TARTU ÜLIKOOL
MATEMAATIKA-INFORMAATIKATEADUSKOND
Arvutiteaduse instituut
Infotehnoloogia eriala

Priit Tiidla

Rütmituvastus Androidil

Bakalaureusetöö (6 EAP)

Juhendaja: PhD, Margus Niitsoo

Autor: ……. "……" mai 2013
Juhendaja: ……. "……" mai 2013

Lubada kaitsmisele
Professor: ……. "……" mai 2013

TARTU 2013

Sissejuhatus 3
1. Probleemi püstitus 4
1.1 Muusikateooria 4
1.1.1 Helivältus 4
1.1.2 Tempo 5
1.1.3 Taktimõõt 5
1.2 Platvormi valik 6
2. Helitöötlus 7
2.1 Heli salvestamine 7
2.1.1 Analoog-digitaal muundamine 7
2.1.2 Wave helivorming 9
2.2 Heli analüüs ja löökide tuvastus 9
2.3 Helianalüüs sagedusruumis 11
3. Andriodi platvorm ja valminud rakendus 13
3.1 Androidi erinevad versioonid 13
3.2 Ekraanisuurused 14
3.3 Holo teemakujundus 16
3.4 *Activity*'d ja *Fragment*'d 17
3.5 Lõimed 19
3.6 Valminud rakendus 20
Kokkuvõte 22
Rhythm detection on Android 23
Viited 24

Sissejuhatus

Nutitelefon on viimaste aastate jooksul kiirelt populaarsust kogunud. Tootjad on hakanud turule oma tippmudelite kõrvale järk-järgult tooma soodsama hinnaklassiga mudeleid, et kõik inimesed leiaks valikust endale sobiva. Nutitelefonide arvu kasv on ka tarkvaratootjaid pannud järjest rohkem panustama mobiilsetele platvormidele. 2013 aasta jaanuari seisuga oli Google'i rakenduste poes 800 000 ja Apple'i poes 775 000 rakendust [8]. Nende hulgast paistavad välja pigem meelelahutuseks või ajaviiteks mõeldud rakendused. Nutitelefonide kasutamine millegi õppimiseks on jäänud kahjuks tagaplaanile, mis jätab osa selle kergelt kaasaskantava seadme potentsiaalist kasutamata.

Autori eesmärgiks on luua Androidi platvormile rakendus, mis aitaks muusikahuvilisel inimesel harjutada rütmis püsimist. Rakendus võrdleb kasutaja plaksutatud või koputatud rütmi mingi teise rütmiga, mis on eelnevalt sisse mängitud kas tema enda, õpetaja või kellegi kolmanda poolt.

Töö esimene peatükk tutvustab valdkonda ja põhjendab Androidi platvormi valikut. Teises peatükis kirjeldatakse helituvastuse ja -töötlemise probleemide ja kasutatud algoritme. Kolmandas peatükis antakse ülevaade Androidi platvormile rakenduse loomisest ja esinenud probleemidest.

1. Probleemi püstitus

Rütmis püsimine on muusiku üks kriitilisemaid oskusi. Noote võib küll tunda, kuid kui neid mängida valel ajal, ei kõla see eriti musikaalselt. Loodava rakenduse kaudu on kasutajal võimalik arendada oma rütmitunnetust.

1.1 Muusikateooria

1.1.1 Helivältus

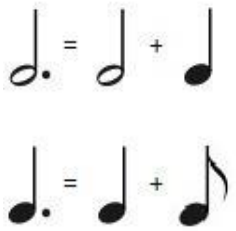
Helivältus ehk noodivältus on muusikaline parameeter, mis näitab heli relatiivset kestust. Noodiga märgitakse muusika noodikirjas üles helikõrgus ja helivältus. Selles töös heli kõrgusi aga ei uurita ja sellest tulenevalt võib mõelda siin kontekstis noodi all vaid helivältust. Heli puudumise relatiivset kestust tähistab noodikirjas paus. Noodi põhiväärtused on matemaatilises suhtes üks kahele.

Joonis 1. Noodid

Iga järgmine vältus on eelmisest kaks korda lühem. Üks täisnoot on kaks poolnooti ehk neli veerandnooti. Kui kaks või rohkem nooti, millel tavaliselt on lipud (kaheksandiknoot ja lühemad), järgnevad üksteisele, võib lipud asendada joonega (Joonis 2). Noodid ühendatakse joonega vaid juhul, kui nad asuvad ühes ja samas taktis.

Joonis 2. Noodivältused

Noodi vältuse pikendamiseks kasutatakse noodi järele kirjutatud punkti. Punkt noodi järel pikendab noodi pikkust poole võrra põhivältusest. Näiteks punktiga poolnoodi pikkus on poolnoodi ja veerandnoodi pikkuste summa (Joonis 3).



Joonis 3. Punktiga noodid

1.1.2 Tempo

Muusika terminoloogias nimetatakse tempoks muusikateose kiirust. Tempo on muusikalise kompositsiooni tähtis element, sest see võib mõjutada teose meeleolu ja ka selle esitamise raskust. Tempo mõõtühikuks on löökide arv minutid (BPM – *beats per minute*). Seda mõistetakse nii, et kindel noot (sõltuvalt taktimõõdust, vaata 1.1.3) tähistatakse löögina ja neid lööke peab olema minutis teatud arv kordi.

Noot	Kestvus (sekundites)
Tervenoot	120/BPM
Veerandnoot	60/BPM
Kaheksandiknoot	30/BPM
Punktiga kaheksandiknoot	45/BPM

Tabel 1. Nootide kestvuse arvutamine sekundites [4]

Nootide pikkust sekundites saab teada, kui teha teisendus löökidest minutis sekunditesse löögi kohta (Tabel 1). Näide: Kui tempo on 110BPM, siis punktiga kaheksandiknoodi kestvus on 45/110 ehk 0,409 sekundit.

1.1.3 Taktimõõt

Takt on heliteose osa, mis kestab tavaliselt ühest rõhulisest löögist järgmiseni. Taktimõõt on meetrilise rütmi tähistusviis. Taktimõõt kirjutatakse kahe arvuga, millest:

- Ülemine näitab, mitu lööki on taktis.
- Alumine näitab, milline vältus vastab ühele löögile

Lääne muusikas kasutatakse kõige sagedamini taktimõõtu 4/4. Selles taktimõõdus on ühes taktis neli lööki ja lööki tähistab veerandnoot. Levinud taktimõõdud on ka 2/2, 2/4, 3/4, ja 6/8.

1.2 Platvormi valik

Iga tootja huvides on see, et tema toode jõuaks võimalikult paljude tarbijateni. Nii on see ka tarkvara puhul. Seetõttu on kõige kasulikum arendada rakendust just sellele platvormile, mida kasutab suurem osa nutitelefonidest.

2012 aasta lõpu seisuga on turule alles jäänud kaks suurt tootjat: Google ja Apple. Google'i Androidi kasutajate osakaal on aastaga tõusnud 19,7%, iOS'i turuosa on suurenenud 0,4% (Tabel 1). Muude platvormide hulka kuuluvad ka Windows Phone ja BlackBerry, kelle suhtes peab aeg näitama, kas suudetakse endale turuosa juurde võtta või ootab ees hääbumine.

	2011	**2012**
Android	48,7%	68,4%
Apple iOS	19,0%	19,4%
Muud	32,3%	12,2%

Tabel 2. Nutitelefonide platvormide globaalne turuosa [1]

Vaadates erinevate platvormide kasutajate osakaalu on selge, et mobiilset rakendust arendama hakates on mõistlik valik teha operatsioonisüsteemide Android ja iOS vahel. Autori jaoks oli tegelikkuses otsuse tegemine lihtne, sest tal oli olemas mõningane kogemus rakenduste arendamisel Androidile. Samas puudus autoril igasugune kokkupuude erinevate Apple'i iOS operatsioonisüsteemi kasutavate seadmetega. Androidi pluss autori jaoks oli ka see, et see platvormi kasutab programmeerimiskeelt Java, milles autor tunneb end kõige kindlamalt.

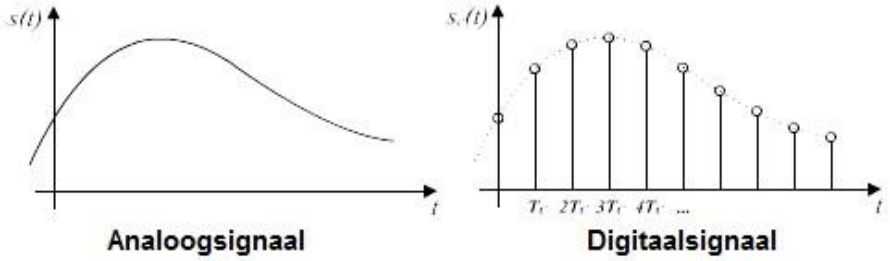
2. Helitöötlus

Heliks nimetatakse rõhulaineid, mis liiguvad läbi õhu või mingi muu meediumi ja mida on võimalik kõrvaga kuulda. Järgnevas peatükis kirjeldatakse kuidas saab analooghelist digitaalheli ja antakse ülevaate selle mõningastest töötlemisvõimalustest.

2.1 Heli salvestamine

2.1.1 Analoog-digitaal muundamine

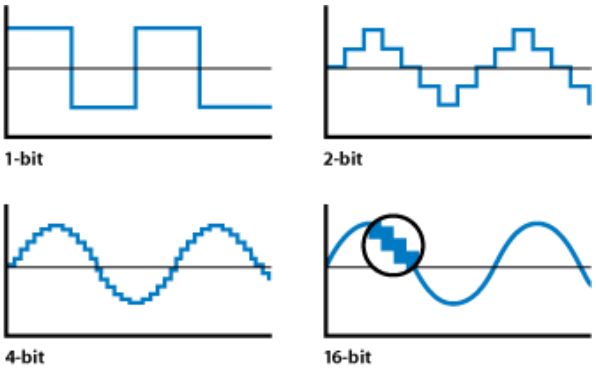
Kuna enamik analoogsignaale on pidevad (omavad piiramata hulgal erinevaid väärtusi piiramata hulgal ajahetkedel), siis selleks, et nendega oleks võimalik arvutil töötada, on need vaja muundada diskreetseteks (kindel väärtus kindlal ajahetkel). Pidevsignaali teisendamist diskreetsignaaliks nimetatakse diskreetimiseks ehk sämplimiseks.



Joonis 4. Sämplimine

Sämplimine ehk kvantimine on protsess, mille käigus võetakse analoogsignaalist iga teatud ajavahemiku järel näidised ehk sämplid. Sämplimissagedus [5] (*samping rate*) on suurus, mis määrab sämplimisel sämplite arvu sekundis. Sämplimissageduse mõõtühikuks on herts (Hz). Praktikas on kasutusel palju erinevaid sämplimissagedusi, mida kasutatake erinevatel eesmärkidel. Alates 8 000 Hz, mis on kasutusel telefonikõnede puhul kuni 2 822 400 Hz, mida kasutavad Sony ja Philips oma Super Audio CD tehnoloogia juures.

Nyquist–Shannoni teoreem väidab, et signaali täiesti õigeks taastamiseks tuleb selle salvestamisel kasutada sämplimissagedust, mis on vähemalt kaks korda suurem kui kõrgeim helisagedus, mida soovitakse salvestada. Kuna inimene on võimeline kuulma sagedusi vahemikus 20 – 20 000 Hz, siis muusikaga seotud rakenduste juures kasutatakse üldiselt sämplimissagedust 44 100 Hz (audio-CD), sest see on teoreemile vastavalt üle kahe korra suurem kui kõrgem sageus, mida inimene on võimeline kuulma.



Joonis 5. Bitisügavus

Bitisügavus [16] ehk kvantisatsioon on teine oluline parameeter signaali diskreetseks muundamisel. See näitab, kui mitme biti pikkust kahendarvu kasutatakse iga võetud sämpli kirjeldamiseks. Mida suurem on bitisügavus, seda täpsemini saab signaali hetkeväärtusi digitaalsel kujul esindada ning seda kvaliteetsem on taasesitatud või salvestatud heli. Nagu sämplimissageduse nii ka bitisügavuse juures on kasutusel mitmed erinevad standardid alates 8 bitist kuni 64 bitini. 8 bitti on piisav keskmise ja madala kvaliteediga heli jaoks, nagu telefonikõne või kassett. Üldiselt piisab audio salvestamisel 16 bitist. Sel juhul saavad sämplid endale väärtuse vahemikust -32,768 kuni 32,767.

Kuigi suurem sämplimissügavus ja bitisügavus annavad töötlemisel parema digitaalse signaali kvaliteedi, ei ole mõistlik praktikas kasutada ülemäära suuri väärtusi, sest see nõuab suurel hulgal salvestusruumi. Teades sämplimissagedust, bitisügavust ja kanalite arvu on lihtsasti võimalik arvutada pakkimata faili suurust. Bitikiirust (audio või video puhul ajaühiks salvestatav informatsiooni hulk bittides) on võimalik arvutada järgmise valemiga:

$$bitikiirus\ (bit/s) = bitisügavus \times sämplimissagedus\ \times helikanalite\ arv$$

Bitikiirus CD kvaliteediga heli puhul:

$$16\ bit \times 44\,100\ Hz \times 2 = 1411{,}2\ \ kbit/s$$

Ühe minuti CD kvaliteediga heli salvestamiseks kulub seega ruumi:

$$1411{,}2\ kbit/s \times 60\ s = 84\,672\ kbit$$

$$84\,672\ kbit\ \div 8 = 10\,584\ kB = 10{,}584\ MB$$

2.1.2 Wave helivorming

Rakenduses salvestatakse heli Wave helivormingusse. Wave ehk Waveform Audio File Format (tuntud ka kui WAV faili nimelaiendi .wav järgi) on Microsofti ja IBMi poolt loodud formaat, mida üldiselt kasutatakse Windowsi süsteemides pakkimata heli salvestamiseks. Kuigi WAV fail võimaldab hoida ka pakitud audiot, siis enamasti kasutatakse seda ikkagi kompresseerimata PCM formaadis (sellisel kujul andmete saamiseks viiakse läbi analoog-digitaalmuundamine, mida on kirjeldatud peatükis 2.1.1) andmete hoidmiseks. Wave vorming on muusikatöötlemises olulisel kohal, sest selles saab hoida CD-kvaliteediga heli. Wave formaadis hoitud heli on ka võimalik vastavate programmide abiga üsna lihtsalt töödelda ja muuta.

Wave vorming kasutab RIFF (Resource Interchange File Format) struktuuri, mis on mõeldud multimeedia failide jaoks. Sellest tulenevalt koosneb Wav fail päisest ja andmetest. Päis sisaldab andmeid faili sämplimissageduse, bitisügavuse, helikanalite arvu, andmete suuruse, andmete formaadi ja mõningate teiste parameetrite kohta.

Wave vormingu plussideks on suhteliselt lihtne struktuur ja võimalus salvestada kõrge kvaliteediga andmeid. Miinuseks on, et faili võimalik maksimaalne suurus jääb alla 4 GB, mis ei pruugi olla piisav, kui on vaja kasutada suuri sämplimissageduse ja bitisügavuse väärtusi või tahetakse salvestada väga pikka faili.

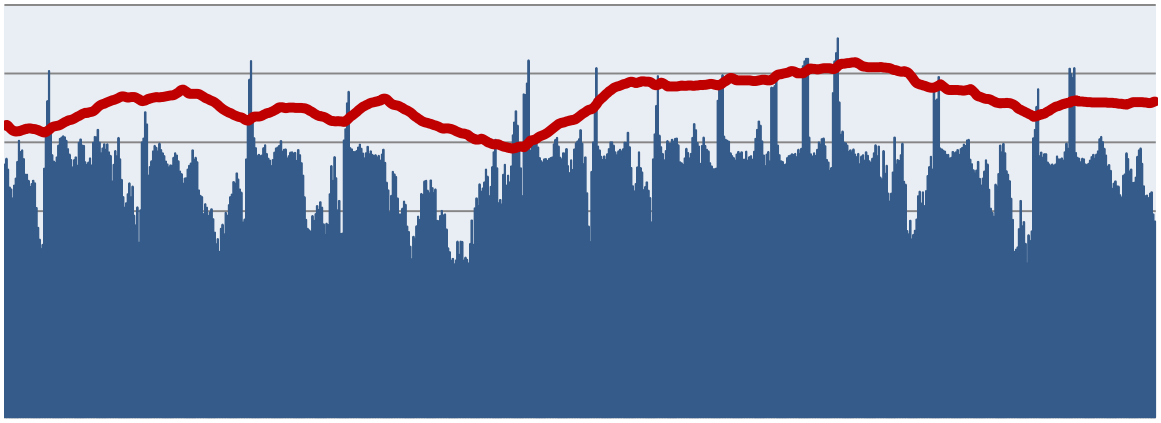
2.2 Heli analüüs ja löökide tuvastus

Digitaalsel kujul heli võib vaadata kui diskreetset signaali aegruumis. See tähendab, et signaali moodustavad ajaliselt üksteise järel olevad signaali väärtused ehk sämplid. Aegruumi võibki mõista kui signaali väärtuste muutumist üle aja. Selleks et saada teostada löögituvastust, tuleb esmalt signaali analüüsida ja leida energia, mida signaal omab teatud ajahetkedel.

Inimese kõrv saab helis olevast energiast aru niimoodi, et mida rohkem energiat heli omab, seda valjem ta tundub. Löögina tuvastatakse heli aga ainult siis, kui tema energia on ülekaalukalt suurem, kui talle eelneva heli energia ja inimese aju reageerib sellele suurele muutusele. Sellest tulenevalt saab väita, et kõrv saab muidu monotoonses helis olevatest energia kõrgpunktidest aru kui löökidest, aga kui mängida ühtlaselt valjut heli ei tuvasta ta ühtki lööki.

Muutuse helienergias saab leida arvutades välja keskmise helienerigia ja võrreldes seda heli hetke energiaga. Hetkeenergiana tähistame energia hulka, mis leidub viimases

1024 võetud sämplis. Seega võib hetke all mõista aega, mis on ligikaudu 23 ms (sämplimissagedusena on kasutusel standard 44 100 Hz). Keskmist helienergiat ei tohiks arvutada tervelt muusikapalalt, sest see võib sisaldada nii pingelisemaid kui rahulikumaid osi. Sellepärast tuleks hetkeenergiat võrrelda selle hetke ligidal oleva keskmise energiaga. Keskmise energia arvutamiseks võetakse 43 sämpliakent ehk 44 032 sämplit, mis on ligikaudu üks sekund. See on piisavalt suur aeg, et hetkeenergia ei mõjutaks keskmist energiat liiga palju ja piisavalt väike aeg, et mitte võtta arvesse loo mingis muus osas asuvat energiat.



Joonis 6. Helienergia muut. Punane joon tähistab konstandiga läbi korrutatud jooksvat keskmist

Võttes sämpliakna suuruseks 1024 sämplit, saab arvutada sämpliakna energia e järgmise valemiga:

$$e = \sum_{k=i_0}^{i_0+1024} S[k]^2$$

Järjend, mis sisaldab kõiki võetud sämpleid on tähistatud S-ga ja i_0 on positsioon, millest järgnevad 1024 sämplit vaadeldakse. Viimast leitud 43 sämpliakent hoitakse järjendis E, mida kasutades saab keskmise energia ühe sekundi kohta (EM) arvutada valemiga:

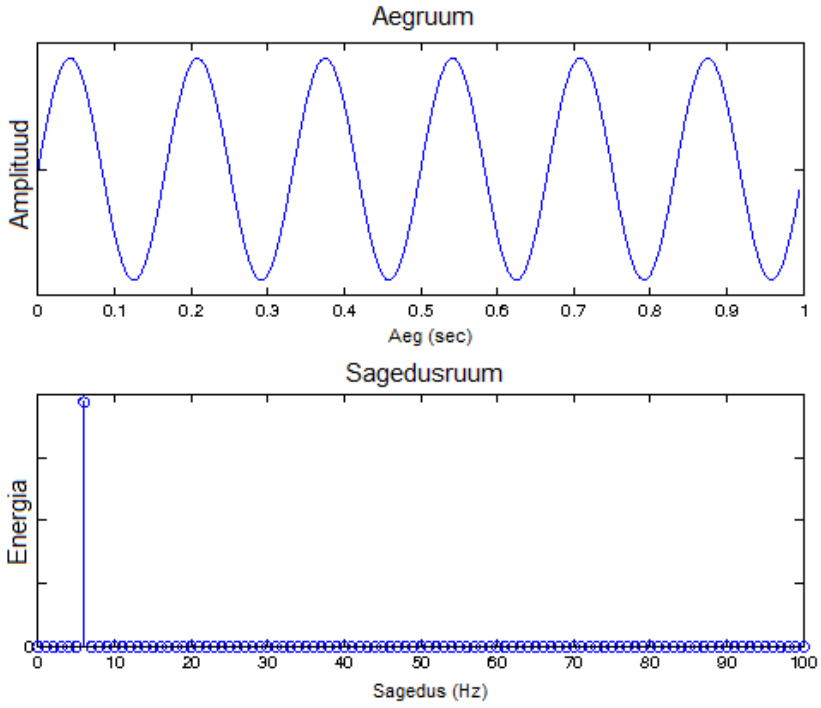
$$EM = \frac{1}{43} \times \sum_{i=0}^{43} E\,[i]$$

Kui keskmine energia on leitud tuleb see läbi korrutada konstandiga, et programm ei tuvastaks igat väiksemat helienergia muutumist löögina. Sobiva konstandi väärtuse määramine on selle algoritmi üks keerulisemaid probleeme. Kui panna konstandiks liiga kõrge väärtus, ei pruugi algoritm leida üles lööke juhul kui muusika sisaldab palju müra. Kui aga konstantiks on liiga madal väärtus, siis võib algoritm löökide hulka arvata ka kohti, kus tegelikult lööki polnud. Konstandiks sobivad üsnagi hästi väärtused, mis jäävad suurusjärku 1.3, kuid parim tuleb selle algoritmi puhul leida proovimise teel. Joonisel 6 oleval juhul on konstandi väärtuseks võetud 1,25.

Kui konstandiga korrutatud keskmine energia on leitud tuleb seda võrrelda parasjagu vaatluse all oleva sämpliakna energiaga. Kui sämpliakna energia on suurem kui leitud keskmine energia, võib seda mõista kui lööki, vastupidisel juhul aga löögiga tegemist ei ole.

2.3 Helianalüüs sagedusruumis

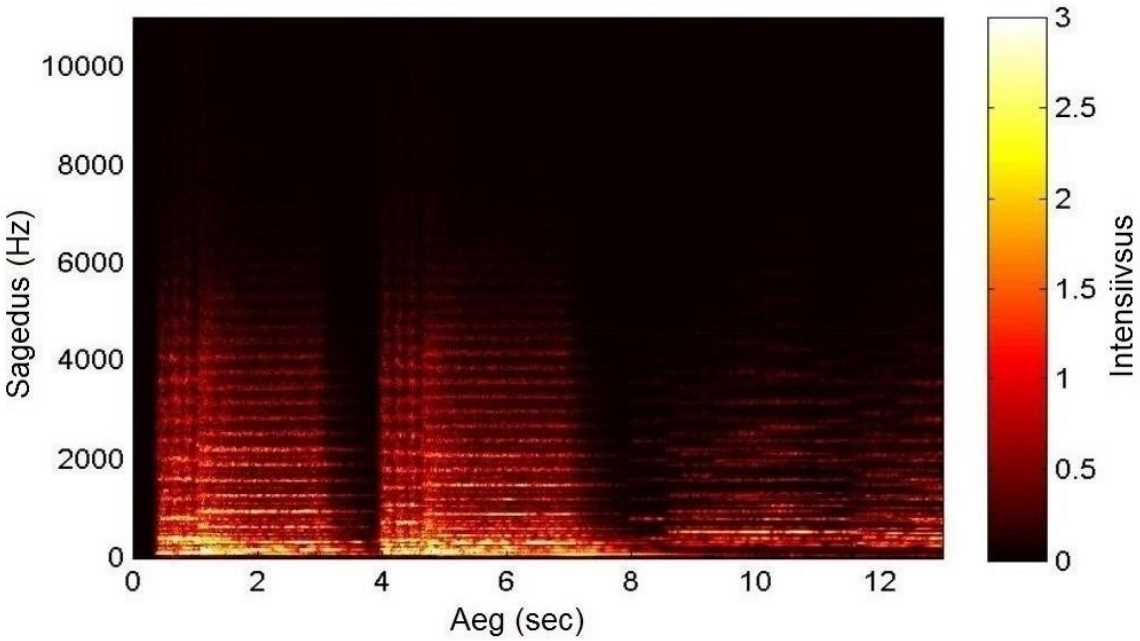
Eelmises punktis kirjeldati löökide tuvastust aegruumis. See pole aga ainuke võimalus heli analüüsi teostamiseks. Nimelt on võimalik helisignaali üle viia aegruumist sagedusruumi.



Joonis 7. Signaal aeg- ja sagedusruumis

Sagedusruumis saab leida iga sagedusevahemiku osakaalu signaalis, kuid signaali täismahus teisendamisel ei ole võimalik teada saada, mis ajahetkel mingi sagedus esines. Selleks et lisada esinenud sagedustele ka ajaline mõõde tuleb signaali teisendada sagedusruumi sämpliakende kaupa.

Löökriistadega mängitud nootide leidmiseks ei ole ilmtingimata vajalik helisignaali sagedusruumi teisendada, sest sellisel moel mängitud löögid väljenduvad mürana ega oma ühte kindlat sagedust. Kui aga tahetakse leida muude instrumentidega mängitud nootide helikõrgusi, on helisignaali sagedusruumi teisendamine möödapääsmatu.



Joonis 8. Helisignaali sageduspilt [15]

Sagedusruumis helisignaalist löökide leidmine omab sarnasusi sama toiminguga aegruumis. Mõlemal juhul tuleb vaadelda helisignaali hetkeenergiat. Kui aegruumis mõistetakse hetkeenergiana vaatluse all olevate sämplite amplituudide summat, siis sagedusruumis on tegemist mingi kindla sageduse intensiivsusega. Löök tuvastatakse, kui sämpliakna sagedusspektris tuvastatakse mingi kerkmisest kõrgema intensiivsusega sagedusemuutus. Tegu võib olla juba varem esinenud sageduse intensiivsuse muutumisega või mingi uue sageduse esinemisega sagedusspektris.

Signaalis esinevate sageduste uurimine jäi hetkel töö skoobist välja, kuid selle näol võiks olla tegemist töö ühe võimaliku edasiarendusega. Seda implementeerides oleks rakendus võimeline leidma lisaks noodipikkustele ka noodikõrgused.

3. Andriodi platvorm ja valminud rakendus

Androidi näol on tegemist hetkel kõige populaarsema nutitelefonide operatsioonisüsteemiga. Selles peatükis võetakse vaatluse alla mõningad punktid, mis on olulised Androidi platvormile rakenduse arendamise puhul.

3.1 Androidi erinevad versioonid

Hakates Androidi platvormile rakendust looma tuleb esmalt valida, milline on vanim tarkvara versioon, mida toetatakse. Uuem tarkvara pakub küll rohkem võimalusi ja uusi API-liideseid (*application programming interface*), kuid samas peaks jätkama ka vanemate versioonide toetamist.

Versioon	Koodnimi	Osakaal
1.6	Donut	0,2%
2.1	Eclair	2,2%
2.2	Froyo	8,1%
2.3 – 2.3.7	Gingerbread	45,6%
3.1 – 3.2	Honeycomb	1,3%
4.0.3 – 4.0.4	Ice Cream Sandwich	29,0%
4.1 – 4.2	Jelly Bean	13,6%

Tabel 3. Androidi versioonide jaotus veebruar 2013 seisuga

Versioon	Koodnimi	Osakaal
1.6	Donut	0,1%
2.1	Eclair	1,7%
2.2	Froyo	4,0%
2.3 – 2.3.7	Gingerbread	39,8%
3.1 – 3.2	Honeycomb	0,2%
4.0.3 – 4.0.4	Ice Cream Sandwich	29,3%
4.1 – 4.2	Jelly Bean	25,0%

Tabel 4. Androidi versioonide jaotus aprill 2013 seisuga (versioone, mis omavad jaotust alla 0,1% statistikas ei kajastata)

Statistikat Androidi erinevate versioonide kasutuse kohta uuendatakse iga kuu aja järel. Andmeid kogutakse jälgides seadmeid, mis külastavad Google Play Store rakenduste poodi. Statistika avalikustatakse iga kuu alguses Android Developers [2] veebilehel. Tabelitest 3 ja 4 võib näha, et kahe kuu jooksul on oluliselt vähenenud versioonide Froyo ja Gingerbread kasutajate hulka ja suurenenud Jelly Bean'i kasutajaskond, mis tähendab, et mobiilikasutajad on uuendanud tarkvara oma seadmes või vahetanud oma vana seadme uue vastu. 2013 aprilli seisuga omavad Android 2.2 ja sellest vanemaid versioone kokku veel 5,8% kasutajatest ja võib eeldada, et see number väheneb iga kuu. Seega võib väita, et praegusel hetkel Androidi rakendust arendama hakates oleks mõistlik võtta kõige vanemaks toetatavaks versiooniks 2.3, sest juba see kataks ligi 95% rakenduse võimalikust kasutajaskonnast.

3.2 Ekraanisuurused

Androidi operatsioonisüsteemi kasutavad väga paljud erinevad seadmed, millel lisaks muule on ka erinev ekraani konfiguratsioon. Ekraani konfiguratsiooni parameetrid on selle suurus ja tihedus. Android jagab erineva suuruse ja tihedusega ekraanid erinevatesse kategooriatesse (tabel 5), et nende toetamine oleks mõnevõrra lihtsam.

	ldpi	**mdpi**	**tvdpi**	**hdpi**	**xhdpi**	**xxhdpi**	**Total**
Small	9,5%						9,5%
Normal	0,1%	16,1%		37,9%	25,0%	0,8%	79,9%
Large	0,7%	2,7%	1,0%	0,5%	0,8%		5,7%
Xlarge	0,1%	4,6%		0,1%	0,1%	0,1%	4,9%
Total	10,4%	23,4%	1,0%	38,5%	25,9%	0,8%	

Tabel 5. Androidi ekraani suurused ja tihedused

Ekraani suuruse all mõeldakse ekraani füüsilist suurust, mida mõõdetakse mööda ekraani diagonaali. Suurused on grupeeritud neljaks: *Small*, *Normal*, *Large*, *Xlarge*. Tiheduse all mõistetakse pikslite hulka mingil ekraani osal. Tiheduse mõõtühikuks on dpi (*dots per inch* – pikslit tolli kohta). Nii nagu ekraani suuruste puhul, paigutatakse ka tiheduse puhul ekraanid nelja gruppi: *low(ldpi)*, *medium(mdpi)*, *high(hdpi)*, *extra high(xhpdi)*.

Selleks, et rakendus näeks ühesugune välja erinevate ekraani suuruste ja tiheduste juures, tuleks kasutajaliidest kujundades erinevad paigutused defineerida kasutades ühikut dp (*desnity-independent pixel* – tihedusest sõltumatu piksel). Üks tihedusest sõltumatu piksel on võrdne ühe füüsilise piksliga 160 dpi ekraanil. Seda arvestatakse baasina leidmaks paigutuse mõõtmed pikslites teistsuguse tihedusega ekraani puhul. Süsteem arvestab ekraani tegelikku tihedust ja skaleerib dp ühikud töö käigus jooksvalt ümber. Teades paigutuse mõõtmeid dp ühikutes ja ekraani tihedust (dpi), saab mõõtmed pikslites (px) leida konkreetse ekraani puhul valemiga:

$$px = dp \times (dpi \div 160)$$

Näiteks väga suure tihedusega (*xhdpi*, ~320dpi) ekraani puhul on 1 dp võrdne kahe füüsilise piksliga ekraanil.

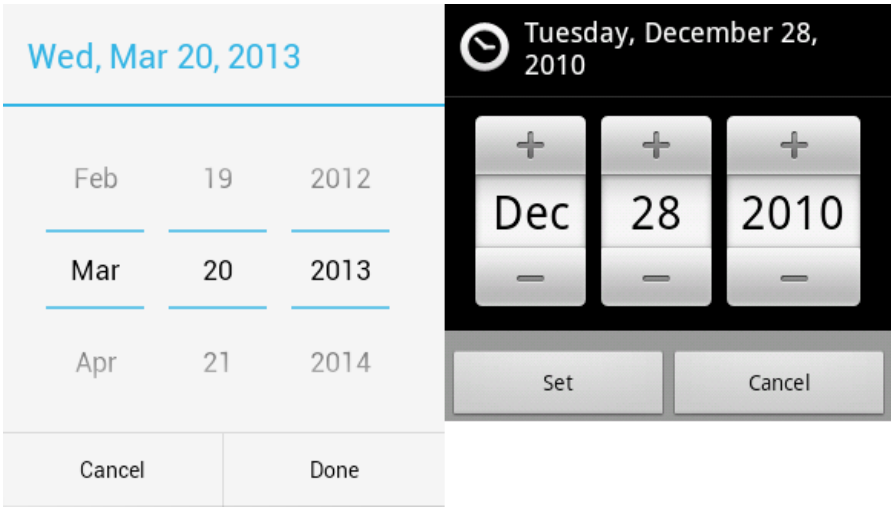
Androidi süsteemis on mõeldud selle peale, et kasutusel on paljud erineva suuruse/tihedusega ekraanid ja seetõttu teeb see enamuse tööst kasutaja eest ära, skaleerides kasutajaliidese elementide paigutused ise ümber. Selleks et erinevaid ekraani konfiguratsioone kombekamalt käsitleda, peaks rakenduse arendamisel silmas pidama mõningaid soovitusi:

- **deklaleerida Androidi manifesti failis, mis ekraanisuurusi rakendus toetab**
  Kui deklaleerida, mis ekraani suurusi rakendus toetab, saavad rakendust alla laadida vaid seadmed, mis vastavad nõudmistele. Toetatud suurusi saab deklaleerida, lisades Android manifesti faili elemendi <*supports-screens*>.
- **luua erinevate ekraani suuruste jaoks erinevad kasuajaliidese paigutused**
  Android muudab küll kujunduse suurust vastavalt kasutatava seadme ekraanile, kuid suuremate ekraanide puhul saaks mõnikord paremini ära kasutada lisaruumi ja väikse ekraani puhul tuleks olla kindel, et kõik mahub ekraanile ära. Erinevaid konfiguratsioone saab luua suuruste jaoks *small*, *normal*, *large* ja *xlarge*. Näiteks kasutajaliidese paigutusi kirjeldavad failid suurte ekraanide jaoks tuleks projektis paigutada kausta *layout-xlarge/*.
- **luua erinevate kraani tiheduste tarbeks erinevad pildifailid**
  Vaikeseadeid kasutades skaleerib Android projektis kasutatud pildifaile. Näiteks kui kasutatavad pildid on antud vaid keskmise tihedusega ekraani jaoks, siis skaleeritakse neid üles suure tihedusega ekraani puhul ja alla väikse tihedusega ekraani puhul. Selleks, et pildid näeks iga ekraani suuruse puhul head välja, tuleb luua erinevad pildid erineva tiheduse (*ldpi*, *mdpi*, *hdpi*, *xhdpi*) puhuks.

Näiteks kõrge tihedusega ekraanide jaoks mõeldud pildid peaks paigutama kausta *drawable-hpdi/*.

3.3 Holo teemakujundus

Rakenduse kasutajaliidese elementide loomisel kasutati Holo teemakujundust. Teemakujundused on Androidis andmaks ühesugust stiili kogu rakendusele. Stiiliga määratakse ära kasutajaliidese elementide visuaalsed omadused nagu näiteks värv, kõrgus ja fondi suurus. Defineeritud teemasi soovitatakse kasutada, et soodustada erinevate rakenduste visuaalset kokkukuuluvust ning parandada platvormi üldist kasutuskogemust. Vanematel Androidi versioonidel erinesid süsteemide kujundusteemad, mis raskendas ühesuguse kindla kujundusega rakenduse arendamist erinevatele versioonidele. Alates Androidi 4.0 versioonist on aga kõikidel seadmetel kasutusel Holo teema, mis lubab olla kindel, et kasutajaliides näeb samasugune välja kõikidel Android 4.0 või uuemat versiooni kasutavatel seadmetel. Lisaks parandab Holo teema oluliselt kasutajaliidese väljanägemist võrreldes varasemate versioonidega. Joonisel 9 on näha kuupäeva valimiseks kasutava *DatePicker*'i kujundus kasutades Holo teemat ja Android 2.1 kasutusel olevat kujundusteemat.

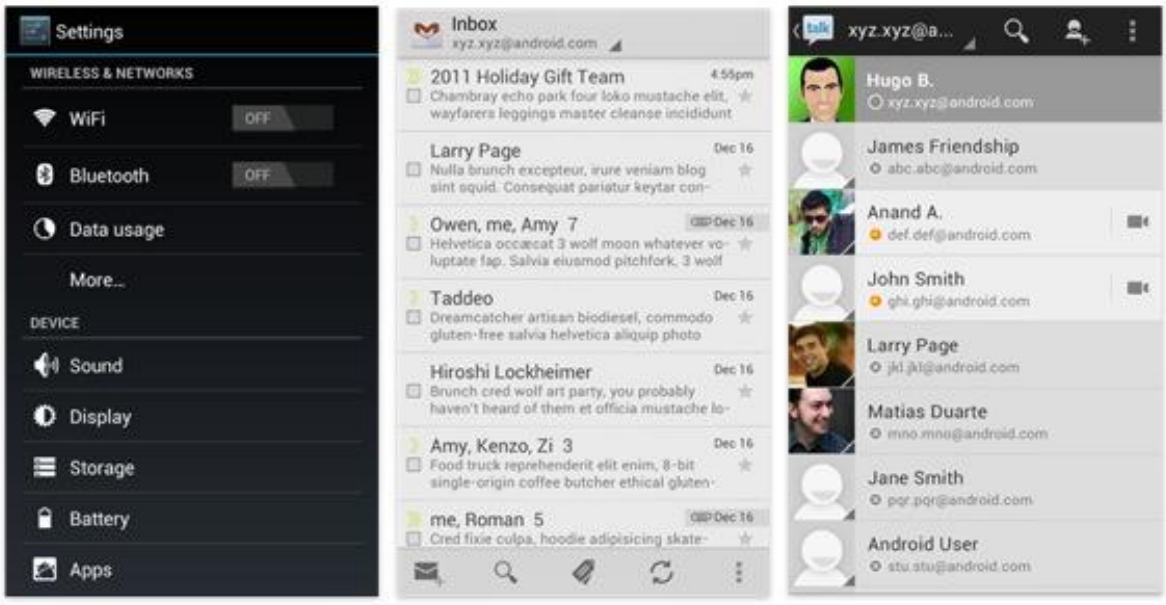


Joonis 9. DatePicker (vasakul Holo kujundusteema)

Valida on võimalik kolme Holo teema vahel: „Holo Light", „Holo Dark" ja „Holo Light with dark action bars" (Joonis 10). Kui tahta rakendusele anda teistest mõnevõrra erinevat väljanägemist, tuleks võtta baasiks ikkagi üks nendest kolmest teemakujundusest ja alustada kasutajaliidese kohandamist seda silmas pidades.

Holo teema kasutamise probleem seisneb selles, et see on küll standardina kasutusel seadmetes, mis omavad Androidi versiooni 4.0 või sellest uuemat, kuid vanemates seadmetes seda kasutusel pole. Kui soovitakse kasutada Holo teemat, kuid toetada samal ajal ka vanemaid versioone, tuleb kasutatavad teemad ja stiilid erivate Androidi versioonide jaoks eraldi defineerida Androidi projekti ressurssides.

Siiski on olemas võimalus kasutada Holo teemat ka vanemate Androidi versioonide juures. Selle jaoks tuleb projektis kasutada teeki HoloEverywhere [9], mida tehti ka loodava rakenduse puhul. Selleks, et kasutajaliidese elementidel rakenduks teegis olev Holo teema, tuleb kasutatavad klassid projekti importida teegist, mitte Androidi süsteemist. HoloEverywhere'i veebilehelt on võimalik teek vabalt alla laadida ja samuti on sealt leitavad õpetused ning näited teegi kasutamise kohta.



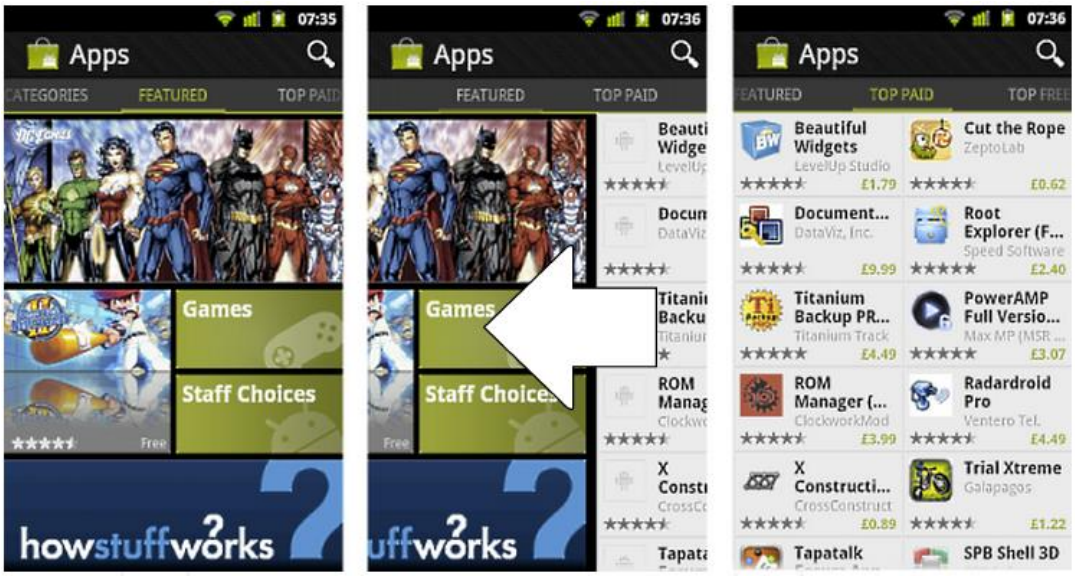
Joonis 10. Vasakult alates teemad Holo Dark, Holo Light, Holo Light with dark action bars [12]

3.4 *Activity*'d ja *Fragment*'d

Activity [10] on rakenduse komponent, mis kuvab ekraanile vaate, mille kaudu saab kasutaja teha selle *activity* poolt paikapandud tegevusi, milleks võivad olla näiteks helistamine, pildistamine või kaardi vaatamine. Tüüpiliselt on rakenduses üks „peamine" *activity*, mis käivitub siis, kui rakendus avatakse esimest korda. Iga *activity* võib omakorda käivitada järgmise *activity*, mis võib läbi viia muid tegevusi. *Activity*'d hoitakse LIFO (*last*

in, first out) tüüpi magasinis. Uue *activity* käivitamisel lisatakse see magasini ja peatatakse eelmise töö. Kui kasutaja vajutab seejärel nuppu „tagasi“, siis kustutatakse see *activity* magasinist ja naastakse uuesti eelmise *activity* juurde.

Fragment [11] esindab *activity* ühte kasutajaliidese osa. *Fragment* on kui *activity* osa, millel on oma elutsükkel ja mida saab lisada ning eemaldada sel ajal, kui *activity* on käimas. *Fragment* peab alati olema *activity* sees ja selle elutsükkel on otseselt sõltuvuses *activity* elutsüklist. Kui aga *activity* on käimas, on võimalik iga selle sees oleva *fragment*'iga toimetada teistest sõltumatult. *Fragment*'ide vahetuseoperatsiooni saab lisada magasini, mida haldab *activity*. Tänu sellele on võimalik „tagasi“ nupu vajutusel teostada vastupidine *fragment*'ide vahetusoperatsioon ja naasta eelmise *fragment*'i juurde.



Joonis 11. *ViewPager* [13]

Loodavas rakenduses on kasutatud süsteemi, kus on üks „peamine“ *activity*, mis tegeleb tema sees loodavate *fragment*'ide haldamisega. Tänu sellisele ülesehitusele saab rakenduse erinevate funktsioonide vahel navigeerimiseks kasutada *ViewPager*'it, mis on kasutusel näiteks Google Play rakenduses (joonis 11). *ViewPager* võimaldab *fragment*'ide vahel liikuda näputõmmetega (*swipe*). *ViewPager* laeb korraga ära lisaks vaadeldavale *fragment*'ile vaid sellest nii vasakul kui paremal paikneva *fragment*'i, sest suure *fragment*'ide arvu puhul võtaks kõikide elementide laadimine suurel hulgal ruumi ja toimuks liiga aeglaselt.

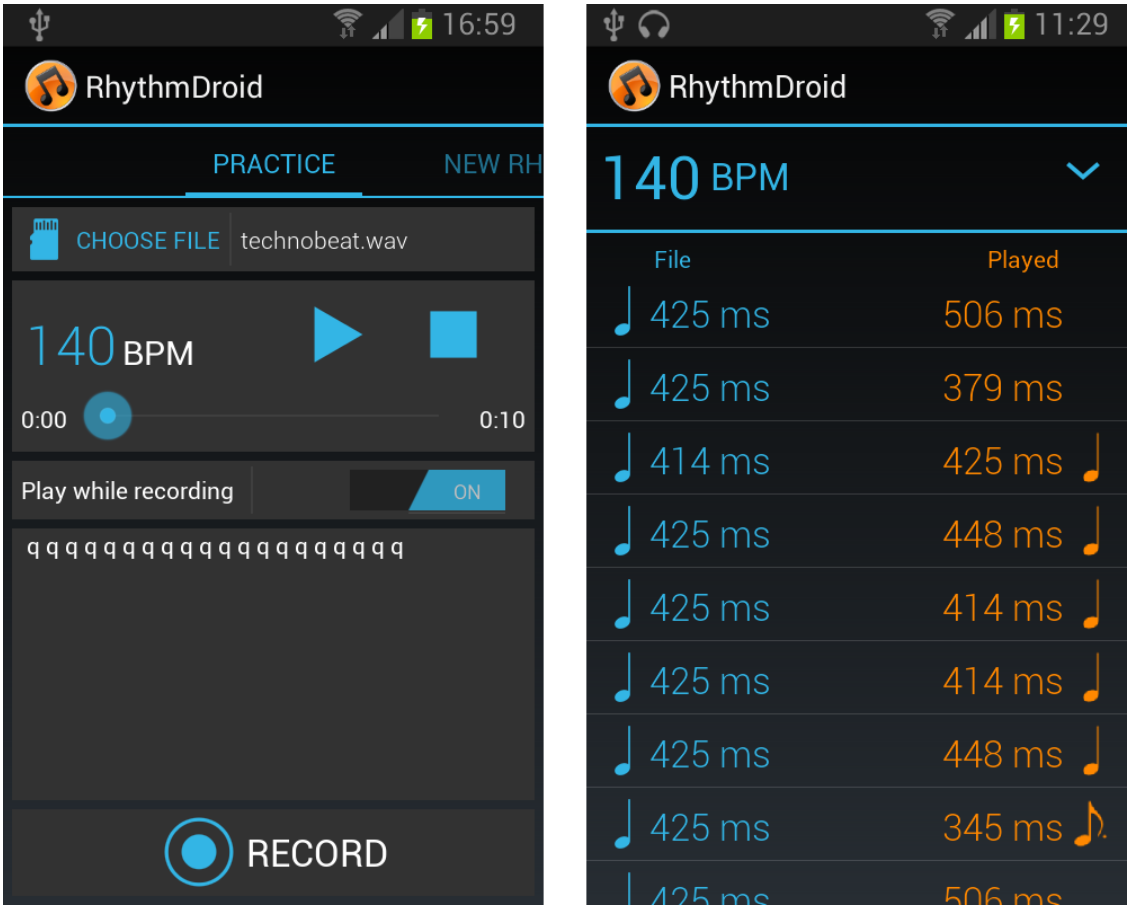
3.5 Lõimed

Mistahes Androidi rakenduse käivitamisel luuakse tema jaoks automaatselt *main* (peamine) lõim. Selle lõime poolt viiakse läbi kõik ühe rakenduse toimingud. *Main* lõime võib mõista ka kui kasutajaliidese lõime, sest see tegeleb kujunduse ekraanile kuvamise ja kasutaja sisendite töötlemisega. Selline ühe lõimega mudel tekitab aga probleeme, kui läbi on vaja viia ajaliselt pikemaid protseduure. Aeganõudvate protsesside täitmise ajal (nt võrgu poole pöördumine, andmebaasi päringud) blokeerub terve kasutajaliides, mis muudab rakenduse sel ajal kasutamatuks. Selle tõttu tuleb aeganõudvate operatsioonide läbiviimiseks luua uued lõimed. Rütmi tuvastamise rakenduses on lõimi kasutatud kolme tegevuse teostamiseks:

- heli salvestamine - lõime ülesanne on lindistada heli, luua uus fail ja ja seejärel salvestada sinna lindistatud heli. Lõim alustab tööd, kui kasutaja vajutab salvestamise alustamiseks ettenähtud nupule. Lõime töö katkestatakse, kui kasutaja otsustab lindistamise lõpetada, vajutades selleks ettenähtud nupule.
- heli taasesitamine – lõime ülesandeks on kasutaja poolt valitud faili mällu lugemine ja heli taasesitamine. Lõim alustab tööd, kui kasutaja vajutab heli taasesitamiseks ettenähtud nuppu. Lõime töö lõpetatakse, kui heli mängimisel ollakse jõutud faili lõppu või kui kasutaja katkestab heli esitamise.
- metronoom – lõime ülesandeks on iga veerandnoodi järel mängida metronoomi tiksu sarnast heli. Lõim alustab tööd koos heli salvestamise alustamisega ning lõpetab töö koos salvestamise lõpetamisega.

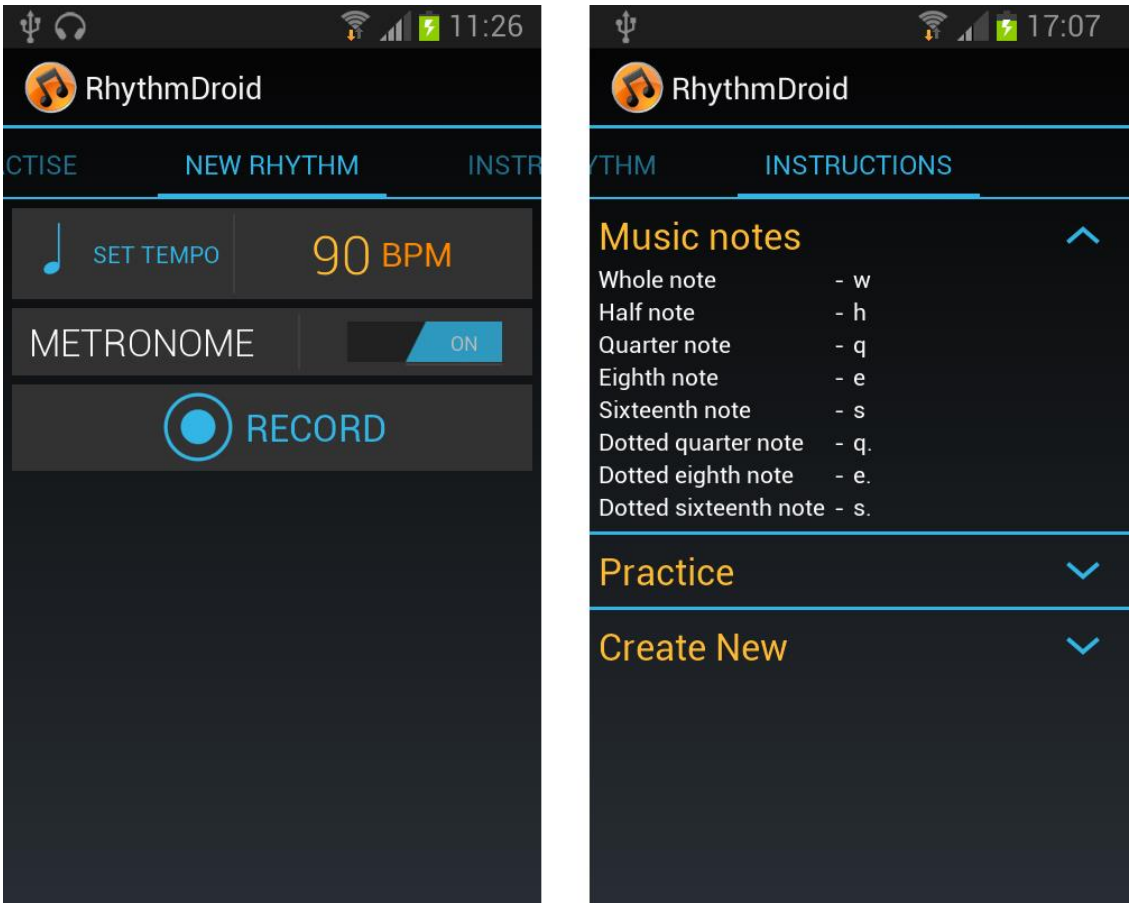
3.6 Valminud rakendus

Töö käigus valmis rakendus RhythmDroid. Rakenduse tuumiku moodustavad kolm lehekülge „PRACTICE“, „NEW RHYTHM“ ja „INSTRUCTIONS“, mille vahel on võimalik liikuda näputõmmetega (*swipe*).



Joonis 12. "PRACTISE" ja nootide võrdluse leheküljed

„PRACTICE“ leheküljel on võimalik valida audiofail eelnevalt telefoni salvestatute hulgast. Kuna rakendus võimaldab salvestada ja uurida ainult .wav faile, siis kuvatakse loendis ainult selles vormingus olevad failid. Valides mingi helifaili, püüab rakendus leida selle tempo ja selles olevad noodid. Kui kasutaja pole tempotuvastuse käigus saadud tulemusega rahul, saab ta tempot ka käisitsi muuta vajutades seda kirjeldava välja peale. Lehekülg võimaldab ka mahamängida valitud faili (toetatud ainult ühe kanaliga ehk mono heli). Heli kuulates ja ekraanile kuvatud noote jälgides saab kasutaja seda rütmi järgi mängida. Pärast lindistamise lõpetamist avaneb uus lehekülg, kuhu kuvatakse noodid, mis leiti valitud helifailist ja noodid, mis leiti kasutaja poolt lindistatud helis.



Joonis 13. Leheküljed "NEW RHYHTM" ja "INSTRUCTIONS"

„NEW RHYTHM“ leheküljel saab kasutaja luua uue rütmi, mis salvestatakse rakenduse kausta ning mida on võimalik seejärel „PRACTICE“ leheküljel valida. Selleks, et saada lindistada uut rütmi tuleb esmalt sisestada rütmi tempo. Võimalik on valida, kas lindistamise ajal mängitakse metronoomi tiksu või mitte (salvestamise ajal metronoomi või faili mahamängimisel on soovitatav kasutada kõrvaklappe, et heli ei segaks uut lindistust). Pärast lindistamist on võimalik salvestist kuulata ja näha salvestatud helist leitud noote. „INSTRUCTION“ leheküljel on antud noodi tähistused ja lühikirjeldus kuidas rakendust kasutada.

Rakendust on võimalik vabalt alla laadida internetiaadressilt https://dl.dropboxusercontent.com/u/50876718/RhythmDroid.apk.

Kokkuvõte

Antud bakalaureusetöö eesmärgiks oli luua rakendus Androidi platvormile, mis võimaldaks muusikahuvilisest kasutajal arendada oma rütmitunnetust. Rakendus lubab kasutajal lindistada muusikapala, millest seejärel proovitakse üles leida löögid. Samuti võimaldab rakendus valida juba varem kasutaja poolt lindistatud või telefoni salvestatud loo ja võrrelda seda kasutaja poolt seejärel lindistatud rütmiga.

Töös anti ülevaade muusikateooriast, mida on vaja tunda, et saada aru rakenduse tööst, kirjeldati helisignaali digitaliseerimist ja meetodeid, mis on vajalikud muusikast löökide tuvastamiseks. Lisaks kajastati töös ka mõningaid rakenduses kasutatud Androidi platvormi tehnoloogiaid ja toodi välja mõned punktid, millega tuleks arvestada nimetatud platvormile rakendust luues.

Rakenduse loomise käigus jõuti arusaamale, et valitud meetod muusikast löökide tuvastamiseks ei ole kõige parem. See töötab korrektselt, kui muusikas praktiliselt puudub taustamüra, kuid juba küllaltki vähese müra korral ei pruugi meetod tagada korrektset tulemust.

Rhythm detection on Android

Bachelor's thesis

Priit Tiidla

Summary

The purpose of this thesis was to create a software application that helps people who are interested in music to improve their sense of rhythm. Since the number of people who own a smartphone is on the increase, it was chosen that application will be created for one of the mobile operating systems. Android platform was chosen because it is the most popular and because author had previous experience of developing software for this platform.

Thesis explains the music notes and theory behind them that is needed to understand the working logic of created application. Also it explains the sound digitalization process and gives method for finding beats from audio. Other part of the thesis gives an overview of some of the Android classes and libraries that were used in created application. Also thesis provides some guidelines that should be taken into account when developing application for Android operating system.

For finding beats in audio, method was chosen that finds peaks in sound energy. During the development of application it became clear that this method that was not reliable in every occasion. It finds sound peaks without a problem when there is not any noise present in audio signal, but it struggles to find peaks when there is even slight noise in the audio.

Viited

[1] Strategy Analytics: Android claimed 70 percent of world smartphone share in Q4 2012 http://www.engadget.com/2013/01/29/strategy-analytics-android-70-percent-share/ (29.01.2013)

[2] Android Developers. Platform Versions. http://developer.android.com/about/dashboards/index.html (05.04.2013)

[3] Android Developers. Supporting Multiple Screens http://developer.android.com/guide/practices/screens_support.html (05.04.2013)

[4] How to calculate note duration and Hertz from BPM http://bradthemad.org/guitar/tempo_explanation.php (01.05.2013)

[5] Vikipeedia. Sämplimissagedus. http://et.wikipedia.org/wiki/S%C3%A4mplimissagedus (25.03.2013)

[6] e-teatmik: ingliskeelsete info- ja sidetehnoloogia terminite seletav sõnaraamat http://www.vallaste.ee/index.htm?Type=UserId&otsing=5560 (25.03.2013)

[7] Frederic Patin. Beat Detection Algorithms http://www.flipcode.com/misc/BeatDetectionAlgorithms.pdf (21.02.2003)

[8] How Many Apps Are in Each App Store? http://www.pureoxygenmobile.com/how-many-apps-in-each-app-store/ (05.03.2013)

[9] Backport Holo theme from Android 4.2 to 2.1+ https://github.com/Prototik/HoloEverywhere#readme (22.04.2013)

[10] Android Developers. Activities http://developer.android.com/guide/components/activities.html (23.04.2013)

[11] Android Developers. Fragments http://developer.android.com/guide/components/fragments.html (23.04.2013)

[12] Android Developers. Themes http://developer.android.com/design/style/themes.html (1.05.2013)

[13] Android Developers Blog. Horizontal View Swiping with ViewPager http://android-developers.blogspot.com/2011/08/horizontal-view-swiping-with-viewpager.html (1.05.2013)

[14] Android Developers. Processes and Threads http://developer.android.com/guide/components/processes-and-threads.html (1.05.2013)

[15] Meinard Müller, Anssi Klapuri. A Music-oriented Approach to Music Signal Processing http://ismir2010.ismir.net/proceedings/tutorial_3_Muller-Klapuri.pdf (07.05.2013)

[16] Wikipedia. Audio Bit Depth http://en.wikipedia.org/wiki/Audio_bit_depth (12.05.2013)

Lihtlitsents lõputöö reprodutseerimiseks ja lõputöö üldsusele kättesaadavaks tegemiseks

Mina Priit Tiidla (30.05.1991)

1. annan Tartu Ülikoolile tasuta loa (lihtlitsentsi) enda loodud teose "Rütmituvastus Androidil", mille juhendaja on Margus Niitsoo,

    1.1. reprodutseerimiseks säilitamise ja üldsusele kättesaadavaks tegemise eesmärgil, sealhulgas digitaalarhiivi DSpace-is lisamise eesmärgil kuni autoriõiguse kehtivuse tähtaja lõppemiseni;
    1.2. üldsusele kättesaadavaks tegemiseks Tartu Ülikooli veebikeskkonna kaudu, sealhulgas digitaalarhiivi DSpace´i kaudu kuni autoriõiguse kehtivuse tähtaja lõppemiseni.

2. olen teadlik, et punktis 1 nimetatud õigused jäävad alles ka autorile.

3. kinnitan, et lihtlitsentsi andmisega ei rikuta teiste isikute intellektuaalomandi ega isikuandmete kaitse seadusest tulenevaid õigusi.

Tartus, **13.05.2013**