, že vzorek nebude ležet přesně na kvantizační úrovni, rozděluje se prostor kolem jednotlivých úrovní na tzv. pásy tolerance. Ty určují, ke které úrovni daný vzorek náleží. [18]

V této části nastává největší zkreslení hodnot od původního signálu. Díky pásům tolerance kvantované vzorky přestávají odpovídat původním hodnotám signálu, což má za následek vznik tzv. kvantizační chyby. Tato veličina udává, jaká je vzdálenost mezi kvantovanými a původními vzorkovanými hodnotami. Pro potlačení kvantizační chyby je možné řešení zvýšit počet kvantizačních úrovní [18]. Výsledek procesu kvantizace signálu je možné vidět na obrázku č. 6 níže.



Obrázek 6:
Proces kvantizace signálu (Zdroj: [14])

4.5.4 Kódování

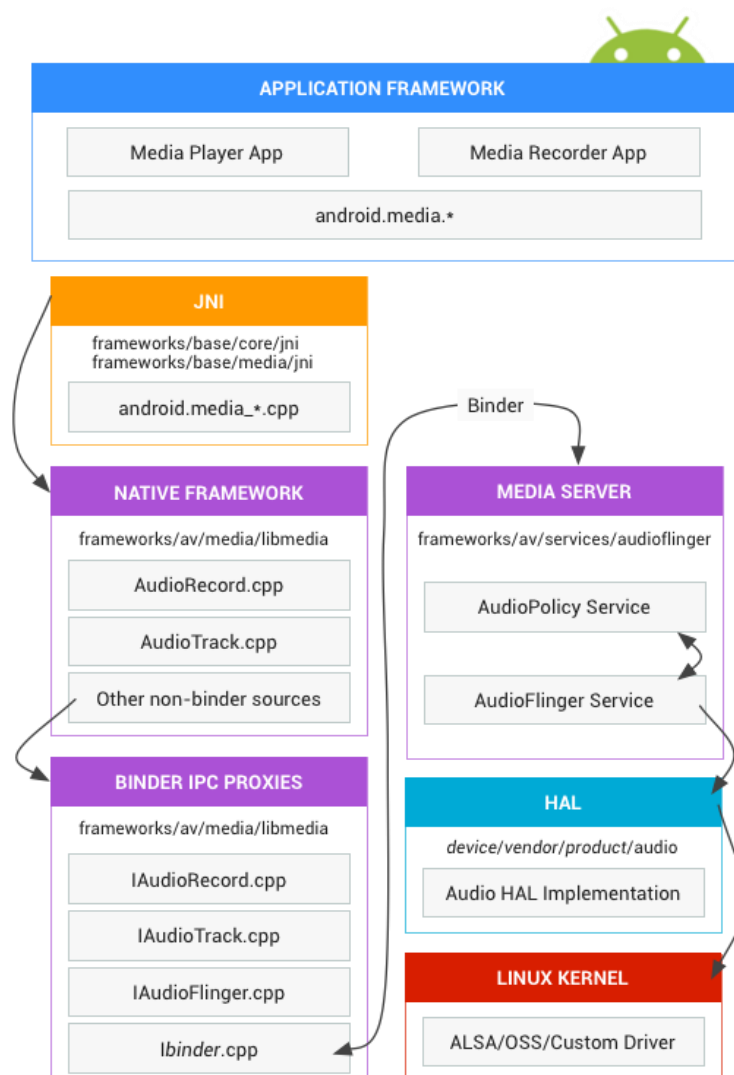
Jelikož je digitální signál zpracováván zařízeními pracujícími ve dvojkové soustavě, je nutné v poslední části digitalizace přiřadit kvantizačním úrovním odpovídající n -bitové hodnoty v binárním tvaru. V dnešní době se jako standard označuje 16 bitové kódování, kterému odpovídá 65536 úrovní. Zde se je hodnota kvantizačního zkreslení velmi malá, nicméně ve studiové úrovni se používá 24 bitové kódování, které ho ještě více eliminuje [18].

5 Android a zvuk

Jak již bylo zmíněno ve třetí kapitole, pro nahrávání, zpracovávání a přehrávání zvuku se v systému Android používá část Media Framework. Její hlavní funkcí je poskytnout rozhraní pro přístup ke knihovnám Media Libraries, které jsou napsány v jazyce C/C++ [21].

Media Framework tvoří několik dynamických knihoven, jako jsou libmediajni, libmedia, libmediaplayservice [21]. Samotnou komunikaci mezi aplikací a hardwarem zobrazuje obrázek č. 7 a probíhá následovně:

1. Aplikace, vytvořená za pomoci tříd z balíčku *android.media* přistupuje k nativním knihovnám libmedia skrze JNI (Java Native Interface).
2. Dalším článkem v řetězci je Media Server, který poskytuje patřičný proces pro interakci s hardwarem. Libmedia v tomto případě figuruje jako klient komunikující s Media Serverem za pomoci mechanismu Binder IPC (Inter Process Communication).
3. V situaci, kdy je potřeba přistupovat ke zvukovému hardwaru se využívá proces AudioFlinger. Jeho hlavní funkcí je správa vstupních/výstupních zvukových streamů a vláken pro nahrávání a přehrávání.
4. AudioFlinger předává data do tzv. HAL (Hardware Abstraction Layer). Díky této vrstvě je umožněna interakce se zvukovým hardwarem a příslušným ovladačem [21].



Obrázek 7:
MediaFramework (Zdroj: [21])

Následuje popis vybraných tříd z balíčku *android.media*:

5.1 MediaPlayer

MediaPlayer je asi nejpoužívanější třídou pro přehrávání zvuku v systému Android. Její součástí je obsáhlá oficiální dokumentace. Další výhodou je, že její použití bývá velmi často ilustrováno v příkladech na internetu a lze je tak snadno pochopit.

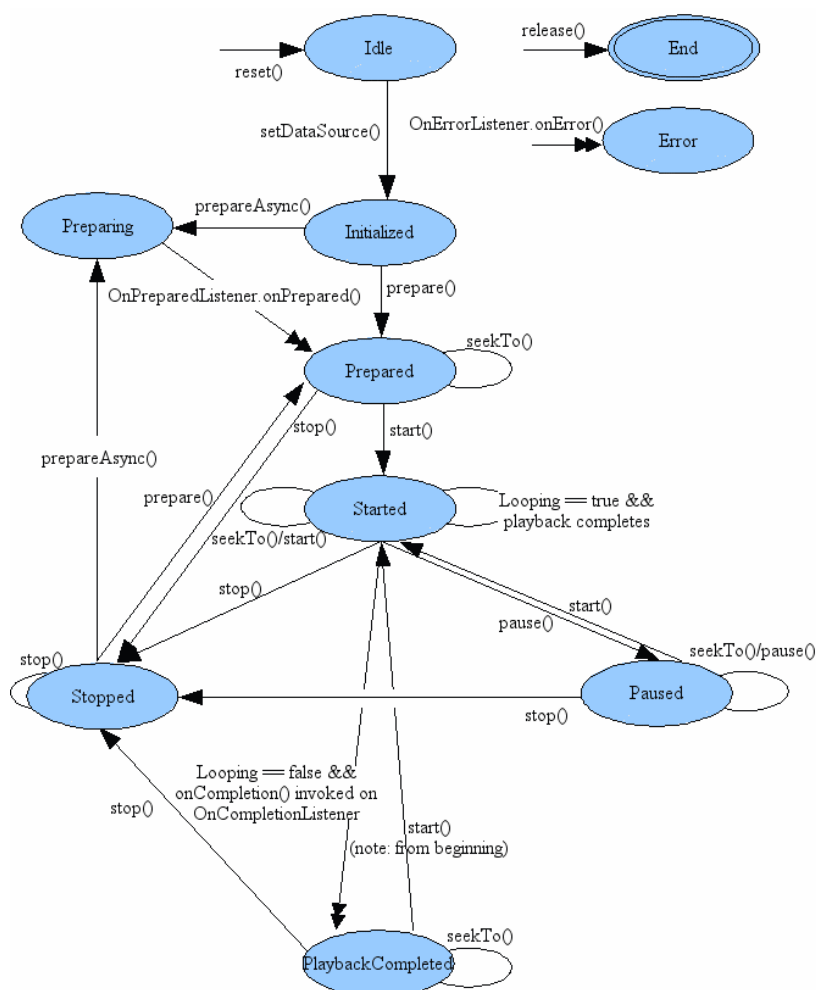
MediaPlayer neslouží jen k přehrávání zvuku, ale také videa. Je koncipován jako API (Application Programming Interface) pro přehrávání médií z různých zdrojů. To mohou být například média, které jsou součástí aplikace nebo uložená na SD kartě. Třída *MediaPlayer* je velmi často využívána pro streamování médií z webu, například pro přehrávání souborů MP3 umístěných na serveru http ([22], [25]).

Rozhraní je jasně určeno pro streamování delších médií, jakou jsou nahrávky hlasu, hudby a videa. Můžeme tak zde nalézt funkce pro posouvání do konkrétní části streamu nebo také bufferování zdrojového souboru po částech ([22], [25]).

Velkou nevýhodou je ovšem rychlost inicializace. Díky dlouhé prodlevě (např. doba mezi stisknutím tlačítka a přehráním média) se třída *MediaPlayer* téměř nedá použít v případech, kdy se klade důraz na včasné přehrávání zvuku – například v herních aplikacích. Naproti tomu je její použití vhodné pro přehrávače hudby nebo videa ([22], [25]).

Další nevýhodou je velká náročnost na operační paměť. Při spuštění více instancí zároveň je velmi pravděpodobné, že aplikace skončí chybou [25].

Obrázek č. 8 uvádím jako ukázkou životního cyklu instance třídy *MediaPlayer*.



Obrázek 8:
Stavový diagram třídy *MediaPlayer* (Zdroj: [22])

5.2 SoundPool

Třída *SoundPool* je opakem třídy *MediaPlayer*. Je určena především pro rychlé přehrávání předem definovaného setu relativně krátkých audionahrávek [25].

Nejčastěji se používá v herních aplikacích jako *pool* pro zvukové efekty. Hra obvykle obsahuje set zvukových audionahrávek, které je potřeba přehrávat znovu a znovu. Často se může jednat o i více nahrávek ve stejnou chvíli [25].

V takovém případě se klade důraz na to, aby prodleva mezi akcí a následným přehráním zvuku byla co nejkratší. Jako příklad může sloužit situace, kdy uživatel ve hře vystřelí ze zbraně [25].

Dalším příkladem může být použití třídy *SoundPool* v mobilní aplikaci pro instant-messaging. Je běžné, že aplikace tohoto typu používají krátká zvuková upozornění pro novou příchozí zprávu. Pokud bychom použili třídu *MediaPlayer* místo *SoundPool*, vystavujeme se riziku dlouhé prodlevy, což by mělo pravděpodobně za následek snížení uživatelské přívětivosti ([23], [25]).

Na rozdíl od užití třídy *MediaPlayer* se třída *SoundPool* používá v situacích, kdy jsou předem známy zvukové zdroje a je tak možné je načíst ještě před plným spuštěním aplikace [25].

Kromě základních operací, jako je načtení a přehrání zvuku, umožňuje třída *SoundPool* ještě:

- nastavení maximálního počtu současně přehrávaných zvuků,
- nastavení priority přehrávání podle *thresholdu*,
- pozastavení a zastavení přehrávání před koncem stopy,
- smyčkování,
- změnu rychlosti přehrávání ³,
- ovládání oddělených stereo korekcí hlasitosti pro každý kanál.

5.3 AudioTrack

Třídy *MediaPlayer* nebo *SoundPool* pokryjí většinu případů užití zpracovávání zvukových nahrávek v systému Android. Nicméně mohou nastat situace, kdy je vyžadována práce s čistými daty a komunikace se zvukovým hardwarem. V těchto případech se používá třída *AudioTrack* [25].

Ta tvoří nejnižší API pro práci se zvukem v systému Android. Jedná se o velmi komplexní a složité rozhraní. Pro práci je nutná alespoň základní znalost problematiky digitalizace zvukového signálu. Jedna instance třídy *AudioTrack* představuje jeden zdroj. Při vytváření je

³ Tato funkcionalita se využívá pro dosažení účinku podobnému Dopplerovu efektu (změna výšky tónu)

nutné definovat parametry, jako jsou:

- `streamType` – typ audio streamu (`STREAM_VOICE_CALL`, `STREAM_SYSTEM`, `STREAM_RING`, `STREAM_MUSIC`, atd.),
- `sampleRateInHz` – velikost vzorkovací frekvence zdrojového souboru,
- `channelConfig` – nastavení počtu kanálů (mono nebo stereo),
- `audioFormat` – typ kódování dat,
- `bufferSizeInBytes` – velikost *bufferu*,
- `mode` – streaming nebo static [24].

Pro zapisování dat se používá metoda `write()`, do které se jako jeden z parametrů vkládá jednorozměrné pole, obsahující čistá audio data. Toto pole může být typu `byte`, `short` nebo `float`. Díky tomu je možné dekódování zvuku z libovolného formátu, který nemusí být běžně v systému podporován [24].

Instance třídy *AudioTrack* může fungovat ve dvou módech: streaming nebo static.

V prvním případě jsou čistá audio data zapisována jako plynulý stream, za pomoci metody `write()`. Ta vrátí návratovou hodnotu, kterou je počet zapsaných bajtů, za předpokladu, že jsou data převedena z Javy do nativní vrstvy (C/C++) a zařazena do fronty pro přehrávání [24].

Streaming mód se používá při přehrávání zvuků, které jsou:

- příliš dlouhé, aby se vešly do paměti,
- příliš velké například z důsledku použité vysoké vzorkovací frekvence nebo kódování,
- načtené ze zdroje nebo vytvořené, zatímco probíhá přehrávání [25].

Static mód je funkčně podobný třídě *SoundPool* zmíněné výše. Najde uplatnění v případech, kdy je potřeba přehrát krátké zvukové vzorky s co nejkratší odezvou, jako jsou například herní aplikace [25].

Po vytvoření si instance inicializuje audio *buffer*. Jeho velikost je daná dle parametru uvedeného v konstruktoru a udává, kolik dat je možno přehrát, než je *buffer* vyprázdněn [24].

V případě static módu se pak jedná o velikost zvuku, který může být přehrán. Ve streaming módu to znamená, že data jsou předávána do nativní vrstvy po částech menších nebo stejně velkých, jako je velikost *bufferu* [25].

5.4 AudioRecord

Třída *AudioRecord* slouží pro nahrávání zvuku za pomoci vstupního hardwaru (nejčastěji mikrofon). Její hlavní funkcí je správa zvukových zdrojů. Pro zaznamenávání dat se používá metoda *read()*, kterou lze použít v závislosti na datovém typu, ve kterém chceme data uchovávat. Při vytváření instance je potřeba definovat následující parametry:

- *audioSource* – zde je nutné nastavit zdroj, odkud budou data načítána. Kromě mikrofonu je také možné nastavit například nahrávání hlasu volajícího nebo volaného účastníka,
- *sampleRateInHz* - velikost vzorkovací frekvence zdroje,
- *channelConfig* – nastavení počtu kanálu (mono nebo stereo),
- *audioFormat* – typ kódování dat,
- *bufferSizeInBytes* – velikost bufferu [26].

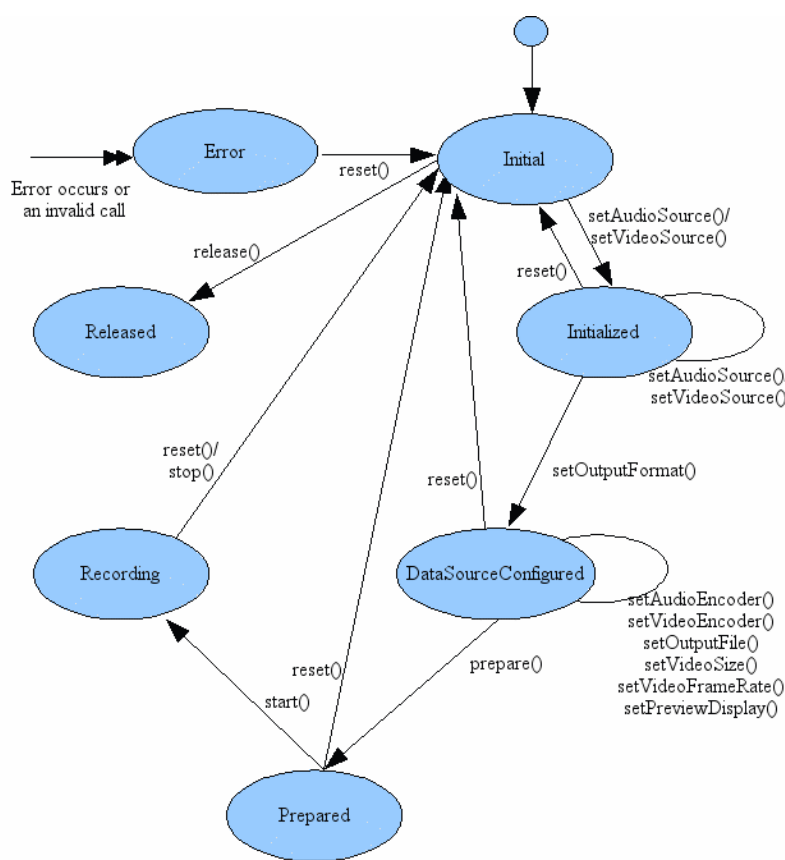
Obdobně jako u třídy *AudioTrack* si po vytvoření instance inicializuje audio *buffer*, do kterého jsou zaznamenávaná data průběžně ukládána v blocích. Jeho velikost je daná dle parametru uvedeného v konstruktoru (při vytváření instance) a udává, kolik dat je možné přehrát, než dojde k vyprázdnění *bufferu* [26].

5.5 MediaRecorder

MediaRecorder je asi nejpoužívanější třídou pro nahrávání zvuku a videa v systému Android. Přestože nabízí poměrně jednoduché a intuitivní API, je kladen velký důraz na volání metod ve správném pořadí, aby se předcházelo chybě *IllegalStateException* [27].

Na rozdíl od výše zmíněných tříd pro zpracování zvuku neumožňuje třída *MediaRecorder* ukládat data průběžně. Po vytvoření instance je nutné pomocí *setterů* nastavit hlavní parametry, jako jsou zdroj signálu, dekodér, název výstupního souboru a výstupní formát, ve kterém se bude ukládat [27].

Třída *MediaRecorder* se tak nedá použít v případech, kdy je potřeba zaznamenávaná data průběžně zpracovávat. Zvuk je nutné nejprve uložit do souboru a až po té je s ním možné manipulovat. V tomto momentu se již nejedná o čistá data, ale o soubor s daným formátem a s ním spojenou i patřičnou kompresi, která způsobuje zkreslení [27]. Obrázek č. 9 níže uvádím pro ilustraci životního cyklu třídy *MediaRecorder*.



MediaRecorder state diagram

Obrázek 9:

Stavový diagram třídy *MediaRecorder* (Zdroj: [27])

6 Praktická část - vývoj aplikace

Jak již bylo zmíněno v úvodu, cílem této práce je vyvinout funkční aplikaci na platformu Android, která bude sloužit jako cvičební pomůcka pro začínající i pokročilé muzikanty. V následující části práce se zabývám analýzou požadavků, návrhem řešení, implementací funkcionalit a popisem aplikace. Posledním krokem je publikace na Google Play.

6.1.1 Analýza a požadavky

Z důvodu podpory různých druhů cvičení pro muzikanty jsem se rozhodl, že aplikace bude sestávat ze tří komponent. První z nich bude tvořit metronom, který slouží k procvičování hry podle stejného tempa. Další částí bude ladička, jejíž hlavní funkcí bude zobrazení frekvence, názvu tónu a případně odchylky od něj. Díky ní bude možno provádět přesné ladění hudebního nástroje. Poslední komponentou bude EarTrainer pro procvičování hudebního sluchu za pomoci hádání intervalů mezi tóny.

Grafika aplikace bude laděna do jednotných barev a při jejím zpracování bude kladen důraz hlavně na funkčnost a její intuitivní použití. Hlavní okno aplikace bude sloužit jen jako rozhraní pro přístup k jednotlivým komponentám. Pro usnadnění použití bude v každém okně k dispozici nápověda. Jelikož se očekává, že po dokončení bude aplikace přidána na Google Play ke stažení, bude navrženo logo, které bude sloužit jako identifikační prvek.

6.1.2 Rešerše aplikací ve světě

Ještě před začátkem vývoje byla provedena rešerše obdobných dostupných aplikací.

Sada hudební

Dostupné z: <https://play.google.com/store/apps/details?id=com.netigen.bestmusicset>

Tato aplikace vytvořená polským vývojovým týmem Netigen obsahuje tři komponenty: metronom, ladičku a simulaci klaviatury včetně zvuků. Při testování jsem objevil chybu ve výpočtu frekvence u ladičky. Dokáže sice správně určit, o jaký tón se jedná, ale hodnota

frekvence, která je zobrazena, je vždy poloviční oproti realitě. Velmi rušivý prvek tvoří vyskakující reklamy a jiná upozornění ve formě animace. Díky tomu nebyl chod aplikace plynulý.

Guitar Tools

Dostupné z: <https://play.google.com/store/apps/details?id=com.crnibero.guitartools>

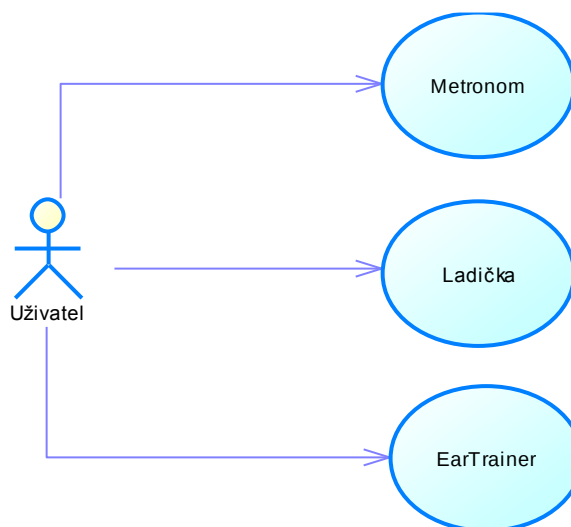
Aplikace od vývojáře Crni Bero v sobě spojuje seznam akordů a úderových technik při hře na kytaru. Dále se zde také nachází ladička a metronom. V průběhu testování jsem narazil na spoustu chyb. Seznam akordů nelze zobrazit. Stejně tomu tak je i u seznamu úderových technik. Metronom obsahuje pouze dva ovládací prvky: zapnutí/vypnutí metronomu a snižování/zvyšování tempa za pomoci tlačítek. Nic víc v uživatelském grafickém rozhraní metronomu není. Poslední částí je ladička, která není chromatická, ale slouží pouze pro přehrání tónů jednotlivých strun akustické kytary.

Tuner a Metronom

Dostupné z: <https://play.google.com/store/apps/details?id=com.soundcorset.client.android>

Zde se vývojářská společnost onsquare pokusila spojit dohromady ladičku, metronom a rekordér. Ladička ladí poměrně přesně. Odchylka měření se při testování pohybovala kolem 1 Hz od reálné hodnoty. Rekordér fungoval bezchybně. Velmi dobře je zpracovaná komponenta metronom, která nabízí široké možnosti nastavení, včetně různých druhů rytmu podle hudebního stylu. Nicméně bych aplikaci vytkl fakt, že při instalaci požaduje přidělení velkého množství práv, jako je například přístup k fotoaparátu, fotkám a médiím či k telefonnímu číslu a ID zařízení.

6.1.3 Use case diagram

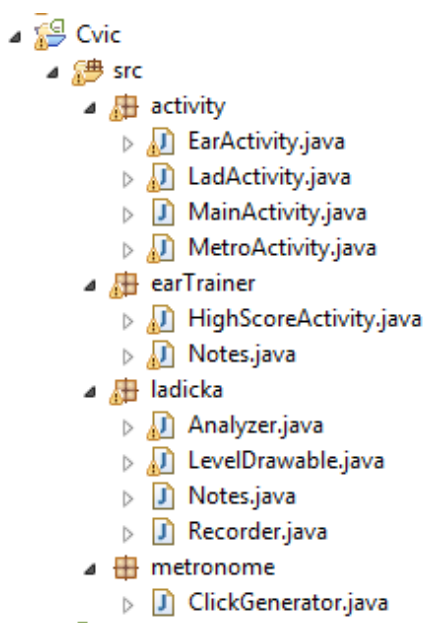


Obrázek 10:

Use case diagram aplikace (Zdroj: autor)

6.1.4 Struktura projektu

Jak je vidět na obrázku č. 10, projekt je rozdělen do jednotlivých balíčků, tak aby byla oddělena logická část od grafiky. Pro každou komponentu je zvlášť vytvořen balíček. Obrázky, zvuky a rozvržení obrazovek se pak nachází ve složce `.res`.



Obrázek 11:

Struktura projektu v prostředí Eclipse IDE (Zdroj: autor)

AndroidManifest.xml

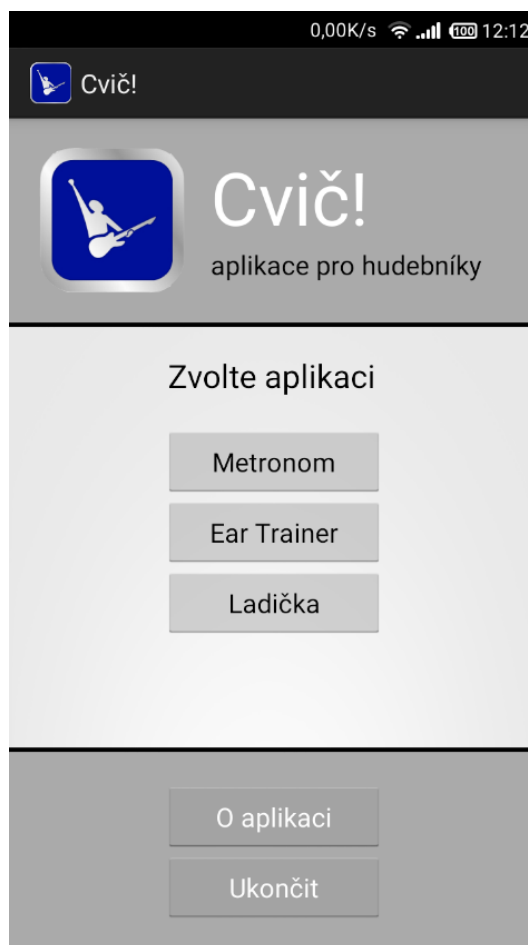
Důležitou část projektu tvoří tzv. manifest. Jedná se o soubor ve formátu XML, který obsahuje veškerou důležitou konfiguraci spojenou s aplikací. Jak je vidět na výpisu kódu č. 1 níže, pro správnou funkci aplikace je nutné ji přidělit práva pro přístup k mikrofonu. Dále je zde nadefinováno minimální požadované API pro spuštění aplikace, které má hodnotu 14 (verze systému Android 4.0 ICE_CREAM_SANDWICH). Aby bylo předejito nestabilitám aplikace při zobrazení na malých obrazovkách, je nastavena minimální hodnota 240 dpi (dots per inch) displeje. Manifest používá mimo jiné služba Google Play, která podle něj určuje, jakým zařízením umožní stažení aplikace.

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <manifest xmlns:android="http://schemas.android.com/apk/res/android"
3     package="com.mn.cvic"
4     android:versionCode="5"
5     android:versionName="1.5" >
6
7     <uses-sdk
8         android:minSdkVersion="14"
9         android:targetSdkVersion="19" />
10
11     <compatible-screens>
12         <screen android:screenSize="small" android:screenDensity="hdpi" />
13         <screen android:screenSize="small" android:screenDensity="xhdpi" />
14
15         <screen android:screenSize="normal" android:screenDensity="hdpi" />
16         <screen android:screenSize="normal" android:screenDensity="xhdpi" />
17
18         <screen android:screenSize="large" android:screenDensity="hdpi" />
19         <screen android:screenSize="large" android:screenDensity="xhdpi" />
20
21         <screen android:screenSize="xlarge" android:screenDensity="hdpi" />
22         <screen android:screenSize="xlarge" android:screenDensity="xhdpi" />
23     </compatible-screens>
24
25     <uses-permission android:name="android.permission.RECORD_AUDIO" />
26
```

Výpis kódu 1:

Ukázka manifestu (Zdroj: autor)

6.1.5 Obrazovka základního menu



Obrázek 12:
Základní menu aplikace (Zdroj: autor)

6.2 Komponenta metronom

Jak již bylo zmíněno v úvodu práce, velkým problémem, se kterým se musí začínající muzikant potýkat, je hraní ve správném (stejném) tempu. Tuto dovednost je důležité si osvojit již v začátcích učení, jelikož se chybné prvky hry ve smyslu zrychlování/zpomalování později těžce přeučují.

Pro procvičování a trénování hry ve správném tempu se používá metronom. Jedná se o zařízení, vydávající krátký cvakavý zvuk, který udává přesné tempo.

6.2.1 Analýza a požadavky

Hlavní doménou metronomu je jeho přesnost, proto bude při vývoji kladen důraz převážně na tuto vlastnost. Jakékoliv zpoždění či odchylka od tempa by z něj v praxi činila nepoužitelnou část aplikace. Dalším požadavkem je dostatečný rozsah tempa. Z důvodu pokrytí co nejvyššího počtu různých hudebních stylů je potřeba, aby bylo možno dosáhnout hranice 200 úderů za minutu. Pro větší uživatelskou přívětivost bude v grafickém uživatelském rozhraní vytvořeno několik ovládacích prvků, které budou práci s metronomem usnadňovat. Je nutné, aby jakékoliv vstupy od uživatele byly ošetřeny a bylo předejito případným nestabilitám aplikace. Poslední požadavek je kladen na ukazatel úderů. Ten bude sloužit pro větší přehlednost a snazší orientaci při hře.

6.2.2 Příklad užití

Komponenta bude sloužit jako náhrada za klasický metronom. Od toho se bude odvíjet i jeho použití. Hráč nastaví požadované tempo a spustí přehrávání zvuku. Přes ovládací prvky bude moci korigovat frekvenci kliků.

6.2.3 Problémy při realizaci

Během vývoje a testování komponenty jsem narazil na následující problémy:

Generovaný zvuk

Jelikož je hráč při hraní s metronomem nucen poslouchat opakování stejného tónu, bylo potřeba najít spíše zvuk obsahující vyšší harmonické frekvence než generovaný tón. Delší použití metronomu, přehrávajícího generovanou sinusoidu o určité frekvenci, bylo při testování aplikace nepříjemné.

Časová nespolehlivost vláken

První verze komponenty byla založena na principu přehrávání zvuku za pomoci vlákna z rozhraní *Runnable*. Nastavená frekvence metronomu byla pak předávána jako parametr do metody *sleep()*, která vlákno uspala na určitou dobu mezi přehráním jednotlivých zvuků. To se

ovšem brzy ukázalo jako špatné řešení, jelikož vlákno sdílí paměť s ostatními vlákny v procesu [28]. Proto docházelo k různým časovým prodlevám mezi přehráním zvuků a metronom se zpožďoval nebo zrychloval. Byť se jednalo o odchylky v řádech milisekund, v praxi se komponenta nedala použít.

Stejný problém se objevil při pokusu o implementaci animovaného ukazatele úderů. Cílem bylo napodobit pohyb rafičky klasického metronomu, která se pohybuje ze strany na stranu v závislosti na frekvenci úderů. K tomuto účelu byla použita třída *Animation* z balíčku *android.view.animation*, která poskytuje rozsáhlé možnosti nastavení, jako například dobu trvání animace nebo počet opakování. Bohužel, jak už vyplývá z oficiální dokumentace, metoda *setDuration()*, která slouží k nastavení doby trvání, nastavuje pouze skutečnost, jak dlouho by měla animace trvat. V důsledku toho problému docházelo opět ke zpožděním v délkách milisekund, díky čemuž se nedalo ukazatelem řídit.

Tap tempo

Mezi ovládací prvky byla přidána funkcionalita ve formě tzv. tap tempa. Uživatel si tak může nastavit tempo skladby za pomoci opakovaného stisknutí tlačítka. Frekvence metronomu pak odpovídá frekvenci kliknutí.

6.2.4 Popis

Komponenta se skládá ze dvou tříd.

Třída MetroActivity

Potomek třídy *Activity* představuje viditelnou část komponenty. Jelikož se jedná o tzv. „aktivitu“, slouží tato třída jako grafické uživatelské rozhraní metronomu s podporou jeho ovládacích prvků a ukazatelem aktuálního úderu. Pro tyto prvky se zde nachází definované handlers, které určují jejich funkce. Dále jsou zde metody pro kontrolu vstupu od uživatele, které zamezují například překročení minimální hodnoty frekvence úderů. Pro animaci ukazatele je nutné předat zprávu o aktuálním úderu ze třídy *ClickGenerator*. Podle něj se grafický prvek mění.

Po vytvoření instance aktivity dojde k načtení zvukového souboru ze složky *.raw* za pomoci metody *initClick()*. Jelikož se pro přehrávání zvuku používá třída *AudioTrack*, je nutné vložit

data do pole typu *Byte*, které bude později předáno třídě *ClickGenerator*. Ve výpisu kódu č. 2 se metoda *initClick()* nachází.

```
public void initClick() throws IOException {  
  
    InputStream raw = getResources().openRawResource(R.raw.click);  
    clicksArray = new byte[CLICK_BUFFER_SIZE];  
    indexOfClick = 0;  
    for(int i = 0; i < clicksArray.length; i++) {  
  
        int y = raw.read();  
        //y==-1 když je načten poslední bajt  
        if (y == -1){  
            break;  
        }  
        clicksArray[indexOfClick++] = (byte)(y);  
    }  
}
```

Výpis kódu 2:

Metoda initClick() (Zdroj: autor)

Jak již bylo zmíněno výše, řešení přehrávání zvuků za pomoci vláken nebylo vhodné, proto byla vytvořena vnořená privátní třída *AsyncClick*, která je potomkem třídy *AsyncTask*. Jedná se o třídu, která provádí úlohy ve vláknech uživatelského rozhraní z vlákna běžícího na pozadí. Jak je vidět na obrázku č. x, při vytvoření instance třídy *AsyncClick* je současně vytvořena i instance třídy *ClickGenerator*, ve které je následně inicializován *AudioTrack* podle příslušných parametrů (vzorkovací frekvence, počet kanálů, druh kódování, velikost *bufferu* atd.) Při zavolání metody *execute()*, která slouží pro spuštění *AsyncTasku*, se pak volají metody instance třídy *ClickGenerator*. Konkrétně se jedná o *setter*, nastavující tempo a metodu pro přehrávání zvuku *play()*.

Pro podporu funkcionality tap tempa byla vytvořena vnořená privátní třída implementující rozhraní *OnClickListener*. Při opakovaném stisknutí tlačítka se ukládá systémový čas do proměnných. V dalším kroku se hodnoty obou proměnných odečtou. Následuje další výpočet, zaokrouhlení a kontrola, zda výsledné tempo nepřekračuje limit 200 úderů za minutu. V posledním kroku se za pomoci *setteru* v *AsyncTasku* změní tempo. Ve výpisu kódu č. 3 je třída *ClickAsync* obsažena.

```
private class ClickAsync extends AsyncTask<Void, Double, Void>{

    ClickGenerator clickGen;

    ClickAsync() {
        clickGen = new ClickGenerator(clicksArray, indexOfClick, handler);
        clickGen.initTrack();
    }
    @Override
    protected Void doInBackground(Void... params) {

        clickGen.setTempo(Hz);
        clickGen.play();

        return null;
    }

    public void setTempo(double t){
        clickGen.setTempo(t);
    }

    public void stop() {
        clickGen.stop();
        clickGen = null;
    }
}
```

Výpis kódu 3:

Třída ClickAsync (Zdroj: autor)

Třída ClickGenerator

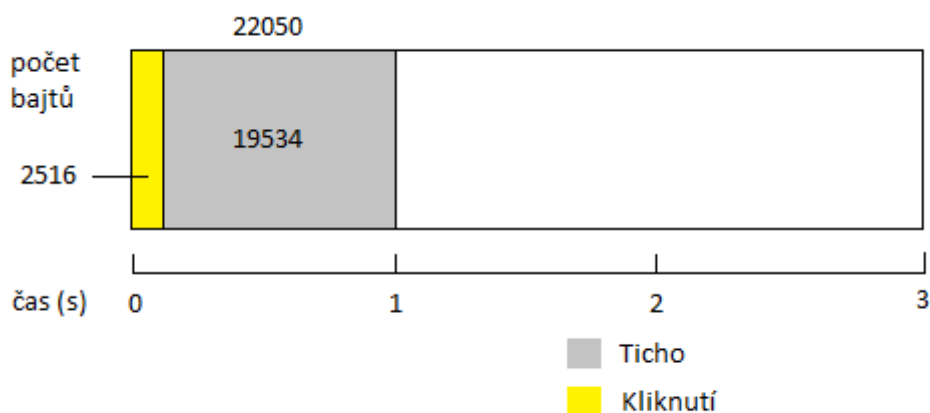
Hlavní aplikační logiku tvoří třída *ClickGenerator*. Při vytvoření instance se jako parametry konstruktoru předávají:

- zvukový soubor (klik metronomu) převedený do pole typu *byte*,
- číslo posledního indexu tohoto pole,
- handler pro komunikaci s ukazateli tempa.

Jak již bylo zmíněno výše, původní myšlenka přehrávat kliknutí metronomu v jednom vlákne a dobu mezi kliknutími ovládat za pomoci uspaní vlákna byla neuskutečnitelná. Proto byla vytvořena metoda *play()*, která do *AudioTracku* současně nahrává zvukový soubor převedený do pole typu *byte* a prázdné pole typu *byte*, které tvoří ticho mezi jednotlivými kliknutími metronomu.

AudioTrack má v této implementaci nastavenou vzorkovací frekvenci na 22050 vzorků. To

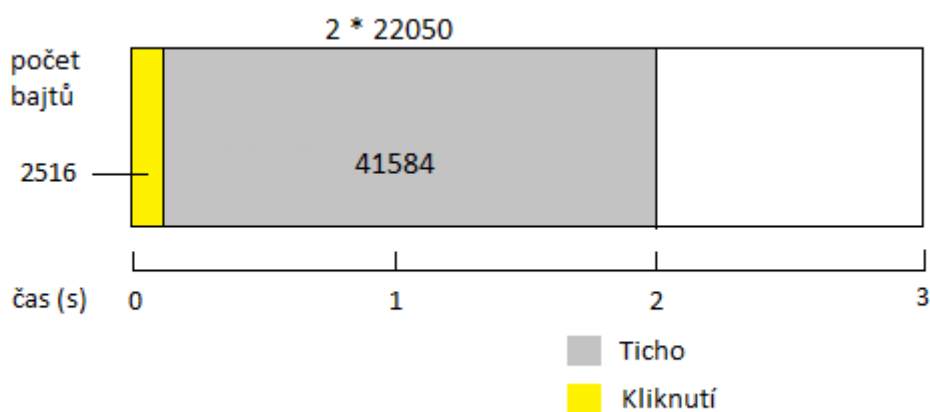
znamená, že je potřeba do něj naplnit 22050 bajtů dat aby proběhl jeden klik za vteřinu (frekvence 60bpm). Viz obrázek č. 13 níže.



Obrázek 13:

AudioTrack v situaci, kdy je tempo 60 úderů za minutu (Zdroj: autor)

Z původního tempa (počet úderů za minutu) je tedy nutné stanovit periodu, která bude udávat v jakém časovém rozsahu je potřeba přehrát jeden zvukový vzorek. Pokud nastavíme hodnotu 30 úderů za minutu, znamená to, že bude potřeba přehrát $2 * 22050$ bajtů. Tato situace je ilustrována na obrázku č. 14 níže. Ještě než je do AudioTracku nahrán zvuk kliknutí ve formě pole typu *byte*, je nutné spočítat jeho délku. Ta je v současné implementaci 2516 bajtů. Následně tuto hodnotu odečteme od $2 * 22050$ bajtů celkového vzorku a dostaneme zbylých 41584 bajtů, které je potřeba doplnit. Proto se do AudioTracku v tomto rozsahu nahraje ticho ve formě prázdného pole – každý element má hodnotu 0.

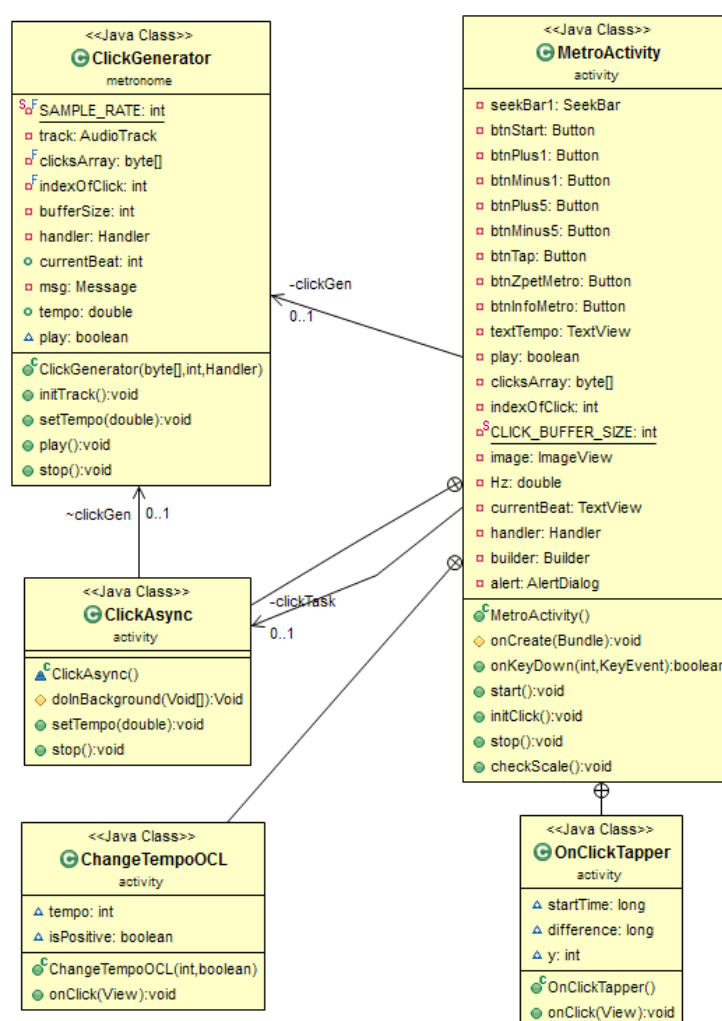


Obrázek 14:

AudioTrack v situaci, kdy je tempo 30 úderů za minutu (Zdroj: autor)

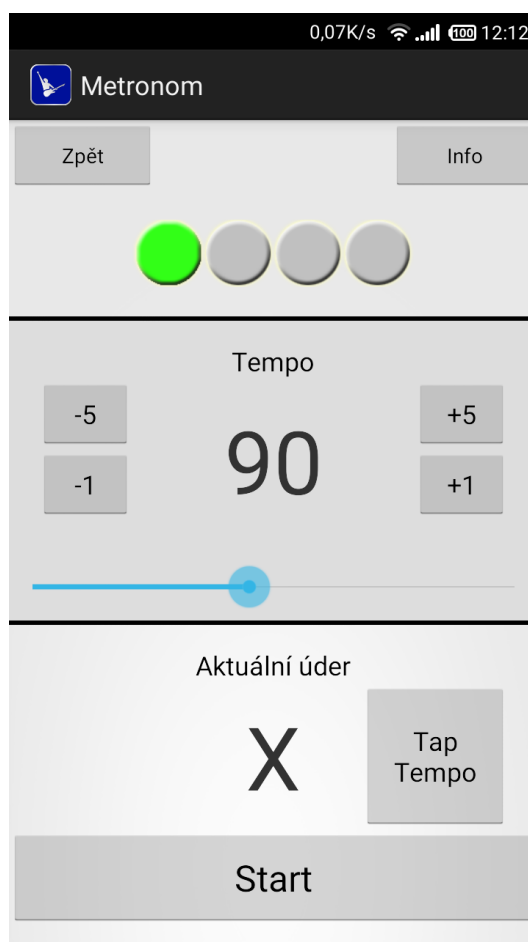
Když je AudioTrack naplněn zvukem kliknutí a tichem za ním podle patřičného tempa, provede se přehrání zvuku. Spolu sním je skrz *handler* předána do třídy *MetroActivity* hodnota aktuálního úderu, která je potřebná pro překreslení ukazatele. Jelikož je AudioTrack inicializován ve *stream* módu, dojde při přehrání k jeho vyprázdnění a celý proces se musí opakovat.

6.2.5 Class Diagram

**Obrázek 15:**

Class Diagram komponenty metronom (Zdroj: autor)

6.2.6 Obrazovka



Obrázek 16:

Grafické uživatelské rozhraní komponenty metronom (Zdroj: autor)

6.3 Komponenta ladička

Následující část práce bude věnována návrhu, implementaci a popisu komponenty, která bude sloužit pro ladění hudebního nástroje. Právě ladička je asi nejpoužívanější pomůckou mezi muzikanty. S prudkým nástupem chytrých telefonů se stala skoro nedílnou součástí programového vybavení, zejména díky možnosti „být stále po ruce“.

6.3.1 Analýza a požadavky

Stejně jako u metronomu je i u ladičky kladen velký důraz na přesnost. Zatímco v prvním případě tato vlastnost představuje provádění operace v neměnném časovém horizontu, u ladičky se jedná o exaktní výpočet frekvence. Hlavním cílem bude tedy implementace logické části, jejíž správná funkčnost bude v průběhu vývoje testována za pomoci generátoru zvuku o konkrétní frekvenci.

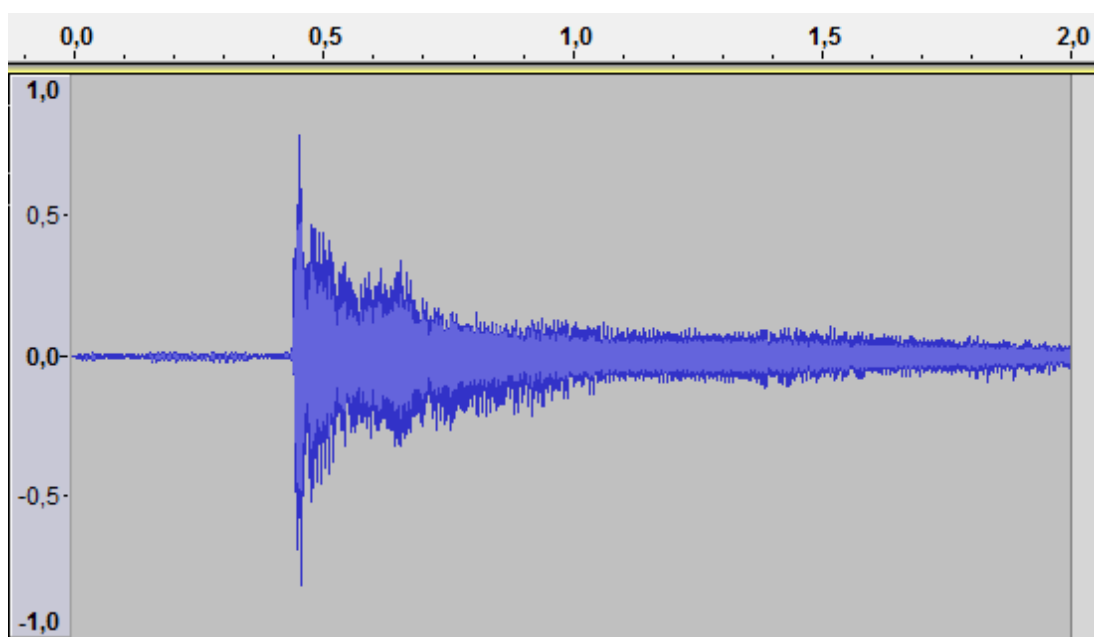
V tomto ohledu je potřeba nastudovat odbornou literaturu a seznámit se s algoritmy, umožňujícími výpočet frekvence z digitalizovaného audiosignálu. Ten bude zaznamenáván skrze hardwarový mikrofón. Další velmi důležitou funkcí ladičky je zobrazení názvu tónu. Z tohoto důvodu bude implementován seznam tónů a jejich přípustných frekvencí. Díky tomu bude možno zobrazit hodnotu frekvence, název tónu a případně o kolik hertzů se od něj frekvence liší. Pro snadnější použití bude v grafickém uživatelském rozhraní komponenty umístěn ukazatel, který bude zobrazovat odchylku od nejbližšího tónu.

Výpočet frekvence – teorie

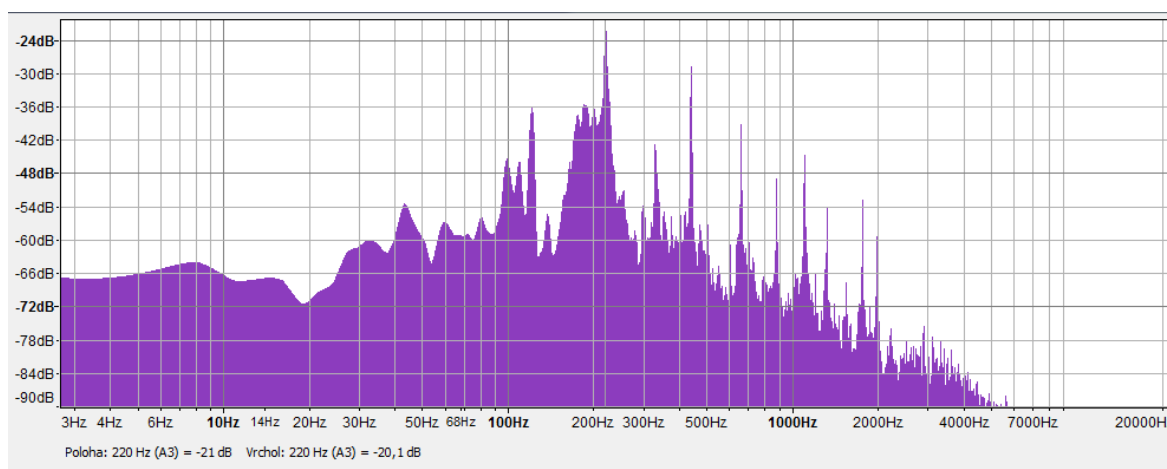
Pokud bychom zaznamenaný signál rozložili, zjistíme, že je tvořen dalšími signály o různých frekvencích. Nachází se zde frekvence, která má nejvyšší amplitudu v celém spektru. Z tohoto důvodu určuje výšku celého tónu. Signál dále obsahuje vyšší harmonické frekvence, které jsou typické pro daný nástroj a tvoří barvu tónu. Posledním prvkem v signálu je šum. Bohužel lidské ucho nedokáže rozeznat jednotlivé signály, kterými je složený tón tvořen a vnímá ho jako jeden tón se specifickým zabarvením [33].

Pro stanovení frekvence s nejvyšší amplitudou je tedy nutné jednotlivé signály oddělit. Pro tyto účely slouží spektrální analýza. Jedná se o metodu, která dokáže transformovat původní v čase vzorkovaný signál do frekvenční oblasti. Touto problematikou se zabývají Fourierovy transformace. V podstatě se jedná o integrální transformace (skalární součiny přes systém vektorů), které jsou založeny na myšlence, že každý signál, respektive funkce, která ho reprezentuje, může být nahrazena jistou lineární kombinací funkcí sinus a kosinus ([33], [34]).

Na obrázku č. 17 níže je vidět vzorkovaný signál před vstupem do Fourierovy transformace. Konkrétně se jedná o nahrávky tónu „A“. Obrázek č. 18 pak ilustruje výstup z tohoto algoritmu. K pořízení obrázků byla provedena Fourierova transformace v programu Audacity.



Obrázek 17:
Reprezentace signálu vzorkovaného v čase (Zdroj: autor)



Obrázek 18:
Signál ve frekvenční doméně (Zdroj: autor)

Jelikož byl při digitalizaci převeden spojitý signál na diskretní periodický, je nutné použít metodu Diskretní Fourierovy transformace, konkrétně v praxi nepoužívanější Rychlou Fourierovu transformaci (FFT - Fast Fourier Transform). Jak již z názvu vyplývá, jedná se o nejrychlejší typ algoritmu Fourierových transformací, převážně proto, že není jako ostatní založen na práci s goniometrickými funkcemi, ale funguje na principu rozdělování do dalších transformací.[34]

Hlubší rozbor algoritmu Fourierovy transformace není účelem této práce, proto se mu nebudu blíže věnovat. Pro hlubší seznámení s touto problematikou doporučuji knihu:

NEVŘIVA, Pavel. *Analýza signálů a soustav*. 1. vyd. Praha: BEN - technická literatura, 2000, 671 s. ISBN 80-7300-004-0

Pro výpočet Rychlé Fourierovy transformace v prostředí systému Android existují externí knihovny. Následující tabulka obsahuje nalezené knihovny v rámci rešerše a jejich silné a slabé stránky, které později hrály roli při výběru.

Tabulka 2: Srovnání knihoven pro výpočet FFT, zdroj[autor]

Název knihovny	Silné stránky	Slabé stránky	Zdroj
FFTW 3.3.4	Nejpoužívanější	Náročné na zdroje Pomalejší než JTransforms	http://www.fftw.org
JTransforms	Open source Nativní Java Rychlejší než FFTW	-	https://sites.google.com/site/piotrwendykier/software/jtransforms
ooura FFT	-	Nutnost použít wrapper Horší dokumentace	http://www.kurims.kyoto-u.ac.jp/~ooura/fft.html

Pro implementaci výpočtu Rychlé Fourierovy transformace byla nakonec vybrána knihovna JTransforms.

Po provedení spektrální analýzy zvukového signálu je ještě nutné provést vyhledání frekvence s nejvyšší amplitudou, které bude dosaženo za pomoci cyklického algoritmu, procházejícího výsledek Rychlé Fourierovy transformace.

6.3.2 Příklad užití

Komponenta bude sloužit pro ladění hudebního nástroje. Pro zjednodušení je příklad užití ilustrován na ladění akustické kytary:

Uživatel brnkne na konkrétní strunu a na grafickém uživatelském rozhraní je mu zobrazena aktuální frekvence, název nejbližšího tónu a případná odchylka od něj. Podle toho uživatel přitahuje nebo povoluje strunu. Proces se opakuje, dokud se na ukazateli nerozsvítí zelený prvek a zelený název frekvence, který určuje, že je kytara přesně naladěna do konkrétního tónu.

6.3.3 Problémy při realizaci

Při vývoji a testování komponenty jsem narazil na následující problémy:

Hlasitostní minimum

Jelikož při spuštění komponenty ladička dochází k neustálému zaznamenávání a vyhodnocování hodnot, bylo potřeba ošetřit situace, kdy vstupní hodnoty tvoří vzorky s nízkou hlasitostí. K tomu dochází v momentech, kdy uživatel zrovna ladičku nepoužívá, například mezi laděním jednotlivých strun. V praxi se sice jedná o krátké časové úseky v jednotkách sekund, nicméně pro tyto situace byla implementována podmínka kontrolující hlasitost zaznamenávaného záznamu, která je popsána v podkapitole níže.

Šum

Z podobného důvodu, kvůli kterému byla naprogramována podmínka hlasitostního minima, byla také zavedena kontrola šumu. V případě, kdy výstupem z logické části aplikace je vypočítaná frekvence nižší než 26 Hz, není tento výsledek předán do grafického uživatelského rozhraní k zobrazení. Právě v rozsahu těchto hodnot je mikrofonom zaznamenáván šum a bez zavedení této kontroly by při nepoužívání ladičky docházelo k zobrazení jeho hodnot na ukazateli.

Rychlost

Doba, mezi kterou uživatel zahraje tón na hudební nástroj a zobrazením vypočítané frekvence na grafickém uživatelském rozhraní, není překvapivě dána z velké části výpočetní složitostí spektrální analýzy. Hlavní roli hraje fakt, že je nutno nejprve zaznamenat zvuk s patřičnou časovou délkou. Čím více dat je zachyceno, tím je možné přesněji vypočítat frekvenci. Na druhou stranu, s delší prodlevou se snižuje uživatelská přívětivost. Proto bylo prováděno testování různých délek záznamů a současné nastavení představuje kompromis mezi oběma požadavky.

Tlačítka pro omezení rozsahu

Pro zvýšení přesnosti ladění byla implementována funkcionality umožňující nastavení sady not. V praxi to znamená, že pokud bude uživatel chtít naladit například strunu „D“, zvolí si za pomoci tlačítka v grafickém uživatelském rozhraní sadu frekvencí, které může tónu „D“ nabývat. Výpočet odchylky od tónu pak bude prováděn na přibližně 5x menším rozsahu hodnot.

6.3.4 Popis

Komponenta se skládá z následující pěti tříd:

Třída Notes

Jak již bylo zmíněno v podkapitole 6.3.1, věnující se analýze a požadavkům na komponentu ladička, je nutné uchovávat seznam tónů a frekvencí, které mohou nabývat. Právě pro tyto účely byla vytvořena třída *Notes*. Jelikož se jedná o statickou třídu, není nutné vytvářet její instanci. Po stanovení frekvence s nejvyšší amplitudou je tato hodnota předána jako parametr do metody *getIndex()*, která prohledá pole evidující frekvence jednotlivých tónů a najde tón s nejbližší frekvencí. Tato metoda je obsahem výpisu kódu č. 4 níže. Jeho index pak vrátí jako návratovou hodnotu. Pro získání názvu tohoto tónu je pak použita metoda *getNote()* s číslem indexu jako parametrem. Poslední *getter*, který se ve třídě nachází, vrací hodnotu odchylky od nejbližšího tónu.

```
public static int getIndex(double freq){
    double nejmensiOdchylka = Math.abs(freq - N[0]);
    int nejblizsiIndex = 0;
    for(int i = 0; i < N.length; i++){
        double odchylka = Math.abs(freq - N[i]);
        if(odchylka < nejmensiOdchylka){
            nejblizsiIndex = i;
            nejmensiOdchylka = odchylka;
        }
    }
    return nejblizsiIndex;
}
```

Výpis kódu 4:

Metoda getIndex() (Zdroj: autor)

Třída LadActivity

Pro podporu grafického uživatelského rozhraní byla v komponentě ladička implementována třída *LadActivity*. Jak již z názvu vyplívá, jedná se o potomka třídy *Activity* z balíčku *android.app*. Interakce s uživatelem je zajištěna za pomoci několika tlačítek. Každému z nich je přiřazen patřičný *onClickListener*, který určuje jeho funkci.

Za zajímavou část považuji vyřešení problému předání informace o výsledné frekvenci z logické části komponenty. Systém Android totiž umožňuje přístup k prvkům grafického uživatelského rozhraní pouze z hlavního vlákna aplikace. Proto bylo nutné ve třídě *LadActivity* předdefinovat prvky zobrazené ve výpisu kódu č. 5 níže.

```
private Thread callback = new Thread() {
    @Override
    public void run() {
        if (recording) {
            frequency = recorder.getFreq();
            index = Notes.getIndex(frequency);
            updateGUI();
        } else {
            this.interrupt();
        }
    }
};
handler = new Handler();
```

Výpis kódu 5:

Deklarace prvků pro předávání informace do hlavního vlákna (Zdroj: autor)

Při procesu vytváření instance třídy *Recorder* jsou tyto prvky použity jako parametry konstrukturu. Pro odeslání vypočtené frekvence zpět do třídy *LadActivity* stačí na instanci *handleru* zavolat metodu *post()* a jako parametr uvést výše zmíněný objekt typu *Runnable*. Právě tato metoda dokáže předat informaci z vedlejšího vlákna do hlavního.

Třída *Recorder*

Mezi hlavní funkce třídy *Recorder* patří zaznamenávání zvuku do instance třídy *AudioRecord* z balíčku *android.media*. Při volání konstrukturu třídy *Recorder* dochází k přetypování vstupních parametrů a jejich uložení do proměnných. Jedná se o instanci třídy *Handler* z balíčku *android.os* a objekt typu *Runnable* definovaný ve třídě *LadActivity*. V dalším kroku je provedena inicializace *AudioTracku* podle příslušných parametrů.

Jelikož je třída *Recorder* potomkem třídy *AsyncTask*, dědí i její metodu *doInBackground()*. Ta plní klíčovou úlohu logické části komponenty, která je prováděna v následujících krocích:

1. Zaznamenání zvukových vzorků.
2. Jejich předání do třídy *Analyzer* jako parametr konstrukturu.
3. Zavolání metody *getFreq()* pro výpočet frekvence.
4. Předání výsledků do třídy *LadActivity* za pomoci handleru.

Detailnější popis metody *doInBackground()*:

Nejprve je na instanci *AudioTracku* zavolána metoda *startRecording()*, díky které začne zaznamenávání zvuku. Algoritmus pokračuje zavoláním metody *read()*. Ta slouží k uložení zaznamenaných vzorků do pole typu *Byte*, přičemž její návratovou hodnotu tvoří počet bajtů, které byly uloženy. Díky tomu je možné kontrolovat, zda jsou ještě dostupné vzorky ke zpracování.

Následuje vytvoření instance třídy *Analyzer*. Jako parametr je do konstrukturu předáno pole typu *Byte* s uloženými zvukovými vzorky. Třída *Analyzer* slouží k výpočtu primární frekvence za pomoci Rychlé Fourierovy transformace a prohledání spektra za účelem nalezení frekvence s nejvyšší amplitudou. Právě proto je v dalším kroku metody *doInBackground()*,

zavolána v Analyzeru metoda *getFreq()* a její návratová hodnota je uložena do lokální proměnné.

V této chvíli už známe hledanou frekvenci. Proto můžeme na instanci třídy *Handler*, která byla do třídy *Recorder* předána jako parametr konstruktoru ze třídy *LadActivity* zavolat metodu *post()*. Jelikož přístup k prvkům grafického uživatelského rozhraní je v systému Android možný pouze z hlavního vlákna, slouží metoda *post()* ke spuštění objektů typu *Runnable* ve vláknech na pozadí. To ji činí velmi účinnou v situacích, kdy je potřeba přistupovat k uživatelskému rozhraní odjinud než z hlavního vlákna aplikace. Hodnota výsledné frekvence je tedy předána zpět do třídy *LadActivity*. Zde se zavolá metoda, s jejíž pomocí se frekvenci zobrazí v grafickém uživatelském rozhraní.

Následuje výpis kódu č. 6 s metodou *doInBackground()*.

```
@Override
protected Void doInBackground(Void... params) {

    while(recording){

        record.startRecording();

        if(record != null){
            int n = 1;

            while (n > 0) {

                n = record.read(byteBuffer, 0, bufferSize);
                analyzer = new Analyzer(byteBuffer);
                frequency = analyzer.getFreq();
                handler.post(callback);

            }
        }
        record.stop();

        return null;
    }
}
```

Výpis kódu 6:

Metoda `doInBackground()` ve třídě `Recorder` (Zdroj: autor)

Třída Analyzer

Třída *Analyzer* slouží k výpočtu primární frekvence za pomoci metod externí knihovny pro výpočet Rychlé Fourierovy transformace. Důležitým krokem je také prohledání spektra za účelem nalezení frekvence s nejvyšší amplitudou. Jak již bylo zmíněno výše v popisu třídy *Recorder*, prvkem vstupujícím do instance ve formě parametru konstruktoru je pole typu *Byte* obsahující zaznamenané vzorky signálu.

Jelikož tyto vzorky představují změnu napětí v audiosignálu zaznamenaného podle Pulzně kódové modulace, tvoří je reálná čísla zaznamenaná v čase. Proto bude do Rychlé Fourierovy transformace předáno pouze jednorozměrné pole. Za takovýchto podmínek není nutné použít hodnoty z celého pole, protože, jak je zmíněno v dokumentaci, knihovna *JTransforms* si dokáže druhou polovinu dopočítat jako obraz hodnot té první.

Pro lepší porozumění uvádím výpis kódu č. 7, který obsahuje metody *getFreq()* a *getHlasitost()*.

Nejprve je ovšem nutné ošetřit výše zmíněnou podmínku hlasitostního minima. K tomu slouží metoda *getHlasitost()*, která provede aritmetický průměr na poli zvukových vzorků a výsledek vrátí jako návratovou hodnotu do parametru podmínky. Pokud je vrácená hodnota příliš nízká, provádění výpočtu frekvence končí a jako návratová hodnota je vrácena frekvence 0.0.

<pre>public double getFreq() { if(getHlasitost() < 30){ freq = 0.0; return freq; } byteToDouble(); transformData(); determineFreq(); return freq; }</pre>	<pre>public double getHlasitost() { int pocetPrvku = audioData.length; double hlasitost = 0.0; double prumernaHlasitost = 0.0; for(int i=0; i<pocetPrvku; ++i) { hlasitost+=Math.abs(audioData[i]); } prumernaHlasitost = hlasitost / pocetPrvku; return prumernaHlasitost; }</pre>
--	--

Výpis kódu 7:

Metody *getFreq()* a *getHlasitost()* (Zdroj: autor)

Pokud je naopak podmínka splněna je nutné převést pole typu *Byte* na pole typu *Double*, jelikož vstupní hodnota do metody pro výpočet transformace vyžaduje pole typu *Double*. K tomuto účelu byla implementována metoda *byteToDouble()* zobrazená níže.

```
public void byteToDouble() {  
  
    ByteBuffer byteBuf = ByteBuffer.wrap(audioData);  
    byteBuf.order(ByteOrder.LITTLE_ENDIAN);  
  
    for (int i = 0; i < bufferSize; i++) {  
        double mono = (double) byteBuf.getShort();  
        audioDbl[i] = mono;  
    }  
}
```

Výpis kódu 8:

Metoda byteToDouble() (Zdroj: autor)

Následně je zavolána metoda *realForward()* z knihovny JTransforms. Jako parametr je použito pole typu *Double*, které nyní obsahuje polovinu zaznamenaných hodnot. Výsledek transformace je uložen zpět do tohoto pole.

Výstupem z Fourierových transformací je posloupnost komplexních čísel. Ty jsou složeny z reálné a imaginární části [35].

```
realForward  
  
public void realForward(double[] a)  
  
Computes 1D forward DFT of real data leaving the result in a . The physical layout of the output data is as follows:  
if n is even then *  
  
a[2*k] = Re[k], 0<=k<n/2 a[2*k+1] = Im[k], 0<k<n/2 a[1] =  
Re[n/2]  
  
if n is odd then *  
  
a[2*k] = Re[k], 0<=k<(n+1)/2 a[2*k+1] = Im[k], 0<k<(n-1)/2  
a[1] = Im[(n-1)/2]  
  
This method computes only half of the elements of the real transform. The other half satisfies the symmetry condition. If you want the full real forward  
transform, use realForwardFull. To get back the original data, use realInverse on the output of this method.  
  
Parameters:  
a - data to transform
```

Obrázek 19:

Metoda realForward() (Zdroj: dokumentace ke knihovně JTransforms)

Jak je vidět na obrázku č. 19 výše, výsledné pole typu *Double*, které se vrátí jako návratová hodnota metody *realForward()*, obsahuje reálnou část na indexu $2*k$ a imaginární část na pozici $2*k+1$.

Jelikož spektrální analýza již proběhla, dalším krokem je nalezení frekvence s nejvyšší amplitudou. K tomuto účelu slouží metoda *determineFreq()* zobrazená ve výpisu kódu č. 9 níže.

```
public void determineFreq() {  
    int c = 1;  
    int bigC = 1;  
    double real = 0;  
    double imag = 0;  
    double ampl = 0;  
    double bigAmpl = 0;  
    DoubleBuffer buff = DoubleBuffer.wrap(audioDbl);  
    while (buff.hasRemaining()) {  
        real = buff.get();  
        if (buff.hasRemaining()) {  
            imag = buff.get();  
        }  
        ampl = Math.sqrt(Math.pow(real, 2) + Math.pow(imag, 2));  
        if (ampl > bigAmpl) {  
            bigC = c;  
            bigAmpl = ampl;  
        }  
        c++;  
    }  
    freq = 44100.0 * bigC / bufferSize;  
}
```

Výpis kódu 9:

Metoda determineFreq() (Zdroj: autor)

Amplituda jednotlivých vzorků se vypočítá jako velikost vektoru komplexního čísla, tj. za pomoci Pythagorovy věty:

$$\text{amplituda} = \sqrt{\text{reálná část}^2 + \text{imaginární část}^2} \quad [35]$$

Metoda *determineFreq()* funguje tak, že nejprve načte komplexní čísla do instance třídy *DoubleBuffer* a pak provádí výpočet amplitudy, dokud není buffer vyprázdněn, přičemž do lokální proměnné ukládá maximální nalezenou amplitudu a index, na kterém se nachází reálná část. Pro výpočet frekvence se pak použije následující vzorec, přičemž velikost bufferu je rovna polovině zaznamenaných vzorků.

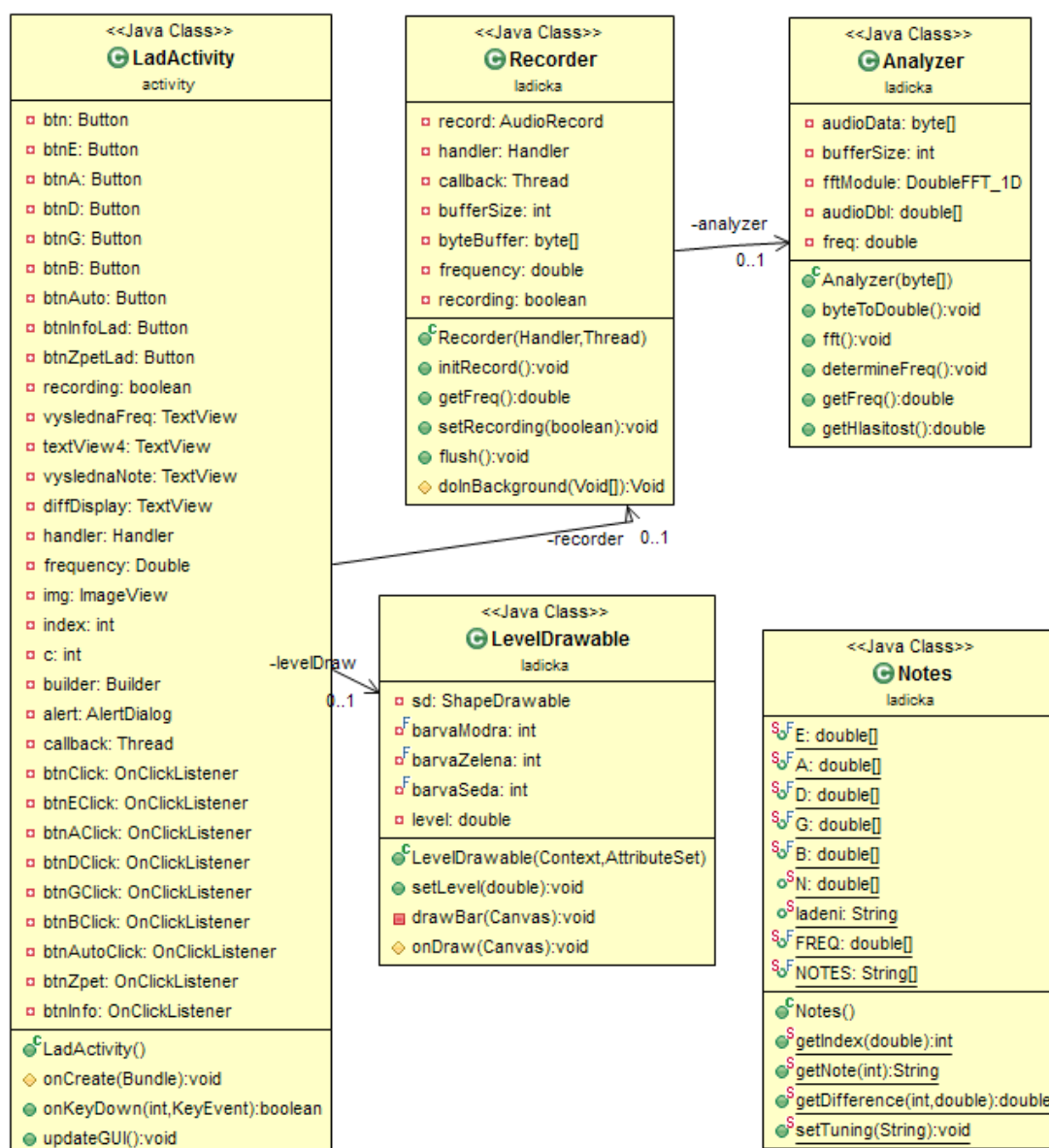
$$frekvence = vzorkovací\ frekvence * \frac{index\ s\ nejvyšší\ amplitudou}{velikost\ bufferu} \quad [35]$$

Třída `LevelDrawable`

Pro zobrazení ukazatele byla implementována třída *LevelDrawable*, která je potomkem třídy *View* z balíčku *android.view*. Velkou část třídy tvoří logika pro nakreslení řady čtverců, které slouží pro snadnější orientaci při ladění nástroje. Princip je poměrně jednoduchý: hodnota aktuální odchylky frekvence od nejbližšího tónu je použita jako podmínka pro zabarvení jednotlivých prvků

.

Class Diagram



Obrázek 20:

Class Diagram komponenty ladička (Zdroj: autor)

6.3.5 Obrazovka



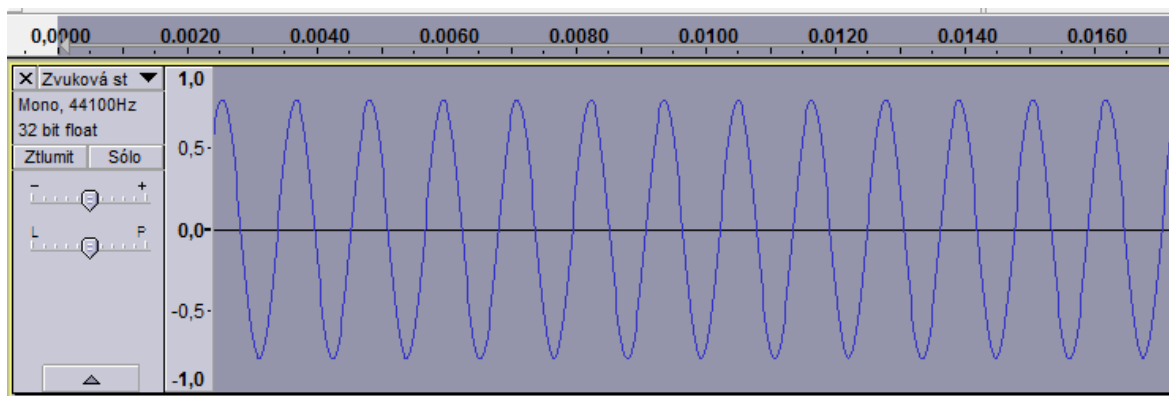
Obrázek 21:

Grafické uživatelské rozhraní komponenty ladička (Zdroj: autor)

6.3.6 Porovnání

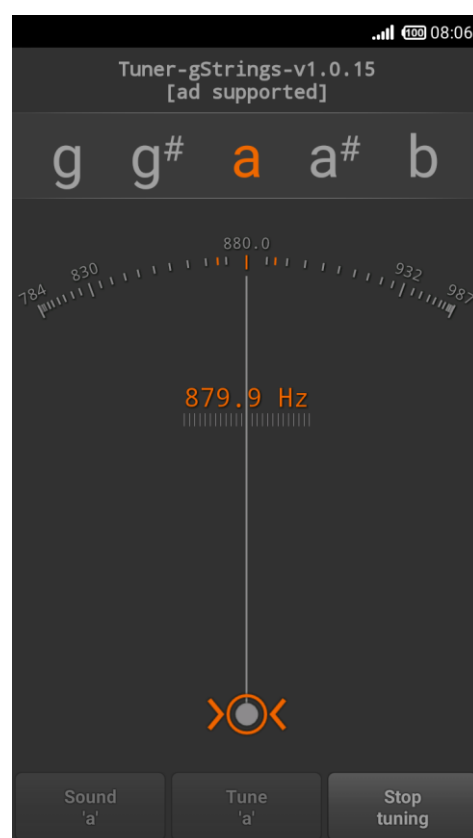
Po dokončení vývoje komponenty jsem provedl srovnání s asi nejpoužívanější ladičkou na platformě Android. Referenční sinusoida byla vždy vygenerována v programu Audacity. Po několika měřeních bylo zjištěno, že ladička vyvinutá v rámci této práce se odchyluje průměrně o 0,84 Hz od vytvořeného tónu. Její protějšek byl na tom poměrně lépe s průměrnou hodnotou odchylky okolo 0,37 Hz. Horší výsledek testu příkládám onomu výše zmíněnému kompromisu mezi délkou zaznamenávaného signálu a přesností výpočtu.

Následuje několik obrázků z průběhu testování:



Obrázek 22:

Vygenerovaný tón o frekvenci 880 Hz v programu Audacity (Zdroj: autor)



Obrázek 23:

Sejmuté obrazovky při testování přesnosti ladění (Zdroj: autor)

6.4 Komponenta EarTrainer

Intervaly znamenají v hudební teorii vzdálenosti mezi dvěma tóny. V případě hry na kytaru není úplně podmínkou, aby je hráč znal. Skladbu je možné zahrát i mechanicky. Nicméně jsou potřebné při komunikaci s jiným hráčem, při zpěvu, improvizaci, čtení hudebního zápisu či skládání. Proto třetí část aplikace tvoří komponenta zaměřená na procvičování hudebních intervalů.

6.4.1 Analýza a požadavky

Jak již vyplývá z definice intervalu, hlavní funkcí komponenty bude přehrávání dvou zvukových vzorků s určitým časovým rozestupem. Jelikož je cílem procvičovat hudební sluch, je nutné, aby byly tyto vzorky určovány náhodně. Další požadavek bude kladen na dostatečnou velikost intervalů. Je potřeba, aby k procvičování docházelo na rozsahu celé oktávy.

Jak již bylo zjištěno při vývoji komponenty metronom, není vhodné používat generovanou zvukovou vlnu, proto budou použity reálné zvuky nástrojů. Pro lepší hratelnost bude implementována možnost si zvolit z několika úrovní obtížnosti. S tím souvisí i evidence skóre, které bude tvořit počet správných odpovědí. Poslední požadavek je kladen na grafické zobrazení intervalu na kytarovém hmatníku, jež bude sloužit pro snazší představu a pomůže vizuálně pochopit a analyzovat daný interval.

6.4.2 Příklad užití

V prvním kroku si uživatel zvolí z několika úrovní obtížnosti. Po stisknutí tlačítka dojde k přehrání dvou zvukových vzorků s určitým časovým rozestupem. Uživatel pak za pomoci ovládacích prvků vybírá z dostupných odpovědí, které jsou současně vyobrazeny na grafickém ukazateli. Pokud si není uživatel jistý správnou odpovědí, může si interval ještě jednou přehrát. Po odeslání odpovědi se provede vyhodnocení. V případě správné odpovědi je do skóre přičtena hodnota 10. Pokud byla využita možnost přehrát zvuk znovu, je přičtena pouze hodnota 5. V případě špatné odpovědi zůstává skóre nezměněno. Hra pokračuje dalším kolem. Po deseti zodpovězených intervalech hra končí a uživateli je nabídnuto, zda si chce svůj výsledek uložit. Následně je zobrazena *Aktivita* s nejlepšími dosaženými výsledky.

6.4.3 Problémy při realizaci

Během vývoje a testování komponenty jsem narazil na následující problémy:

Zarovnání prvků *RadioButton*

Při procvičování intervalů na rozsahu jedné oktávy je nutné evidovat 8 možných odpovědí. Z tohoto důvodu bylo zvoleno provedení každé odpovědi ve formě zaškrtnutí jednoho z prvků *RadioButton*. Nicméně protože se tyto prvky vzájemně vylučují, je nutné je shlukovat uvnitř *RadioGroup*. Seskupením dohromady systém zajišťuje, že z nich bude možné vybrat pouze jeden. Bohužel ani v poslední verzi Androidu není možné obsah *RadioGroup* rozdělit například do několika sloupců. Jediná možnost je tedy mít prvky zarovnané pod sebou. Při zpracovávání uživatelského rozhraní bylo ovšem prvky potřeba rozdělit do dvou sloupců. Řešením se nakonec stalo vytvoření dvou *RadioGroup*. Pro každou z nich byl nadefinován *OnCheckedChangeListener*. Jejich funkcí je zajistit, aby bylo možné zaškrtnout pouze jeden *RadioButton* v obou *RadioGroup* celkem.

Ukládání skóre

Z počátku se nejlepší dosažené výsledky vkládaly do *TextView* v aktivitě *HighScoreActivity*. Problém ovšem nastal v situaci, kdy byla aplikace ukončena nebo byl telefon například restartován. Jelikož byla data uchovávána pouze v operační paměti, došlo k jejich ztrátě. Proto byla implementována funkce pro ukládání dat do textového souboru v interním úložišti zařízení. Skóre je evidováno jako textový řetězec typu *String* a pro jeho načtení k zobrazení v *HighScoreActivity* jsou používány parsery, které dokážou oddělit jednotlivé záznamy.

6.4.4 Popis

Komponenta se skládá ze tří tříd:

Třída *EarActivity*

Potomek třídy *Activity* představuje viditelnou část komponenty. Jelikož se jedná o tzv. „aktivitu“, slouží tato třída jako grafické uživatelské rozhraní. Stejně jako v předchozích komponentách se i zde nacházejí ovládací prvky a k nim přidružené handlers, které určují jejich funkce. Mimo jiné jsou zde implementována i dialogová okna, včetně metod pro kontrolu

vstupů od uživatele.

⁴Za zmínku stojí logika ukládání nejlepších výsledků:

Když je odeslána desátá odpověď a hra končí, je uživateli nabídnuto, zda si chce svůj výsledek uložit. Do dialogového okna napíše své jméno, a pokud vyhovuje předdefinovaným parametrům (délka, povolené znaky), jsou jméno a hodnota skóre předány do dalšího kroku.

Za pomoci třídy *InputStreamReader* je nejprve načten textový soubor, který slouží pro evidenci nejlepších dosažených výsledků. Proces pokračuje přes instance tříd *BufferedReader* a *StringBuilder*, které transformují obsah textového souboru na typ *String* a uloží jej do připravené proměnné. Následně je její obsah metodou *split()* z balíčku *java.lang.String* a příslušným parametrem rozdělen a uložen do jednorozměrného pole typu *String*. Samotný element pole v tuto chvíli obsahuje Stringový řetězec, ve kterém se nachází jméno a hodnota skóre oddělená znakem dvojtečky. V dalším kroku je provedena podmínka na velikost pole. V podstatě se jedná o kontrolu, kolik je v tuto chvíli evidováno záznamů.

Pokud je jich méně než 5, je za pomoci instance třídy *OutputStreamWriter* vloženo jméno hráče a jeho dosažené skóre.

V případě, kdy už pole obsahuje 5 a více záznamů, metoda pokračuje. Nejprve je spuštěn cyklus, který opět za pomoci metody *split()*, parseru a pole kontroluje skóre, přičemž počet opakování cyklu je dán počtem prvků v poli. Cílem cyklu je nalézt nejnižší hodnotu skóre ve všech evidovaných záznamech. Jelikož by nemělo logický smysl vkládat nižší hodnotu, je po dokončení poslední iterace nutné provést kontrolu, zda skóre, které chceme vložit, je vyšší než nejnižší evidovaná hodnota. Pokud není, metoda končí neuložením skóre.

V opačném případě jsou nalezený záznam s nejnižším dosaženým výsledkem a pole obsahující všech 5 doposud evidovaných záznamů předány jako parametry do metody *removeElements()*, která je součástí výpisu kódu č. 10. Její princip je jednoduchý:

V prvním kroku dojde k vytvoření instance kolekce typu *LinkedList* z balíčku *java.util*. Následuje cyklus, který prochází všech 5 prvků pole předaného parametrem a porovnává je

⁴ Pro snazší orientaci v popisu průběhu metody *writeToFile()* doporučuji pročítat zdrojový kód, který je součástí přílohy této práce

s prvkem, který je potřeba odstranit (nejmenší hodnota skóre). Pokud se při porovnávání aktuální záznam nerovná hledanému, je vložen do kolekce. Posledním krokem je převedení kolekce zpět na pole, které je vráceno jako návratová hodnota metody.

V této chvíli bylo získáno pole, které obsahuje 4 záznamy. Následně je rozšířeno o další záznam. Konkrétně se jedná o Stringový řetězec obsahující jméno a skóre uživatele, které chceme uložit, oddělené dvojtečkou.

V závěru je toto pole převedeno na textový řetězec typu *String* a za pomoci instance třídy *OutputStreamWriter* uloženo do textového souboru v interním úložišti.

```
public static String[] removeElements(String[] input, String toDelete) {
    List<String> result = new LinkedList<String>();

    for(String item : input){
        if(!toDelete.equals(item))
            result.add(item);
    }
    String[] pole = (String[]) result.toArray(input);
    String[] novePole = new String[5];
    for(int i = 0 ; i < pole.length; i++){
        if(pole[i] != null){
            novePole[i] = pole[i];
        }
    }
    return novePole;
}
```

Výpis kódu 10:

Metoda removeElements() (Zdroj: autor)

Třída Notes

Logickou část komponenty EarTrainer tvoří třída *Notes*, která slouží pro načtení zvukových vzorků a jejich přehrávání. Při vytvoření její instance se nejprve provede inicializace instance třídy *SoundPool* z balíčku *android.media*, jejíž vlastnosti jsou popsány v teoretické části této práce. V dalším kroku jsou načteny jednotlivé zvuky za pomoci metody *load()*, která vrací hodnotu typu *Integer*. Z tohoto důvodu jsou zvuky uchovávány právě v proměnných typu *Integer*. Následně jsou tyto hodnoty vloženy do kolekce typu *HashMap*. Klíčovou hodnotu záznamu pak tvoří délka intervalu, viz výpis kódu č. 11 níže.

```

public void init() {

    sp = new SoundPool(5, AudioManager.STREAM_MUSIC,0);
    sp.setOnLoadCompleteListener(new OnLoadCompleteListener() {
        @Override
        public void onLoadComplete(SoundPool soundPool, int sampleId,int status) {
            loaded = true;
        }
    });

    c = sp.load(context, R.raw.c, 1);
    d = sp.load(context, R.raw.d, 1);
    e = sp.load(context, R.raw.e, 1);
    f = sp.load(context, R.raw.f, 1);
    g = sp.load(context, R.raw.g, 1);
    a = sp.load(context, R.raw.a, 1);
    h = sp.load(context, R.raw.h, 1);
    c2 = sp.load(context, R.raw.c2, 1);

    generator = new Random();
    done = true;
}

public void initPole(){

    pole.put(1,c);
    pole.put(3,d);
    pole.put(5,e);
    pole.put(6,f);
    pole.put(8,g);
    pole.put(10,a);
    pole.put(12,h);
    pole.put(13,c2);
}

```

Výpis kódu 11:

Metoda init() a initPole() ve třídě Notes (Zdroj: autor)

V momentě, kdy uživatel po spuštění komponenty vybere obtížnost, je za pomoci metody *setDifficulty()* ve třídě *Notes* nastavena hodnota proměnné *level*. Když pak v dalším kroku uživatel stiskne tlačítko pro přehrání intervalu, dojde k zavolání metody *setTony()*. Ta ve své podstatě funguje jako *setter*, který zavolá metodu *getRandom()* a její návratovou hodnotu uloží do lokální proměnné. Jelikož je nutné nastavit dva tóny, které mají být přehrány, je nutné zavolat metodu *getRandom()* dvakrát. Ta je implementována tak, že za pomoci instance třídy *Random* z balíčku *java.util* a dle aktuálně nastavené úrovně obtížnosti v proměnné *level* vrací náhodný element z pole obsahující zvukové vzorky. V tomto momentě jsou 2 zvuky tvořící interval inicializovány do proměnných *prvniZvuk* a *druhyZvuk*.

Pro přehrávání intervalu slouží instance třídy *AsyncTask*. Při provádění její klíčové metody

doInBackground() je pak na instanci třídy *SoundPool* dvakrát volána metoda *play()*. Při prvním volání je jako parametr uvedena proměnná *prvniZvuk*. Následně je zavolána metoda *sleep()* s parametrem 2000. Tímto je docíleno asi dvousekundové prodlevy mezi přehráním jednotlivých zvuků. V posledním kroku je opět zavolána metoda *play()*, tentokrát s hodnotou parametru *druhyZvuk*.

Pro kontrolu správných opovědí slouží metoda *getInterval()*, která vrací hodnotu typu *Integer*, představující vzdálenost mezi právě přehranými zvuky. Tato metoda je volána při stisknutí tlačítka pro odeslání odpovědi, a právě proto je její návratová hodnota předána do třídy *EarActivity*, kde se provede porovnání s odpovědí uživatele. Metoda *getInterval()* je součástí výpisu kódu č. 12.

```
public int getInterval(int prvniZvuk, int druhyZvuk){
    int i;
    if(prvniZvuk > druhyZvuk){
        i = prvniZvuk - druhyZvuk;
        return i*(-1);
    }
    else{
        i = druhyZvuk - prvniZvuk;
        return i;
    }
}
```

Výpis kódu 12:

Metoda getInterval() (Zdroj: autor)

Třída HighScoreActivity

Pro načtení obsahu z textového souboru a zobrazení nejvyšších dosažených výsledků slouží třída *HighScoreActivity*. Její klíčovou část tvoří metoda *readFromFile()*, která je v podstatě opakem metody *writeToFile()* z třídy *EarActivity*. Za pomoci instancí tříd *InputStreamReader*, *BufferedReader* a *StringBuilder* je načten patřičný textový soubor z interního úložiště a převeden na textový řetězec typu *String*.

Pro lepší orientaci v textu doporučuji současně při čtení procházet výpis kódu č. 13.

Jelikož byly pro větší přehlednost do grafického zobrazení implementovány dva *textView* prvky, první pro jméno hráče a druhý pro jeho skóre, je nutné řetězec vrácený metodou *readFromFile()* rozdělit.

Pro tento požadavek je nejprve opět využita metoda *split()*, která byla již použita ve třídě *EarActivity*. Ta obsah Stringového řetězce podle patřičného parseru vloží jako prvky jedno-rozměrného pole typu *String*. Při prvním spuštění komponenty není samozřejmě evidováno žádné skóre, proto je zde ošetřena podmínka kontrolující velikost pole. Pokud je tedy pole prázdné, metoda končí.

V opačném případě je každý prvek nejprve rozdělen metodou *split()*. Tímto jsme zvlášť oddělili jméno a skóre, které jsou v dalším kroku vloženy do kolekce typu *TreeMap*. Tento typ kolekce byl zvolen, jelikož dokáže uchovávat dva prvky rozdílných datových typů a obsahuje implicitní komparátor, který zaručuje, že bude skóre seřazeno dle velikosti. Když jsou všechny záznamy vloženy a seřazeny, následuje extrakce obou prvků (jméno a skóre) z každého záznamu.

Prvky jsou pak postupně vkládány do připravených Stringových řetězců, podle toho jestli se jedná o jméno či skóre. Za každý záznam je na konci přidán znak "\n", aby se následující text zobrazoval na novém řádku. Po vložení všech hodnot se pak každý Stringový řetězec za pomoci metody *setText()* nastaví jako text pro přidělený *TextView* a na obrazovce se zobrazí seznam se jmény uživatelů a jejich dosažených výsledků.

```
mapa = new TreeMap<>(Collections.reverseOrder());

String[] separated = readFromFile().split(",");

if(separated.length!=0){

    if(separated[0]==""){
        return;
    }

    for (int i = 0; i < separated.length; i++) {
        String jednotliv = separated[i];

        String[] separatedJednotlive = jednotliv.split(":");

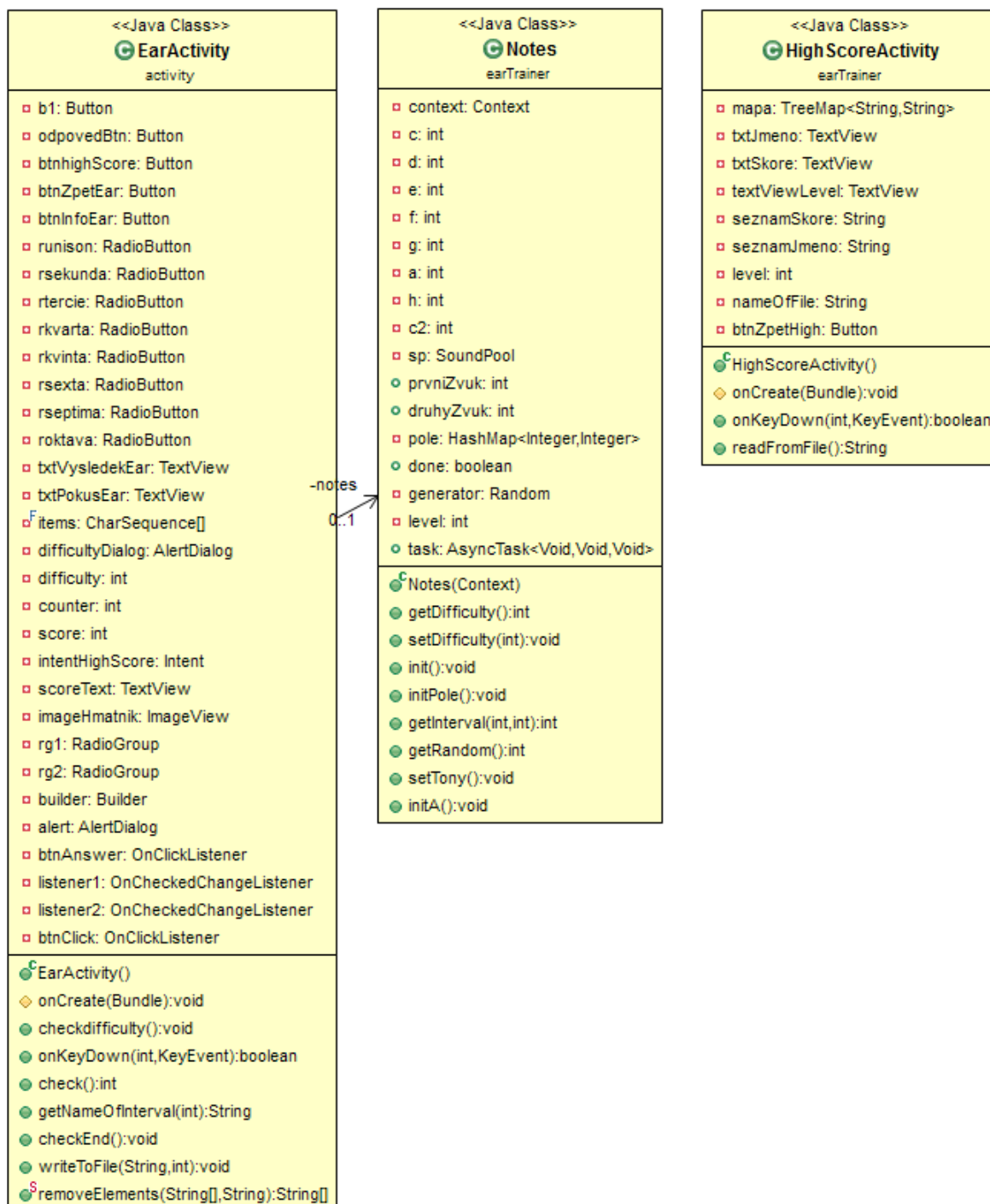
        while(mapa.containsKey(separatedJednotlive[0])){
            StringBuffer result = new StringBuffer();
            result.append(separatedJednotlive[0]);
            String mynewstring = result.toString();
            separatedJednotlive[0] = mynewstring + " ";
        }
        mapa.put(separatedJednotlive[0], separatedJednotlive[1]);
    }

    for (Entry<String, String> entry : mapa.entrySet()) {
        String key = entry.getKey();
        String value = entry.getValue();
        if (seznamJmeno == null) {
            seznamJmeno = value + "\n";
        } else {
            seznamJmeno = seznamJmeno + value + "\n";
        }
        if (seznamSkore == null) {
            seznamSkore = key + "\n";
        } else {
            seznamSkore = seznamSkore + key + "\n";
        }
    }
    txtJmeno.setText(seznamJmeno);
    txtSkore.setText(seznamSkore);
}
else{
}
```

Výpis kódu 13:

Algoritmus pro načítání hodnot skóre (Zdroj: autor)

6.4.5 Class Diagram



Obrázek 24:

Class Diagram komponenty EarTrainer (Zdroj: autor)

6.4.6 Obrazovka



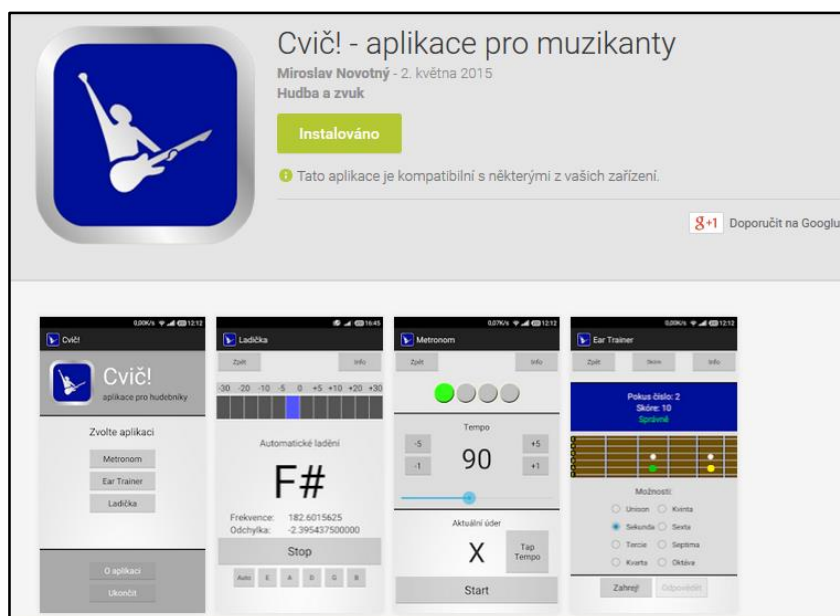
Obrázek 25:

Grafické uživatelské rozhraní komponenty EarTrainer (Zdroj: autor)

6.5 Publikace

2. května 2015 byla aplikace přidána na Google Play, kde prošla schvalovacím procesem a nyní je dostupná ke stažení zdarma. Na její stránce Google Play uvádí informaci o tom, že současná verze je podporovaná na 4698 modelech chytrých zařízení. V době odevzdání této práce ještě nebyla distributorem zpřístupněna možnost aplikaci ohodnotit.

Aplikace dostala jméno „Cvič! – aplikace pro muzikanty“.



Obrázek 26:

Stránka na Google Play věnovaná aplikaci (Zdroj: autor)

7 Závěr

Hlavním cílem práce bylo vyvinout funkční aplikaci, která poslouží jako cvičební pomůcka pro začínající i pokročilé hudebníky. Abych tohoto cíle dosáhnul, provedl jsem z počátku analýzu architektury operačního systému Android. Pro porozumění problematice spojené s digitálním zvukem jsem nastudoval proces digitalizace audiosignálu za pomoci Pulzně kódové modulace, kterou platforma Android při zaznamenávání zvuku využívá. Následně jsem provedl syntézu poznatků o jednotlivých rozhraních umožňujících práci se zvukem, které se v systému Android vyskytují. V další části byla provedena rešerše obdobných dostupných aplikací. Zbytek času jsem se pak věnoval návrhu a vývoji aplikace. Komponenta ladička byla po dokončení podrobena srovnání s jinou dostupnou aplikací pro ladění hudebních nástrojů.

Své snažení jsem završil zpřístupněním hotové aplikace v obchodu Google Play, kde je aktuálně volně ke stažení pod názvem „Cvič! – aplikace pro muzikanty“ v sekci „Hudba a Zvuk“. Podařilo se mi tak úspěšně splnit stanovený cíl praktické a nejtěžší částí této práce.

Vedle návrhu a vývoje samotné aplikace si práce kladla také vedlejší cíl – přiblížit problematiku oblasti zaznamenávání a zpracovávání digitálního zvuku, a to jak v obecné rovině, tak v prostředí systému Android. Tohoto cíle bylo dosaženo. Navíc došlo k rozšíření zadání. Z důvodu implementace funkcionality ladičky bylo nutné nastudovat a zpracovat problematiku spojenou s využitím spektrální analýzy za pomoci výpočtu Fourierových transformací a metody pro stanovení frekvence s nejvyšší amplitudou.

Výstupem práce je tedy kromě funkční aplikace také propracovaný teoretický základ, který jistě poslouží dalším vývojářům zajímajícím se o zaznamenávání a zpracovávání zvuku na platformě Android.

Stanovené cíle práce byly splněny.

Jako možnost budoucího rozšíření aplikace bych viděl návrh a implementaci další funkcionality pro podporu výuky hry. Nicméně z vlastních zkušeností mohu říci, že ani při použití sebelepší aplikace se hráč neobejde bez potřebné dřiny a odhodlání, které si hudební nástroj nárokuje.

Seznam literatury

- [1] YADAV, Srishti. *Global Smartphone OS Market Share Q3 2014* [obrázek]. [cit. 2015-02-14]. Dostupné z: <http://dazeinfo.com/2014/11/04/global-smartphone-os-market-share-q3-2014-android-ios-windows-report/>
- [2] BRACHMANN, Steve. 2014. *A Brief History of Google's Android Operating System* [online]. [cit. 2015-02-14]. Dostupné z: <http://www.ipwatch-dog.com/2014/11/26/a-brief-history-of-googles-android-operating-system/id=52285/>
- [3] VŠETEČKA, Roman. 2014. *Google dal Androidu TV vlastní set-top box. Nexus Player se čekal dříve* [online]. [cit. 2015-02-15]. Dostupné z: http://technet.idnes.cz/google-nexus-player-set-top-box-d37-/tec_video.aspx?c=A141016_085618_tec_video_vse
- [4] Security. *Android Source* [online]. [cit. 2015-02-22]. Dostupné z: <https://source.android.com/devices/tech/security/>
- [5] System and kernel security. *Android Source* [online]. [cit. 2015-02-22]. Dostupné z: <https://source.android.com/devices/tech/security/overview/kernel-security.html>
- [6] An Overview of the Android Architecture. *Techtopia* [online]. [cit. 2015-02-24]. Dostupné z: http://www.techtopia.com/index.php/An_Overview_of_the_Android_Architecture
- [7] ART and Dalvik. *Android Source* [online]. [cit. 2015-02-25]. Dostupné z: <https://source.android.com/devices/tech/dalvik/index.html>
- [8] About. *Google Play* [online]. [cit. 2015-02-27-]. Dostupné z: https://play.google.com/intl/ALL_us/about/overview/index.html
- [9] BERNAT, Petr. 2005. *Akustika, vznik a šíření zvuku, frekvenční analýza a syntéza, sluchový vjem zvukového signálu* [online]. [cit. 2015-03-11]. Dostupné z: http://homen.vsb.cz/~ber30/texty/varhany/anatomie/pistaly_akustika.htm
- [10] NÝVLT, Václav. 2006. *Boříme mýty o digitální kvalitě - kompaktní disk versus gramofon*. Technet [online]. [cit. 2015-03-11]. Dostupné z: http://technet.idnes.cz/borime-myty-o-digitalni-kvalite-kompaktni-disk-versus-gramofon-psx-/tec_audio.aspx?c=A060511_211913_tec_audio_NYV
- [11] ELSEA, Peter. 1996. *ANALOG RECORDING OF SOUND*. Electronic Music: University of California, Santa Cruz [online]. [cit. 2015-03-12]. Dostupné z: http://artsites.ucsc.edu/EMS/music/tech_background/te-19/teces_19.html

- [12] Analog versus Digital: Co je tedy lepší ? 2011. *Old School Electronic Music* [online]. [cit. 2015-03-12]. Dostupné z: <http://elektronicka-hudba.telo-tone.cz/clanky/analog-versus-digital>
- [13] ŠERÝCH, Jakub. *Spojité detail*. Wikimedia Commons [obrázek]. [cit. 2015-03-17]. Dostupné z: <http://commons.wikimedia.org/wiki/File:Spojité%20detail.png>
- [14] ŠERÝCH, Jakub. *Vzorkování*. Wikimedia Commons [obrázek]. [cit. 2015-03-17]. Dostupné z: <http://commons.wikimedia.org/wiki/File:Vzorkování%20A1n%C3%AD.png>
- [15] ŠERÝCH, Jakub. *Kvantování*. Wikimedia Commons [obrázek]. [cit. 2015-03-17]. Dostupné z: <http://commons.wikimedia.org/wiki/File:Kvantování%20A1n%C3%AD.png>
- [16] PTÁČEK, Ladislav. 2008. *Digitalizace zvuku*. Muzikus. 08(11).
- [17] KAŠPÁREK, Michal. 2014. *Vývoj záznamových zařízení XX: Digitální technologie: nástup jedničky a nuly*. In: Muzikus [online]. [cit. 2015-03-22]. Dostupné z: <http://www.muzikus.cz/pro-muzikanty-workshopy/Vyvoj-zaznamovych-zarizeni-XX-Digitalni-technologie-nastup-jednicku-a-nuly~03~zari~2014/?&mprint%5B4451%5D=1>
- [18] TOUŠEK, Jiří. 2011. *Moderní digitálně-analogové převodníky pro audio aplikace*. Plzeň. Diplomová práce. Západočeská univerzita v Plzni.
- [19] VRBA, Jiří. *Úvod do audiovizuálních technologií*. 1. vyd. Olomouc: Univerzita Palackého v Olomouci, 2008, 44 s. ISBN 978-80-244-2060-8.
- [20] KRAVAŘÍK, Jindřich. 2012. *STOPAŘŮV průvodce digitálním zvukem*. PIXEL. 12(04)
- [21] Media. *Android Source* [online]. [cit. 2015-04-01]. Dostupné z: <https://source.android.com/devices/media.html>
- [22] MediaPlayer. *Android Developers* [online]. [cit. 2015-04-01]. Dostupné z: <http://developer.android.com/reference/android/media/MediaPlayer.html>
- [23] SoundPool. *Android Developers* [online]. [cit. 2015-04-05]. Dostupné z: <http://developer.android.com/reference/android/media/SoundPool.html>
- [24] AudioTrack. *Android Developers* [online]. [cit. 2015-04-05]. Dostupné z: <http://developer.android.com/reference/android/media/AudioTrack.html>
- [25] MEMRUK, Ivan. *Intro to the three Android Audio APIs*. In: WiseAndroid [online]. [cit. 2015-04-05]. Dostupné z: <http://www.wiseandroid.com/post/2010/07/13/Intro-to-the-three-Android-Audio-APIs.aspx>

- [26] AudioRecord. *Android Developers* [online]. [cit. 2015-04-06]. Dostupné z: <http://developer.android.com/reference/android/media/AudioRecord.html>
- [27] MediaRecorder. *Android Developers* [online]. [cit. 2015-04-06]. Dostupné z: <http://developer.android.com/reference/android/media/MediaRecorder.html>
- [28] KOEGH, James. 2005. *Java bez předchozích znalostí: průvodce pro samouky*. Vyd. 1. Brno: Computer Press, 274 s. ISBN 80-251-0839-2.
- [29] ALLEN, Grant. 2013. *Android 4: průvodce programováním mobilních aplikací*. 1. vyd. Brno: Computer Press, 656 s. ISBN 978-80-251-3782-6.
- [30] PECINOVSKÝ, Rudolf. 2012. *Java 7: učebnice objektové architektury pro začátečníky*. Vyd. 1. Praha: Grada, 495 s. Knihovna programátora (Grada). ISBN 978-80-247-3665-5.
- [31] PAVLÍČKOVÁ, Jarmila a Luboš PAVLÍČEK. 2012. *Úvod do Javy: učebnice objektové architektury pro začátečníky*. Vyd. 1. Praha: Grada, 227 s. Knihovna programátora (Grada). ISBN 80-245-0963-6.
- [32] TOMÁŠEK, David. *Srovnání analogového a digitálního zvuku*. Materiály k předmětu 4ME240, Vysoká škola ekonomická v Praze [obrázek]. [cit. 2015-04-17]
- [33] REICHL, Jaroslav. *Složené tóny a Fourierova transformace*. Encyklopedie fyziky [online]. [cit. 2015-04-18]. Dostupné z: <http://fyzika.jreichl.com/main/article/view/187-slozene-tony-a-fourierova-transformace>
- [34] ANON, (Diskrétní) *Fourierova transformace*. Učebnice pro studijní obor Aplikovaná fyzika: Univerzita Palackého v Olomouci [online]. [cit. 2015-04-18]. Dostupné také z: <http://apfyz.upol.cz/ucebnice/down/mini/fourtrans.pdf>
- [35] BURK, Phil, Larry POLANSKY, Douglas REPETTO a Mary ROBERTS. *The FFT: Real, Imaginary, Magnitude, Phase*. Columbia University in the City of New York [online]. [cit. 2015-05-01]. Dostupné také z: http://music.columbia.edu/cmc/musicandcomputers/popups/chapter3/xbit_3_3.php

Terminologický slovník

Termín	Význam [zdroj]
Aktivita	Entita systému Android analogická k oknu klasické aplikace [27]
API	Aplikační programové rozhraní [přeloženo]
Balíček	Struktura, obsahující hierarchicky uspořádané skupiny tříd [28]
Buffer	Vyrovňovací paměť [přeloženo]
Bufferování	Proces načítání hodnot do vyrovnávací paměti [autor]
Class Diagram	Diagram tříd [přeloženo]
Čistá audio data	Nekomprimovaný zvuk v digitální podobě [autor]
Emulátor	Speciální software, který simuluje konkrétní zařízení [27]
Getter	Metoda, která vrací hodnotu přidruženého atributu [autor]
Handler	Objekt, který umožňuje komunikaci s vláknem běžícím na pozadí [27]
Implicitní komparátor	Předdefinovaný třídící algoritmus [autor]
Instant-messaging	Typ online komunikace, který dokáže přenášet zprávy v reálném čase [autor]
Kolekce	Datová struktura uchovávající více prvků stejného typu [29]
Kompilování	Proces převodu zdrojového do strojového kódu [autor]
Parser	Algoritmus, který dokáže analyzovat vstupní hodnotu a dle nastavených parametrů ji upravit do výstupu – nejčastěji se používá při zpracování textových řetězců [autor]
Plugin	Komponenta rozšiřující stávající program o novou funkcionalitu [autor]
Pool	Jiný výraz pro kolekci [autor]
Setter	Metoda, která nastavuje hodnotu přidruženému atributu [autor]
Stream	Přenos audiovizuálního materiálu [autor]
Threshold	Referenční hodnota [autor]
Virtuální stroj	Prostředí, ve kterém běží Java aplikace [29]

Seznam obrázků, tabulek a výpisů kódu

Obrázek 1: Zastoupení mobilních OS na trhu v třetím kvartále roku 2014 (Zdroj: [1])

Obrázek 2: Architektura systému Android (Zdroj: [4])

Obrázek 3: Srovnání analogového a digitálního zvuku (Zdroj: [33])

Obrázek 4: Detail spojitého signálu (Zdroj: [13])

Obrázek 5: Proces vzorkování signálu (Zdroj: [13])

Obrázek 6: Proces kvantizace signálu (Zdroj: [14])

Obrázek 7: Stavový diagram třídy MediaPlayer (Zdroj: [21])

Obrázek 8: MediaFramework (Zdroj: [22])

Obrázek 9: Stavový diagram třídy MediaRecorder (Zdroj: [27])

Obrázek 10: Use case diagram aplikace (Zdroj: autor)

Obrázek 11: Struktura projektu v prostředí Eclipse IDE (Zdroj: autor)

Obrázek 12: Základní menu aplikace (Zdroj: autor)

Obrázek 13: AudioTrack v situaci, kdy je tempo 60 úderů za minutu (Zdroj: autor)

Obrázek 14: AudioTrack v situaci, kdy je tempo 30 úderů za minutu (Zdroj: autor)

Obrázek 15: Class Diagram komponenty metronom (Zdroj: autor)

Obrázek 16: Grafické uživatelské rozhraní komponenty metronom (Zdroj: autor)

Obrázek 17: Reprezentace signálu vzorkovaného v čase (Zdroj: autor)

Obrázek 18: Signál ve frekvenční doméně (Zdroj: autor)

Obrázek 19: Metoda realForward() (Zdroj: dokumentace ke knihovně JTransforms)

Obrázek 20: Class Diagram komponenty ladička (Zdroj: autor)

Obrázek 21: Grafické uživatelské rozhraní komponenty ladička (Zdroj: autor)

Obrázek 22: Vygenerovaný tón o frekvenci 880 Hz v programu Audacity (Zdroj: autor)

Obrázek 23: Sejmuté obrazovky při testování přesnosti ladění (Zdroj: autor)

Obrázek 24: Class Diagram komponenty EarTrainer (Zdroj: autor)

Obrázek 25: Grafické uživatelské rozhraní komponenty EarTrainer (Zdroj: autor)

Obrázek 26: Stránka na Google Play věnovaná aplikaci (Zdroj: autor)

Tabulka 1: Příklady nejpoužívanějších frekvencí, zdroj[34]

Tabulka 2: Srovnání knihoven pro výpočet FFT, zdroj[autor]

Výpis kódu 1: Ukázka manifestu (Zdroj: autor)

Výpis kódu 2: Metoda initClick() (Zdroj: autor)

Výpis kódu 3: Třída ClickAsync (Zdroj: autor)

Výpis kódu 4: Metoda getIndex() (Zdroj: autor)

Výpis kódu 5: Deklarace prvků pro předávání informace do hlavního vlákna (Zdroj: autor)

Výpis kódu 6: Metoda doInBackground() ve třídě Recorder (Zdroj: autor)

Výpis kódu 7: Metody getFreq() a getHlasitost() (Zdroj: autor)

Výpis kódu 8: Metoda byteToDouble() (Zdroj: autor)

Výpis kódu 9: Metoda determineFreq() (Zdroj: autor)

Výpis kódu 10: Metoda removeElements() (Zdroj: autor)

Výpis kódu 11: Metoda init() a initPole() ve třídě Notes (Zdroj: autor)

Výpis kódu 12: Metoda getInterval() (Zdroj: autor)

Výpis kódu 13: Algoritmus pro načítání hodnot skóre (Zdroj: autor)

Příloha 1. Kompletní Class Diagram

