TARTU ÜLIKOOL
MATEMAATIKA-INFORMAATIKATEADUSKOND

Arvutiteaduse instituut
Informaatika eriala

Ats Vendik

Mängu arendus Android platvormil

Bakalaureusetöö (6 EAP)

Juhendaja: Margus Niitsoo

Autor: …… "……" mai 2012
Juhendaja: …… "……" mai 2012

Lubada kaitsmisele
Professor: …… "……" mai 2012

TARTU 2012

Sissejuhatus

Viimaste aastate jooksul on tõusnud nii nutitelefonide arv maailmas, kui ka nende arvutusvõimsus. Tegemist ei ole enam lihtsalt seadmetega, mille eesmärgiks on ainult helistada - keskmine nutitelefon võimaldab tänapäeval lehitseda veebi, teha pilte ning videoid, lugeda kaarti, kuulata muusikat ning mängida mänge. Nutitelefoni väike suurus ja kaal võimaldavad teda lihtsalt igal pool kaasas kanda ning seetõttu on nende kasutamine meelelahutusplatvormina kiiresti tõusmas.

Autori eesmärgiks on disainida ja implementeerida meelelahutuslik mäng ühele neist nutitelefonide platvormidest. Kombineerides kasutaja muusikakogumiku ning lihtsa *2D* mängu elemendid loodetakse luua edukas nišitoote väheküllastunud turule. Bakalaurusetöö piiratud mahu tõttu keskendub antud töö just mängu arendusprotsessi dokumenteerimisele ning tehtud tehniliste valikute ja otsuste põhjendamisele .

Töö esimene peatükk annab ülevaate projektis kasutatavatest tehnoloogiatest ning põhjendab nende valikut nende konkurentide ees. Teises peatükiks vaadeldakse Android platvormiga seonduvaid spetsiifilisi probleeme ning kirjeldatakse neile leitud lahendusi. Kolmanda peatüki eesmärgiks on pakkuda ülevaade muusikamängule vajalikest algoritmidest ning kirjeldada lühidalt võimalikke visualiseerimismeetodeid.

1 Probleemi püstitus

Iga tarkvaraprojekt vajab õnnestumiseks plaani, mis määraks projekti eesmärgid ning kasutatavad tehnoloogiad nende saavutamiseks. Eriti tähtis on see etapp mobiilsete rakenduste puhul - seda nii piiratud ressursside, kui ka platvormide ja nende versioonide mitmekesisuse tõttu.

1.1 Mängu ülevaade

Selle alampeatüki eesmärgiks on püstitada projekti üldplaan ning kirjeldada arendatavat projekti.

1.1.1 Üldplaan

Nutitelefonide mängude turg on küllaltki küllastunud, konkurents on suur ja "hallist massist" välja paista on üsna raske. Kõikides populaarsemates žanrides on mängude valik väga suur ning olgugi, et küllaltki kõrge protsent neist on madala kvaliteediga, takistavad nad ikkagi uue üksikarendaja turule sissemurdmist. Lisaks sellele on mobiilimängude hinnad võrreldes personaalarvuti omadega tunduvalt madalamad, seetõttu on levinud skeem, kus ühe suurema projekti asemel tehakse mitu väiksemat, lootuses et vähemalt üks neist toodab arendajale kasumit. See aga muudab väikearendajale turule sisenemise veelgi raskemaks.

Vähemalt üks pääsetee võib siiski leiduda - nišitooted. Leides vähetäidetud niši, millel on teoreetiliselt potentsiaali mõõdukalt populaarseks saada, on võimalik isegi üksik- ja väikearendajatel arvestatavat kasumit teenida. Sellest lähtuvalt sai valitud ka selle lõputöö eesmärk - teha mäng, mis on seotud mängija muusikakoguga.

See idee ei ole kindlasti unikaalne, muusikamänge leidub igal nutitelefoni platvormil, kuid enamasti on nende muusika koos rakendusega kaasa pandud ning seetõttu väga piiratud. Eelis kasutaja muusikakogu kasutamisel mängus on tohutu - see võimaldab mängu mängida erinevate maitsetega inimestel, säästab ruumi kaasapandud muusika arvelt ning pikendab aega, millal mäng tundub huvitavana. Ideele annab hoogu juurde veel asjaolu, et tänapäeval kasutatakse nutitelefone sageli pleieritena, mis tähendab, et tihtipeale on kasutaja muusikakollektsioon juba telefoni mälukaardil olemas.

1.1.2 Eeskujud

Antud projekti primaarseks eeskujuks on *Windows* ja *Zune Hd* platvormil välja antud mäng *Audiosurf* (joonis 1). Tegemist on suhteliselt sarnase ideega, kus mängu maailm genereeritakse kasutaja valitud muusikapalast ning mängija põhieesmärgiks on saavutada loo vältel võimalikult kõrge punktisumma. Selle jaoks on vaja vältida mängurajal leiduvaid takistusi, mis on genereeritud valitud loo rütmi järgi. Vahelduse pakkumiseks on võimalik valida mitme erineva režiimi vahel, mis lisavad baasmängule erinevaid väikseid detaile.

Joonis 1. Ekraanitõmmis mängust *Audiosurf*

Mängu raskuse määrab nii valitud režiim, kui ka lugu - valjemate osade ajal genereeritakse rohkem takistusi ning tõstetakse mängija liikumiskiirust, aeglasemad osad kulgevad rahulikus tempos ning võimaldavad mängijal hetkeks lõdvestuda. Tulemuseks saadud punktisumma on võimalik interneti teel globaalsesse andmebaasi üles laadida, kuid kuna süsteem on lihtsalt murtav, leidub seal väga palju valesid ning isegi võimatuid skoore.

Kokkuvõttes on *Audiosurf* ajaviitemäng - seda on võimalik mängida lõputult, olgu siis ajendiks parema tulemuse saamine või lihtsalt muusika ning genereeritud keskkonna nautimine. Selle töö raames valmiv mäng töötab samal põhimõttel, erinevuseks on mängu väljanägemine ning mängumehhaanika.

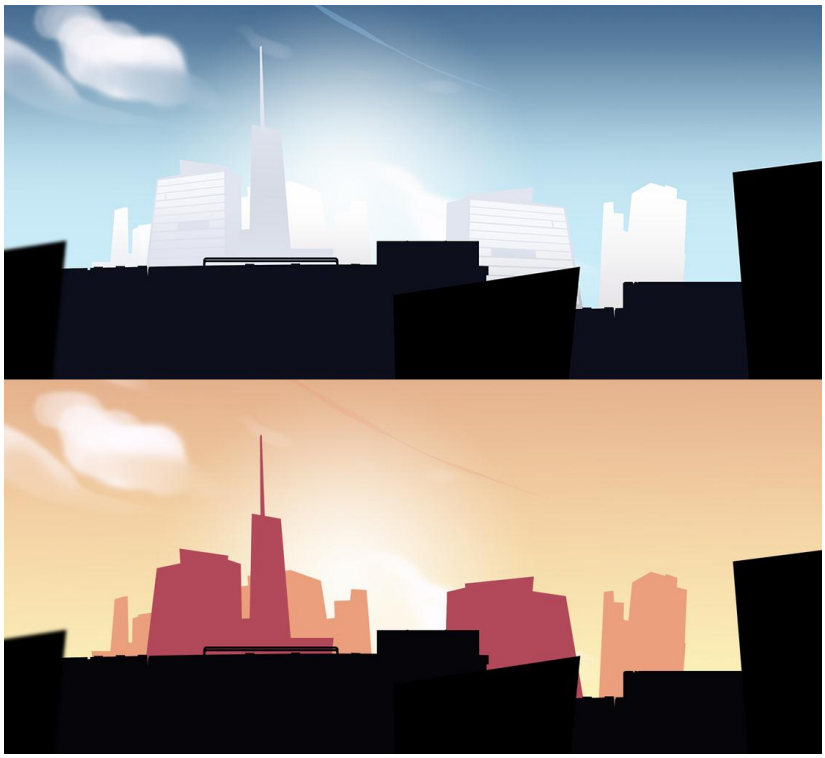
1.1.3 Mängu kirjeldus

Erinevalt *Audiosurf*-st ei ole antud projekt *3D* vaates, see lihtsustab graafika programmeerimist ja vähendab kunstnikul mängu objektide loomisele kuluvat aega. Sügavuse imiteerimiseks koosneb taust mitmest kihist, mis kõik liiguvad erineva kiirusega, andes nii edasi võltsi kolmedimensioonilise tunde. Kõik tagapool asuvad kihid on kergelt udused - see tugevdab *3D* illusiooni ja garanteerib, et mängijal on kergem tausta päris mängust eraldada.

Mängija eesmärgiks on sarnaselt *Audiosurf*-le koguda võimalikult suur punktisumma, selle jaoks peab ta õigeaegselt tabama rütmi, mis on genereeritud tema enda valitud loo abil. Mängija kontrollitud karakteri liikumiskiirus on seda suurem, mida valjem on antud hetkel valitud muusikapala helitugevus - see tagab et mängu intensiivsus kasvab käsikäes muusika omaga, pakkudes nii paremat elamust. Samuti sõltub muusikast maastiku kuju - vaiksetes kohtades liigub maastik ülesmäge, intensiivsuse kasvades maastik tasaneb ning eriti valjudes kohtades liigub hoopis allamäge. Lisaks sellele sõltub muusikast ka mängu dominantne värv, mis võib varieeruda külmadest toonidest, nagu sinine ja roheline soojade värvideni, nagu punane ja kollane (joonis 2).

Joonis 2. Kunstniku visioon värvi muutustest mängus

Mängu võib jagada kaheks osaks: aluskiht ja pealiskiht. Enamus tööst langeb aluskihile, mis tegeleb peaaegu kõigega: alustades ressursside lugemise ja muusika analüüsiga lõpetades maailma genereerimise ning joonistamisega. Aluskiht defineerib maastiku kuju (joonis 3), mängija kiiruse ja muusikarütmi järgi genereeritud elemendid, pealiskihi ülesandeks on aga nende abil kujundada mängurežiim. Selline jaotus võimaldab programmeerijal kergelt mängu uusi režiime lisada või olemasolevaid muuta. Näiteks saab disainida mängu, kus elementideks on takistused ja peategelase ülesandeks on kõiki takistusi vältida, hüpates neist üle, kaotades punkte iga ebaõnnestumise eest. Alternatiivselt võib kujundada režiimi, kus elemente tuleb hoopis koguda ning punktisumma määrab mängu lõpus õnnestumiste arv.



Joonis 3. Alternatiivne mängumaastik

Põhirežiimiks ongi elementide kogumise mäng, kus igale elemendile on eelnevalt määratud üks kolmest värvist, punane sinine või lilla. Et element üles korjata peab mängija enda karakteri värvi vastavaks muutma, seda on võimalik saavutada vajutades ekraani eri osadele. Ekraani vasak pool vastab sinisele, ekraani parem pool vastab punasele ning mõlemad pooled korraga on nende kahe segu ehk lilla. Lisaks värvile on elementidel ka muud omadused - näiteks annavad raskemini kättesaadavad elemendid rohkem punkte. Tavaliste elementide seas võib leiduda ka erinevaid ajalisi ja kohapõhiseid boonuseid - näiteks topelt punkti summa kolmeks sekundiks või kõikide elementide värvi muutmine

punaseks. Samuti asuvad rajal negatiivsed elemendid, mille efekt võib ulatuda eksitavate illusioonide tekitamisest kuni mängija täieliku kaotuseni.

Pikendamaks kasutaja huvi mängu vastu on mängus erinevad trofeed ning medalid. Neid on võimalik võita täites eelnevalt püstitatud ülesandeid, näiteks uue rekordi loomine või raskemat sorti loo ilma vigadeta läbimine. Lisaks sellele on võimalik mängijatel oma mängutulemused interneti edetabelisse üles laadida, see aga nõuab eelnevalt sisse logimist. Lugude tuvastamiseks on võimalik kasutada kas ebausaldusväärset silti muusikafaili küljes või mõnda internetipõhist tuvastamiseteenust nagu *MusicBrainz*. Esialgne prototüüp väliseid teenuseid ei kasuta, kuid vajaduse korral saab vastava toe mängule hiljem juurde lisada.

1.2 Platvormi valik

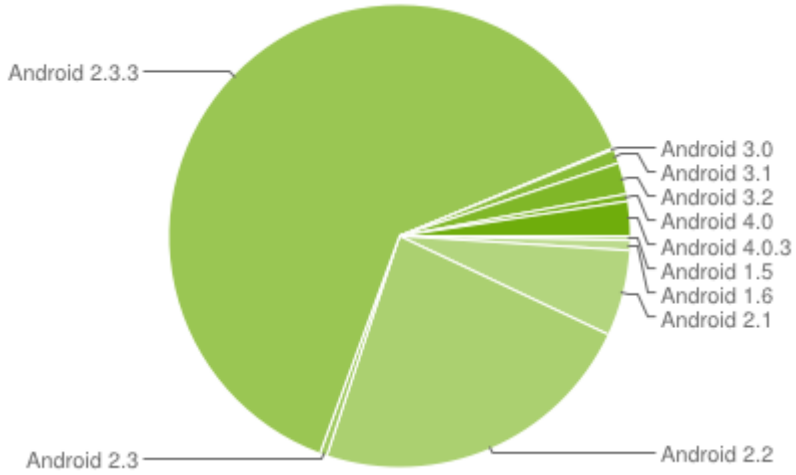
Mängu ning ka teiste tarkvaraprojektide arendamisel on tähtsal kohal kasutatavad tehnoloogiad. Enamasti on võimalik valida mitme erineva lahenduse vahel ning seega on vajalik optimaalse otsuse tegemiseks kaaluda nende kõigi positiivseid ning negatiivseid külgi.

2012 aasta esimese kvartali seisuga on nutitelefonide turul neli suuremat tegijat - nendeks on *Google*'i *Android*, *Apple*'i *iPhone*, *RIMi Blackberry* ning *Microsofti Windows Phone 7*. See aga ei tähenda, et neil kõigil oleks võrdne turuosa - Ameerika kohta tehtud statistika uuringu põhjal [1] on enimkasutatud platvormiks *Android* oma ligikaudu 50% turuosaga, teisel kohal on aga *iPhone* umbes 30% osakaaluga (tabel 1). Ülejäänud platvormide osakaal turul on marginaalne ning trend näitab, et see kahaneb tulevikus veelgi.

Tootja	November 2011	Veebruar 2012	Muutus
Google	**46,9%**	**50,1%**	**+3,2%**
Apple	**28,7%**	**30,2%**	**+1,5%**
RIM	**16,6%**	**13,4%**	**-3,2%**
Microsoft	**5,2%**	**3,9%**	**-1,3%**
Teised	**2,6%**	**2,4%**	**-0,2%**

Tabel 1. Nutitelefonide platvormide tootjate turujaotus Ameerikas

Eelnevat arvestades tuleks valik teha *Androidi* ja *iPhone*'i vahel, kuid enne otsustamist tuleb kaaluda ka muid platvormispetsiifilisi asjaolusid. Väga tähtsaks kriteeriumiks on seadmete mitmekesisus - erinevad operatsioonisüsteemide versioonid, protsessorikiirused, ekraanisuurused jne. Kõik need raskendavad oluliselt nii rakenduse loomist, testimist ja haldamist, kuna arvestama peab väga paljude parameetritega. Siinkohal on selge eelis *iPhone*'l - aktiivses tootmises on ainult kolm erinevat telefoni ning kaks tahvelarvutit. *Android* platvorm on aga fragmenteerunud (joonis 4) ning seetõttu arendajate seas halva kuulsusega.



Joonis 4. Märtsi lõpus 14 päeva jooksul *Google Playd* kasutanud seadmete operatsioonisüsteemide versioonid

Otsustavaks sai kokkuvõttes autori varasem kogemus - olles umbkaudu kaks aastat Androidile rakendusi arendanud tunneb autor end selles keskkonnas palju mugavamalt ning töötab seal efektiivsemalt.

Kuid ka *Androidi* versioone on mitmeid - uued versioonid täiendavad vanu ning lisavad uusi võimalusi, kuid vanu seadmeid ei ole tootjate tõttu alati võimalik uuendada - seega on kasulik valida minimaalne versioon, mis rahuldaks projekti nõudmised. Antud juhul on selleks Android 2.1, mis lisas toe mitme näpu tuvastamisele puutetundlikul ekraanil.

2 *Android* platvormi võimalused

Mänguarendus mobiilsetele platvormidele toob kaasa mitmed probleeme, mida teistel platvormidel ei pruugi leiduda. See peatükk põhjendab paari Android platvormiga seotud tehnoloogia valikut ning kirjeldab kuidas tehtud valikud programmi disaini mõjutavad.

2.1 Tehnoloogia valik

Antud alapeatüki ülesandeks on analüüsida erinevaid võimalikke arenduskeskkondi ja graafika teeke ning seejärel langetada tulemuste põhjal informeeritud otsus.

2.1.1 Arenduskeskkond

Android toetab kahte arenduskeskkonda - *Java* ning *C* / *C++*. *Java* on kindlasti rohkem levinud, paremini toetatud ning paljude arvates ka mugavam ja lihtsamini kasutatav. Samuti on internetist näidiste leidmine ja probleemide lahendamine tänu kasutajaskonna suurusele tunduvalt lihtsam. Teisest küljest leiduvad siin aga tavalised *Java* puudujäägid - väiksem jõudlus ning automaatne prügihaldus. Kuigi võib väita, et kumbki neist ei tohiks tavaliste rakenduste arendust oluliselt segada, jääb sellest mängudele ja arvutusmahukatele protsessidele nagu dekodeerimine selgelt puudu.

Õnneks on võimalik ka alternatiivne lähenemine *C* / *C++* kaudu. Valides selle tee peab aga loobuma paljudest mugavustest - staatiline süntaksianalüüs on palju puisem, kõik kolmanda osapoole teegid peab enamasti ise kompileerima, *debuggeri* käivitamine ja arusaadava *stacktrace*'i kättesaamine nõuavad palju lisavaeva ning mälu profileerijate kasutamisest võib praegu veel ainult unistada. Lisaks peaks mainima, et täielik *C* / *C++* tugi lisati alles *Android* 2.3 versiooniga, eelnevat on võimalik seda keskkonda kasutada ainult läbi *Java Native Interface*'i, mis tähendab, et väike osa programmist peab siiski olema *Javas* kirjutatud.

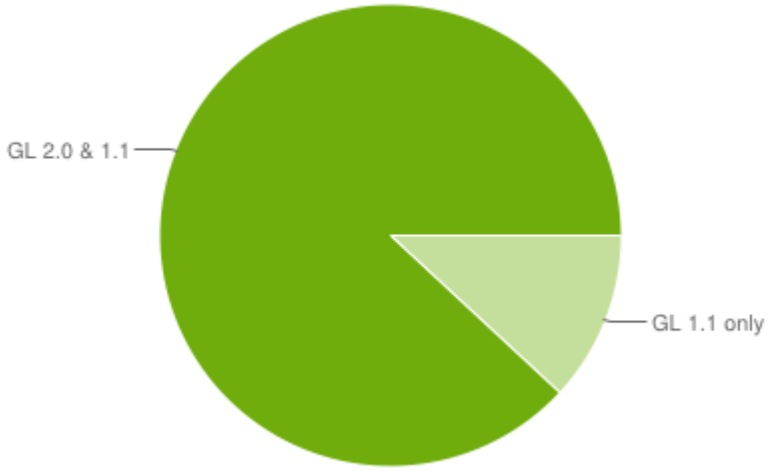
Kõik see tasub ennast aga ära - dekodeerimine kasutades *C* / *C++*'i on kuni paarkümmend korda kiirem, sama käib ka heli analüüsi ja muude sarnaste operatsioonide kohta. Suurimaks eeliseks ei ole aga mitte kiirus vaid hoopis mitme platvormi tugi - nimelt saab väikeste kohandustega kogu *C* / *C++* koodibaasi ka *iPhone*'i peal kompileerida. Arvestades ajakulu, mis kuluks kogu projekti ümberkirjutamiseks *Javast Objective C*'sse tundub *C* / *C++* keskkonna kasutamisele kuluv aeg hea investeeringuna.

2.1.2 Graafika

Kuna keskkond sai valitud pidades silmas mitme platvormi tuge, siis on loogiline, et ka graafika joonistamise teek peaks soovitud platvormidel toetatud olema. Siin on ainsaks valikuks *OpenGL ES*, aga nagu varemgi on ka sellel olemas mitu erinevat versiooni, mis erinevad nii võimaluste kui ka kasutamise poolest.

Vanemad *OpenGL ES* versioonid 1.0 ja 1.1 on toetatud pea kõikide *Android* telefonide poolt, nende kasutamine on OpenGL mõistes lihtne ning võimalikke probleemiallikaid on vähe. Kui arendaja on algaja või kui selle versiooni pakutud võimalustest piisab, siis ei ole mõtet 2.0 peale edasi liikuda, sageli on see aga vältimatu.

Versioon 2.0 on saadaval vaid uuemates ja kiiremates telefonides (joonis 5) ning on olemuselt oma eelkäijatest väga erinev. Kui traditsiooniliselt sisaldavad uued versioonid vanade võimalusi, siis antud juhul on seal tehtud suurpuhastus - kadunud on kõik sisseehitatud maatriksi matemaatika ning pinud ja lagipunkti- ning pikslivarjundajate kasutamine on kohustuslik. Seetõttu peab kasutama mõnda välist matemaatika teeki, näiteks *GLM* ning kirjutama enda varjundajad ka kõige lihtsamate operatsioonide joaks.



Joonis 5. Märtsi lõpus 7 päeva jooksul *Google Playd* kasutanud seadmete *OpenGL ES* versioonid

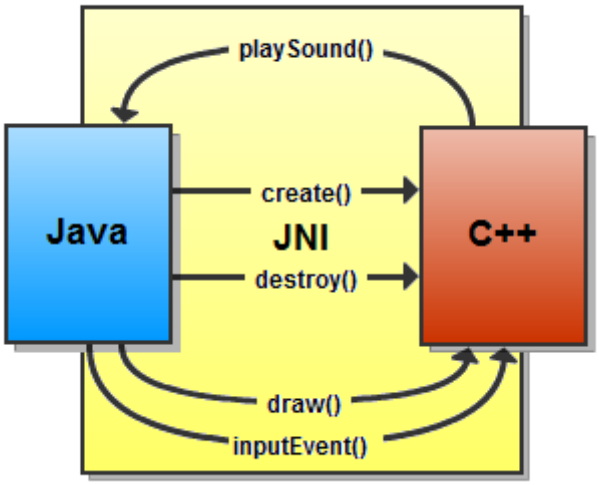
Kuna antud projekti jaoks on programmeeritavate piksli varjundajate olemasolu kriitiline, siis osutub valituks just *OpenGL ES* 2.0.

2.2 Programmi arhitektuur

Selle alapeatüki eesmärgiks on kirjeldada kuidas valitud arenduskeskkond ning *Android* platvorm programmi arhitektuuri disaini mõjutavad.

2.2.1 *Java* ja *C++* omavaheline suhtlus

Hoolimata sellest, et valitud programmeerimiskeeleks sai *C++*, ei ole võimalik *Java* keelt täielikult vältida. Kogu mänguspetsiifiline kood on kirjutatud *C++*'s, kuid valituks osutunud platvorm *Android* 2.1 nõuab, et programmi alguspunkt asuks *Javas*. See tähendab, et *Java* ülesandeks on nii programmi käivitamine, sulgemine, *OpenGL* konteksti loomine ning ka eelnevalt dekodeeritud helifailide mängimine (joonis 6).



Joonis 6. *JNI* sild *Java* ja *C++* vahel mängus

Java ja *C++* vaheline suhtlus on võimalik *Java Native Interface*'i ehk *JNI* abil. *JNI* on raamistik mis võimaldab *Java* virtuaalmasinas jooksval koodil kutsuda välja *C++* koodi ning ka vastupidi. Programmeerimise seisukohalt tähendab see üldiselt vajaminevate meetodite dubleerimist *C++* poolel, et need *Java* poolele nähtavaks teha ning nende kahe keele objektitüüpide omavahelist konverteerimist.

Selline jaotus osutub kasulikuks ka mitme platvormi toe lisamisel - kuna enamus platvormispetsiifilist koodi asub *Java* pool, piisab ainult selle osa ümberkirjutamisest soovitava platvormi peale ning päris mängu koodi muutma ei pea.

2.2.2 Lõimed

Iga *Androidi* rakendus, mis kasutab mingil kujul *OpenGLi* puutub oma eluea jooksul kokku vähemalt kahe olulise lõimega. Antud projekti puhul on lõimede arv aga natuke suurem (tabel 2) - see on tingitud vajadusest esitada dekodeeritud muusikafaile, mis on liiga suured et telefoni mällu ära mahtuda. Mida rohkem on lõimesid, seda keerulisemaks muutub sünkroneerimisvigade vältimine ning seetõttu on väga oluline aru saada, mis ülesanded neil lõimedel on ja kuidas täpselt nad üksteisega kokku puutuvad.

Lõime nimetus	**Väljakutse**	**Lõime ülesanne**
GLThread	Iga ettemääratud ajavahemiku tagant	Joonistab ühe kaadri, ainuke lõim kus saab *OpenGL*i kasutada
main	Kasutaja interaktsioon telefoniga	Teavitab programmi kasutaja tegevustest nagu näpuvajutused
PlayerThread	Kui vana helipuhver saab mängitud	Esitab ette antud puhvri helina
ReaderThread	Kui üks kahest helipuhvrist saab mängitud	Loeb failist eelnevalt dekodeeritud andmed puhvrisse
GC ja *HeapWorker*	Kui süsteemil nõuab rohkem mälu	Koristada mälust objektid, mida enam ei kasutata

Tabel 2. Mängu seisukohast olulised lõimed

Mängu vaatepunktist võib kõige olulisemaks lõimeks pidada joonistamislõime, mille nimeks antud juhul on *GLThread*. Eraldi lõime loomine joonistamiseks ei ole mängumaailmas just väga levinud tava, kuid *Androidi* puhul on kergem sellega lihtsalt leppida. See tähendab, et joonistamislõim on ainuke koht, kust on lubatud *OpenGL* joonistamiskäske välja kutsuda. Kaadri joonistamiseks kutsub joonistamislõim iga ettemääratud aja tagant välja joonistamismeetodi - seda tehakse tavaliselt iga 16.(6) ms tagant, kuid võib olenevalt seadmest ja tema koormusest erineda. Antud projektis kasutatakse joonistamislõime ka mänguloogika uuendamise jaoks, vältimaks lõime ebausaldusväärsusest tulenevaid ajastusvigu mõõdetakse iga eelneva kaadri jaoks kulunud

aega ning uuendatakse mängu sellest sõltuvalt. See tagab, et mäng kulgeb igas olukorras samas tempos ja et mängiv muusika on mängumaailmaga sünkroniseeritud.

Teiseks oluliseks lõimeks on kasutajaliideselõim ehk *main*. Selle kaudu teavitatakse programmi kasutaja toimingutest rakenduse kasutajaliidese ja nuppudega. Kui tegemist ei oleks mänguga, oleks kõige tähtsamaks lõimeks just kasutajaliideselõim, kuid antud juhul on kõige kergem kogu sellest lõimest saadud info lihtsalt joonistuslõime edasi suunata ja seda siis mänguloogika uuendamise ajal analüüsida. Nii on võimalik hoida minimaalsena koodi kriitiline osa.

Järgmised kaks lõime - *PlayerThread* ja *ReaderThread* - on olulised ainult antud konkreetsele projektile. Nende ülesandeks on koostööna lugeda eelnevalt mälukaardile salvestatud ja dekodeeritud fail ning seda katkematult taasesitada. See saavutatakse topeltpuhverdamise abil, mis on antud valdkonnas küllaltki standardne. Ideeks on kasutada kahte puhvrit, millest ühte mängitakse samal ajal kui teist täidetakse, vastavalt *PlayerThread*-i ja *ReaderThread*-i poolt. *Android* võimaldab enamus muusikafaile ka otse taasesitada, kuid see nõuaks eelnevalt dekodeeritud failiga võrreldes rohkem ressursse.

Viimaseks tähtsaks lõimeks, millega puutuvad kokku kõik *Java* rakendused, on prügikoristuslõim *GC*. See lõim ei saa põhjustada ühtegi sünkroniseerimisprobleemi, kuid tal on omadus kogu programm prügikoristusajaks seisma panna. See käib aga ainult *Java* poolt hõivatud mälu kohta, C++ puhul vastutab prügikoristuse eest programmeerija.

2.3 Platvormi kitsendused

Antud alapeatükis vaadeldakse paari *Android* platvormi poolt seatud piirangut.

2.3.1 Ressursihaldus

Enne rakenduse telefoni saatmist pakitakse see kokku *apk* failiks, mis iseenesest on lihtsalt ümbernimetatud *zip* fail. Sellesse faili pannakse kokku kõik projektis olnud ressursifailid ja vajaminev kood programmi jooksutamiseks. Enamasti pakitakse ressursifailid kokku, eranditeks on juba niigi kokkupakitud faili formaadid nagu *mp3*, *png* ja mõned teised (tabel 3). Kokkupakitud ressursifailidel on *Android* 2.1-s aga seatud teatud piirang - nimelt saab läbi *Android API* (*Application Programming Interface*) kasutada programmis ainult faile, mille suurus ei ületa ühte megabaiti. Selle vältimiseks on võimalik faili laiend ära vahetada mõne sellise vastu, mida arvatakse juba niigi kokku pakitud olevat.

Pildifailid	Videofailid	Helifailid
jpg, jpeg, png, gif	mpg, mpeg, mp4, m4v, 3gp, 3gpp, 3g2, 3gpp2, wmv	wav, mp2, mp3, ogg, aac, mid, midi, smf, rtttl, imy, xmf, m4a, amr, awb, wma

Tabel 3. Failiformaadid mida ei pakita [2]

Olukorda raskendab veelgi asjaolu, et *Android* 2.1-s ei ole võimalik *C++* poolt üldse ressursifailidele ligi saada. Sellele on kaks lahendust: kas avada ressursifail *Java* pool ning saata see baitide kaupa *C++* poole üle või kasutada mõnda kolmanda osapoole teeki ja kõik vajalikud failid *apk* failist mällu lahti pakkida. Mõlemal juhul tuleb saadud baidijada ise või mõne välise teegi abiga soovitud kujule viia, näiteks et *png* failid *OpenGLile* loetavaks muuta, tuleb kasutada välist *png* lugemiseks mõeldud teeki. Antud projekti jaoks sai valitud viimati mainitud lahendus, kuna see väldib eelnevalt kirjeldatud probleemi faili suurustega.

2.3.2 Erinevad ekraanisuurused

Android on tuntud oma laia valiku erinevate seda kasutavate riistvaraplatvormide s.o. telefonimudelite poolest. See võib küll olla meelepärane klientidele, kuid põhjustab arendajatele palju probleeme. Üheks neist on tohutult suur hulk erinevaid ekraanisuurusi ja resolutsioone, mis kõik vajavad eraldi testimist. Erinevalt personaalarvutitest ei ole ühel seadmel võimalik mitut erinevat resolutsiooni katsetada ning emulaatoril ei olnud kuni selle aasta aprillini võimalik ühtegi *OpenGL* 2.0 rakendust käivitada.

Olukorra leevendamiseks pakub Android omalt poolt paari erinevat tööriistakomplekti millest võib olenevalt rakenduse tüübist rohkem või vähem kasu olla. Üheks neist on võimas *Androidi* kasutajaliidese disainimissüsteem, mis võimaldab erinevatele ekraanisuuruste ja punktitihedusega seadmetele deklareerida erinevad kujundused. Sellest süsteemist saavad kasu lõigata aga vaid programmid, mis kasutavad *Androidi* poolt pakutavat kasutajaliidest.

Mängudele, mille ambitsiooniks on mitmeplatvormitugi, on kombeks oma kasutaja-liides ise *OpenGLi* abil valmistada, mis tähendab vähem tööd platvormide vahetamisel.

Mängude arendamisel oli kuni viimase aprillini programmeerija ainukeseks abi-vahendiks füüsiline *Android* telefon. Hiljuti väljalastud uue generatsiooni emulaator lisas aga toe mitte ainult *OpenGL* 2.0-le, vaid ka riistvara poolsele kiirendusele [3], mis

tähendab, et arendajatel on võimalik kõiki oma mänge reaalsetel kiirustel oma personaalarvutis testida. See tõotab omakorda parandada väljalastavate mängude kvaliteeti eriti väikearendajatel, kes tihtipeale ei suuda endale testimiseks suurt telefoniparki lubada.

Antud projekti raames valmiva mängu kasutajaliides on küllaltki lihtne ning seetõttu piisab paarist lihtsast strateegiast, et tagada hästi töötav lahendus suurel valikul erinevatel seadmetel. Kõige lihtsamaks ja küllaltki standardseks neist on sunnitud ekraani orientatsioon, mis tähendab, et mäng käivitub alati põiki vaates ning telefoni keeramine vaadet ei muuda.

Vältimaks ebaausat eelist suuremate resolutsioonidega telefoniomanikele peab nähtav mänguala olema igal seadmel ligikaudu sama suur. Et antud mängus liigub kasutaja eksklusiivselt vasakult paremale, siis piisab garantiist, et ainult mänguosa laius on alati sama suurusega, nähtava ala kõrgus mängu raskust oluliselt ei mõjuta ning võib seetõttu seadmelt seadmele varieeruda.

Sarnane lähenemine on võimalik ka menüüde juures, kuid selle erinevusega, et fikseeritud laiuse asemel on määratud kindel ekraani suhe, näiteks 16:9. Laiemate ekraanide puhul on siis näha rohkem tühja ala vasakul ja paremal ning kõrgematel ekraanidel üleval ja all, kuid alati on garanteeritud, et lubatud 16:9 ala on ekraanil täies mahus olemas. Kaks võimalikku tekkivat tühja ala on võimalik täita mistahes taustapildiga, mis läheks ülejäänud menüüga sujuvalt kokku.

Selline lähenemine tagab, et kõikidel kasutajatel on olenemata seadmetest mäng sama raskusega, säilitades samas kogu funktsionaalsuse. Erandiks võiks lugeda eriti suure ekraaniga seadmeid, näiteks tahvelarvuteid, mille puhul on heaks tavaks pakkuda kõrgema resolutsiooniga taustapilte ja spetsiaalselt disainitud menüüsid, mis lõikaksid suurest ekraanist rohkem kasu.

3 Muusika kasutamine mängus

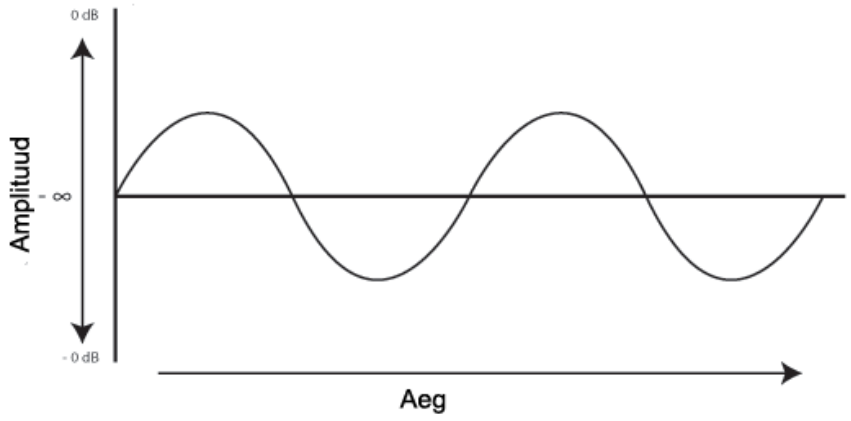
Muusika on antud projekti tähtsaks komponendiks, siin peatükis antakse ülevaade digitaalse heliga seonduvatest probleemidest, tema analüüsist ning visualiseerimisest.

3.1 Dekodeerimine

Antud alapeatükk kirjeldab lühidalt mis digitaalne heli on ning põhjendab tema kodeerimise vajadust ning meetodeid.

3.1.1 Digitaalne heli

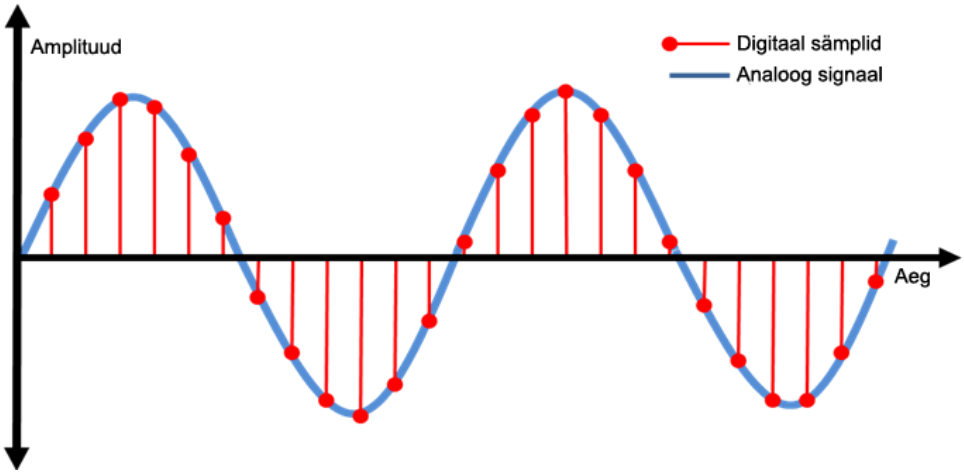
Helist võib mõelda kui õhu võnkumisest, mida on võimalik esitada õhurõhumuutuste funktsioonina üle aja (joonis 7). See tähendab, et selle funktsiooni abil saab iga ajaühiku kohta arvutada välja õhurõhu muudu mingi määratud keskmise rõhu kohta. Selle info alusel on võimalik elektromagneti ja kõlari membraaniga kõikvõimalikke salvestatud helisid taasesitada. Sarnaselt saab samade vahenditega suvalisi helisid ka salvestada.



Joonis 7. Helisignaal kui funktsioon üle aja [4]

Et heliga oleks arvutil võimalik töötada on vaja see enne digitaalsele kujule üle viia. Selle jaoks kasutatakse kvantimist ehk protsessi millega võetakse soovitavast helifunktsioonist iga eelnevalt määratud aja tagant näidised ehk sämplid (joonis 8) ning salvestatakse need väärtuste jadana andmekandjale. Võetud sämplite arvu sekundi kohta nimetatakse sämplimissageduseks ja tema ühikuna kasutatakse hertsi (Hz). Erinevatel tegevusaladel kasutatakse erinevaid sämplimissagedusi, mis võivad ulatuda 8000Hz-st, mida kasutatakse telefonides, kuni 2 822 400 Hz-ni mis on kasutusel *Super Audio CD*-s.

Nyquist–Shannoni teoreem väidab, et signaali perfektseks rekonstrueerimiseks peab sämplimissagedus olema vähemalt kaks korda suurem, kui signaalis kasutatav kõrgeim heli sagedus. Kõrgeim sagedus, mida inimkõrv tuvastada suudab on 20kHz ning seetõttu on enimkasutatud sämplimissagedusteks muusikafailides 44,1kHz ja 48kHz.



Joonis 8. Analoogsignaali digitaliseerimine [5]

Teiseks oluliseks parameetriks digitaalse heli juures on bitisügavus ehk kvantisatsioon, mis näitab kui mitut bitti ühe sämpli kirjeldamiseks kasutatakse. Erinevatel kasutusaladel on jällegi erinevad standardid, mis võivad varieeruda alates 8-st bitist kuni 64 bitini. Liiga väike kvantisatsioon põhjustab kvantisatsioonimüra, mis tõttu kasutatakse 8 bitti ainult siis kui kvaliteedil suurt tähtsust ei ole, näiteks telefonikõnede puhul. Praktikas piisab muusika salvestamisel 16 bitist kuna sellest tulev kvantisatsioonimüra on mikrofoni ja kõlarite poolt põhjustatud müraga võrreldes marginaalne. Tavaliselt on sämplid esitatud märgiga täisarvudega, mille vahemik ulatub 8 biti puhul -128-st kuni 127-ni, 16 biti puhul -32768-st kuni 32767-ni jne. Lisatäpsuse saamiseks on võimalik sämplid salvestada ujuvkomaarvudena mis on normaliseeritud vahemikku -1.0 kuni 1.0, kuid reaalne vajadus selle jaoks tavakasutajate seisukohast puudub.

3.1.2 Heli pakkimine

Teades helifaili pikkust, tema sämplimissagedust, bitisügavust ning kanalite arvu on väga lihtne välja arvutada vaadeldava faili suurus. Võttes loo mille pikkuseks on 180 sekundit, sämplimissageduseks on 44100Hz, biti sügavuseks on 16 bitti ja kanalite arvuks on 2 võib välja arvutada, et loo suurus megabaitides oleks 180 * 44100 * 16 * 2 / 8 / 1024 / 1024 = ~30,2(MB). Isegi tänapäevaste andmekandjate puhul väljuks sellisel kujul

muusika- või filmikollektsiooni suurus päris kiiresti kontrollidest, õnneks aga on antud probleem juba lahendatud helifailide pakkimise ehk kodeerimise näol.

Erinevaid algoritme helifailide pakkimiseks on mitmeid, kuid kõik need saab jagada kahte suuremasse kategooriasse: kadudega ning kadudeta algoritmid. Esimesed neist on tavakasutajate seas palju populaarsemad, seda enamasti nende väikese failisuuruse tõttu. Reegli kohaselt nõuavad sellisel kujul failid lahtipakkimiseks rohkem arvutusvõimsust kui kadudeta failid, kuid kuna tänapäevane tehnika on kas niigi piisavalt võimas või sisaldab vastavat riistvaramoodulit, siis on see aspekt oma tähtsuse kaotanud. Kadudeta failid on enamasti hinnas pigem muusikaentusiastide ja lindistamisega tegelevate professionaalide hulgas, kuid suured kõvakettad ja muud andmekandjad võimaldavad ka tavakasutajatel kadudeta lahendustele üle minna.

Nimetus	Tavaline faililaiend	Tüüp	Teek
MPEG Audio Layer I, MPEG Audio Layer II, MPEG Audio Layer III	.mp1, .mp2, .mp3	Kadudega	libmpg123
Vorbis	.ogg	Kadudega	libtremor
Free Lossless Audio Codec (FLAC)	.flac	Kadudeta	libvorbis
PCM Wav	.wav	Kadudeta	manuaalselt

Tabel 4. Mängu pool toetatud helifailide formaadid ja nende dekodeerimiseks kasutatavad teegid

Valides dekodeerimisteeki tuleb üldiselt teha valik mugavuse ning kiiruse vahel. Antud juhul oli võimalik valida *libffmpeg*, mis oskab lugeda pea kõiki inimesele tuntud heli- ja videofailiformaate ja mitme väikse kuid spetsiaalselt mobiilsetele platvormidele viimase piirini optimeeritud teegi vahel. Kuna kasutajad ei ole nõus mobiilimängu laadimise järgi kaua ootama, sai otsus langetatud just teise variandi kasuks (tabel 4). Dekodeerimiskiiruste varieerumise näitena võib tuua teegi *libmpg123*, millel kulus originaalsätetega 4 minutilise faili dekodeerimiseks 38 sekundit. Kompileerides teegi parameetritega, mis lubavad kasutada ainult täisarvude matemaatikat ning spetsiaalseid

Assembleri rutiine langes sama faili lahtipakkimiseks kulunud aeg umbes kolmele sekundile - peaaegu 13 kordne võit.

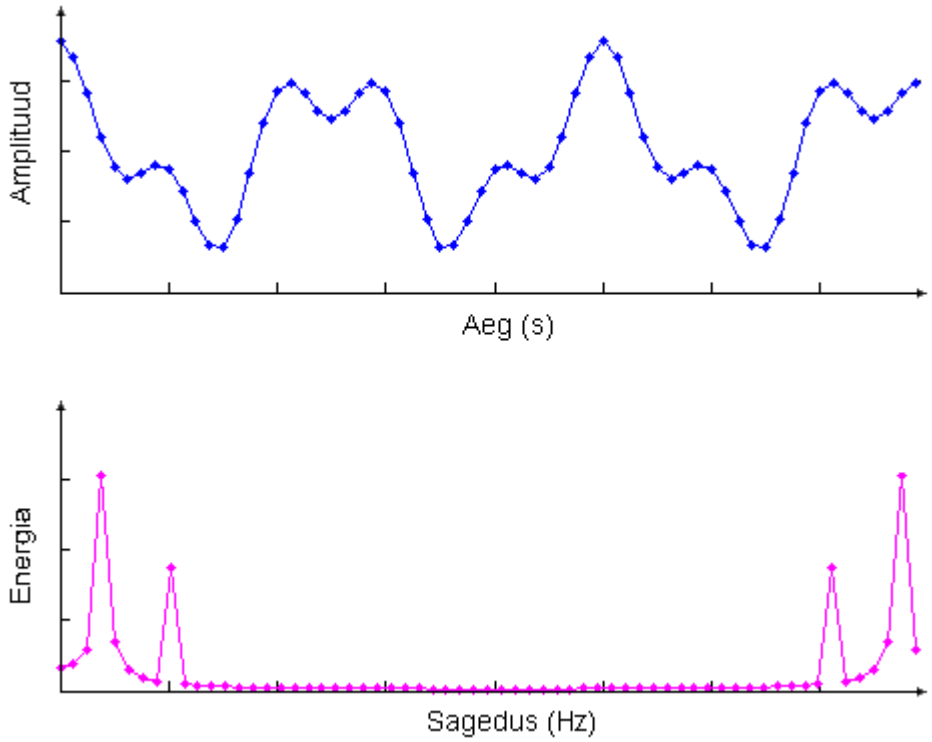
3.2 Heli analüüs

Siin alapeatükis kirjeldatakse antud projektis heli analüüsiks kasutatavat algoritmi.

3.2.1 Diskreetne Fourier' teisendus

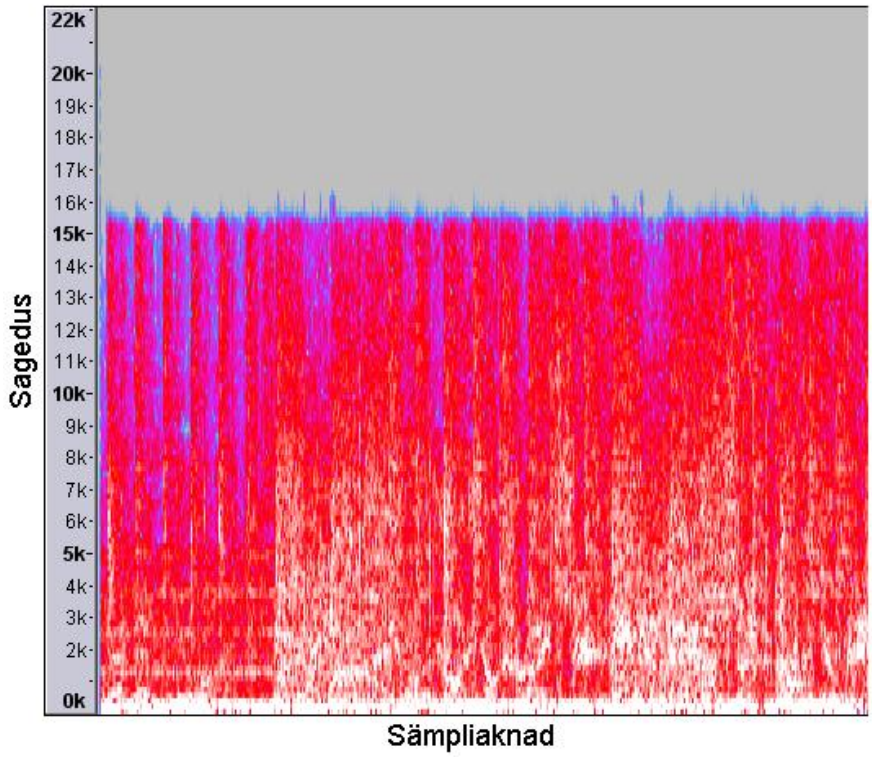
Digitaalsel kujul salvestatud heli võib vaadata kui diskreetset signaali aegruumis. Diskreetne tähendab siin seda, et signaali moodustavad järjestiku paiknevad üksikväärtused, milleks antud juhul on sämplid. Aegruum näitab lihtsalt, et signaal väljendab väärtuste muutumist üle aja. Sellisel kujul signaali on küllaltki raske analüüsida ning seetõttu on kasulik ta üle viia sagedusruumi. Sagedusruumis asuvast signaalist on võimalik välja lugeda signaali kompositsioon ehk kui suur tähtsus igal sagedusvahemikul antud signaali koostises on.

Diskreetse signaali teisendamiseks aegruumist sagedusruumi saab kasutada teisendust nimega diskreetne Fourier' teisendus, mis võtab sisendiks lõpliku arvu diskreetseid väärtuseid, teise nimega sämplimisakna, ning tagastab selle signaali osa kompositsiooni üle sagedusvahemike (joonis 9). Kui sisendiks on n reaalarvulist väärtust, siis väljundiks on n / 2 + 1 sagedusvahemikku, kus kõikide sagedusvahemike summa on võrdne poole signaali sämplimissagedusega. Kõik sagedusvahemikud on laiuse poolest võrdsed, väljaarvatud kaks äärmist, mis on kaks korda kitsamad kui ülejäänud. Seega oleneb vastuse täpsus sisendi suurusest, mis antud projekti raames on määratud 1024-ks, mis on küllaltki standardne väärtus. Väljund on siis jagatud 513-ks eri sagedusvahemikuks, mis on täiesti piisav tuvastamaks eraldi madal-, kesk- ja kõrgsageduserohked loo osad.



Joonis 9. Diskreetse Fourier' teisenduse sisend (sinine) ja väljund (lilla)

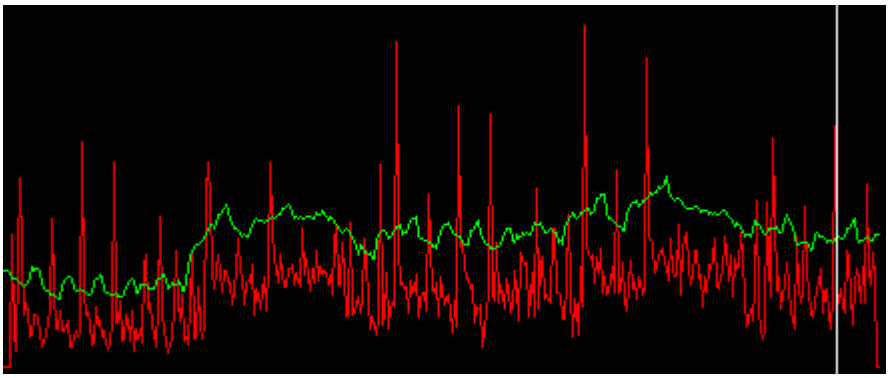
Teisendades signaali sämpliakna kaupa täismahus sagedusruumi saab koostada selle signaali spektraalesituse (joonis 10), mis sisaldab infot igast sämplimisaknast saadud sagedusvahemike energiate tasemete kohta. Sellisel kujul infol põhinebki antud projekti rütmituvastuse algoritm.



Joonis 10. Signaali spektraalesitus. Värv näitab energiat, mis kasvab suunas sinine-punane-valge. Iga piksel vasakult paremale tähistab ühte sämplimisakent

3.2.2 Rütmi tuvastamine

Liikudes joonisel vasakult paremale on näha, kuidas iga sagedusvahemiku energia aja kulgedes väheneb või suureneb. Kuna rütmituvastamisel on tähtis vaid energia muutus, täpsemalt positiivne energia muutus, siis on kasulik leida iga sämpliakna kohta tema sagedustevahemike muut võrreldes talle eelnenud sämpliaknaga. Vähenenud energiat tuleb löökide tuvastamisel ignoreerida, sest muidu võib näiteks negatiivne energia muut kõrgsagedustes neutraliseerida positiivse energia muudu madalsagedustes, mille tulemusena jääks löök tuvastamata, seetõttu peab leitud muutudest võtma absoluutväärtuse. Järgmiseks tuleb iga sämpliakna kohta arvutatud muudud kokku liita, et leida kogu positiivse energia kasv antud ajahetke kohta. Kui on vaja, siis saab kõikide muutude asemel kokku liita vaid määratud sagedusvahemike omad - nii on võimalik tuvastada lööke, mis asuvad näiteks ainult madalsagedustes, mis tavaliselt vastavad basstrummile.



Joonis 11. Signaali positiivse energia muut. Iga piksel vasakult paremale tähistab ühte sämplimisakent. Roheline joon tähistab konstandiga korrutatud jooksvat keskmist [6]

Järgmiseks sammuks on saadud signaali tippude tuvastamine. Selle jaoks on vaja arvutada iga sämpliakna jaoks jooksev keskmine ning ignoreerida kõiki väärtusi, mis sellest allapoole langevad (joonis 11). Tuvastavate löökide arvu vähendamiseks on võimalik jooksev keskmine mingi valitud konstandiga läbi korrutada, konstandi väärtus sõltub rakendusest, kuid antud projekti raames töötavad kõige paremini väärtused piirides 1,5 - 2,5.

Vältimaks olukordi, kus selgelt eristatavad löögid järgnevad üksteisele liiga kiiresti saab saadud tulemuse klasterdada. Valides mingi ajavahemiku, saab poolenisti kattuvate akendega kogu väljundi läbi käia, jättes iga akna jaoks alles vaid kõige suurema energiaga löögi. Nii saab garanteerida, et tuvastatud löökide intervall on vähemalt pool valitud ajavahemikust.

3.3 Visualiseerimine

Selle alapeatüki ülesandeks on anda ülevaade mängus kasutatavatest muusika visualiseerimise meetoditest.

3.3.1 Maastik

Antud projekti keskseks ideeks on mängumaailma ja muusika vastavus, mis peab väljenduma mängu igas elemendis. Kuna kõige tähtsamaks objektiks mängumehaanika vaatenurgast on musta värvi maastik (joonis 2), siis langeb väga suur osa sellest ülesandest just maastiku sobival viisil genereerimisele. Võimendamaks hetkel muusikas valitsevat emotsiooni on heaks taktikaks manipuleerida peategelase liikumiskiirust ning maastiku kaldenurka. On loogiline mõelda, et valju muusika taustal peaks liikumine olema kiirem ja allamäge ning vaikse muusika korral vastupidi. Kuna rütmituvastuse etapis on väikeste

muudatustega võimalik välja arvutada kogu muusika energia iga sämpliakna kohta, siis on maastiku nurga välja arvutamine suhteliselt lihte ülesanne. Üheks võimalikuks, kuid mitte ainsaks teeks on helitugevuse kujutamine mingisse määratud vahemikku, näiteks -60 kuni 60, ning interpreteerida iga sellist väärtust kui maastiku nurka kraadides x-telje suhtes.

Saadud maastik on väga krobeline, mille tulemusena on peategelase liikumine väga ebaühtlane. Loogiliseks järgmiseks sammuks oleks tulemuse ümardamine, kuid selle tulemusena kaotab maastik kõik huvitavad elemendid - näiteks künkad, mis varem vastasid trummilöökidele. Siin on abiks eelnevalt rütmituvastuses arvutatud löögid - nende abil on võimalik künkad ümardatud maastikule uuesti lisada, kuid palju kontrollitumal viisil, mis ei häiri peategelase sujuvat kulgemist.

Sellisel kujul ehitatud maastik on muusikaga väga hästi seotud, kuid kuna see on arvutuslikult genereeritud näeb ta visuaalselt suhteliselt igav välja. Olukorra parandamiseks saab aga väga lihtsalt maastiku kunstniku poolt joonistatud detailidega täiendada, lisades sinna rohtu, puid ja muid detaile (joonis 12)

Joonis 12. Maastiku detailid

3.3.2 Taust

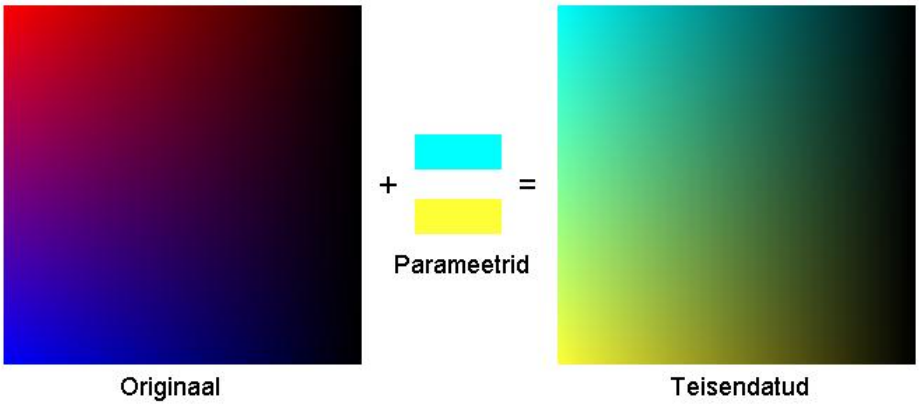
Sekundaarse osa mängu visuaalsest poolest moodustab taust. Olgugi, et mänguline tähtsus tal peaaegu puudub, on tal väga oluline osa muusika emotsiooni edasiandmisel. Antud projektis on selle saavutamiseks kasutusel kaks meetodit, kuid nende mõistmiseks on enne vaja aru saada, millest taust koosneb ja kuidas teda ekraanil esitatakse.

Kõige tagumiseks elemendiks mängus on lihtne kunstniku poolt joonistatud pilt, mida on võimalik ilma vahesid nägemata üksteise kõrval lõputult korrata. Pildi teema muutub ülevalt alla liikudes kosmosest taevaks ning seejärel taevast mereks, parasjagu

aktiivne taustapildi osa sõltub peategelase asukohast y teljel. Sinna peale joonistatakse omakorda olenevalt aktiivsest teemast eraldi individuaalsed objektid - puud, sambad, lennukid, planeedid, pilved jne - mille järjekord ja tihedus on arvutuslikult genereeritud. Individuaalsete objektide kasutamine tähendab, et igale objektile on võimalik määrata erinev liikumiskiirus ning igat objekti on võimalik teistest sõltumata suurendada, vähendada, pöörata ning peegeldada. Nii on võimalik imiteerida kolmemõõtmelist keskkonda, kus kaugemal olevad objektid on väiksemad ning liiguvad aeglasemini. Lisaks sellele võimaldab selline lahendus pakkuda mitmekesisemat keskkonda tagades iga loo jaoks erineva kogemuse. Samuti on see kasulik ka muusika ja mängu omavahelise vastavuse tagamiseks - näiteks on taustaobjektid võimalik panna muusika rütmi järgi pulseerima.

Teiseks tähtsaks komponendiks muusika emotsioonide esitamisel mängus on värv. Nagu eelnevalt mainitud peab mängu värv olema sõltuvuses muusika valjudusest, vali muusika peaks esile tooma soojad toonid ning vaikne muusika külmad toonid. Et aga suvalist pilti ei ole võimalik reaalajas etteaimatavalt ümber värvida, on kasutusele võetud spetsiifiline pildiformaat.

Kõik kunstniku poolt valmistatud objektid peavad kasutama väga piiratud värvipaletti - lubatud on ainult sinine, punane, must ning kõik nende interpoleerimisel saadud vahepealsed värvid. Andes parameetriks kaks suvalist värvi on kasutades pikslivarjundajat võimalik sellisel kujul pilt teisendada uueks pildiks, mis kasutab eksklusiivselt ainult nende kahe parameetriks antud värvi ja musta omavahel interpoleeritud vahevärve (joonis 13).



Joonis 13. Värvipaleti teisendamine kahe suvalise värvi abil

Lisaks eelnevale on visualiseerimisel kasutusel palju standardseid elemente nagu osakeste süsteemid, udustamis- ning kontrastifiltrid, kuid antud lõputöö raames need eraldi mainimist ei vääri.

Kokkuvõte

Antud bakalaureusetöö eesmärgiks oli arendada mäng Android platvormile, mis võimaldaks mängijal mängimiseks kasutada tema enda muusikakollektsiooni. Mängu žanri valimisel oli kõige olulisemaks parameetriks väike konkurents, mis võimaldab mängul potentsiaalsetele ostjatele rohkem silma paista.

Töös analüüsiti ning põhjendati erinevaid antud projektiga seotud tehnoloogilisi valikuid ning kirjeldati suuremaid töö käigus tekkinud probleeme ning nende lahendusi. Töö piiratud mahu tõttu oli tähelepanu keskpunktiks valitud ainult Android platvormi ning muusikamängu spetsiifilised teemad.

Valminud mäng sisaldab kõiki töö käigus kirjeldatud elemente, kuid kommerts-kõlbliku tooteni jõudmiseks on vaja mängu veel viimistleda. Enne toote turule laskmist on plaanis mängust teha ka iOS versioon, mis on valitud tehnoloogiate ja disainiotsuste tõttu saavutatav suhteliselt väikese vaevaga.

Game development For Android Platform

Bachelor's thesis

Ats Vendik

The purpose of this thesis was to document the development process of a game on the *Android* platform. The genre was deliberately chosen to service the relatively undiscovered niche of music games and to avoid competing on the overly saturated mainstream market.

The thesis focuses on explaining the numerous technological choices made regarding the development platform, environment and libraries as well as describing multiple encountered problems and their solutions. Due to the limited size of the thesis most of the attention is concentrated on Android platform and music game specific issues, like thread management and onset detection.

The game that was developed in the duration of this project includes all the described elements, but requires additional polish to achieve the level of quality commonly found in successful commercial mobile games.

.

Viited

[1] Flosi, S. ComScore Reports February 2012 U.S. Mobile Subscriber Market Share. http://www.comscore.com/Press_Events/Press_Releases/2012/4/comScore_Reports_February_2012_U.S._Mobile_Subscriber_Market_Share (03.04 2012)

[2] Hardy, B. Dealing with Asset Compression in Android Apps http://ponystyle.com/blog/2010/03/26/dealing-with-asset-compression-in-android-apps/ (26.03 2012)

[3] *Ducrohet, X. Meier, R.* A Faster Emulator with Better Hardware Support. http://android-developers.blogspot.com/2012/04/faster-emulator-with-better-hardware.html (09.04 2012)

[4] Kioskea. Digital Audio. http://en.kioskea.net/contents/audio/son.php3 (16.10 2008)

[5] Corbet, C. MATLAB EQ: Background on Equalization. http://cnx.org/content/m15655/latest/ (19.12 2007)

[6] Zechner, M. Onset Detection Part 6: Onsets & Spectral Flux. http://www.badlogicgames.com/wordpress/?p=161 (22.02 2010)