

# **Musiikki-instrumentti –ohjelman suunnittelu ja toteutus iOS- käyttöjärjestelmään**

Jarkko Honkanen

Opinnäytetyö  
Joulukuu 2012  
Tietotekniikka  
Ohjelmistotekniikka

## TIIVISTELMÄ

Tampereen ammattikorkeakoulu  
Tietotekniikka  
Ohjelmistotekniikka

JARKKO HONKANEN:

Musiikki-instrumentti -ohjelman suunnittelu ja toteutus iOS-käyttöjärjestelmään

Opinnäytetyö 33 sivua  
Joulukuu 2012

---

Tämän opinnäytetyön tarkoituksena oli toteuttaa musiikkiaiheinen sovellus Apple iPad 2 -tablet-laitteelle ja siinä samalla opettaa työn tekijälle kyseiselle laitteelle ohjelmointi.

Työssä tutustuttiin laitteen ominaisuuksiin ja siinä käytettävään iOS-käyttöjärjestelmään, ohjelmistokehitykseen ja itse sovelluksen toteutukseen. Ohjelmistokehitys-osassa käsiteltiin kehitysympäristöä, sovelluksen testausta simulaattorissa ja laitteessa, sekä ohjelmointikieltä.

iOS-käyttöjärjestelmälle sovellusten ohjelmointi tapahtuu Objective-C-kielillä. Objective-C on C-kielen laajennos, joka lisää siihen olio-ominaisuuksia. Tämä kieli eroaa ulkonäöltään paljon muista kielistä, mutta on helppo opittava C- ja C++-kieliä osaavalle.

Työssä toteutettu esimerkkisovellus oli musiikki-instrumentti, jonka ruudulla on nappeja, joita koskettamalla käyttäjä voi soittaa musiikkia. Soittonapit on järjestelty hieman yleisestä tavasta poikkeavalla tavalla. Tarkoituksena on soittamisen helpottaminen sellaiselle henkilölle, joka ei ole kokenut soittaja. Esimerkkisovelluksen aiheen takia työssä on tarpeellista käsitellä myös hieman musiikin teoriaa.

## **ABSTRACT**

Tampereen ammattikorkeakoulu  
Tampere University of Applied Sciences  
Computer Science  
Software Engineering

JARKKO HONKANEN:  
iOS Development  
Developing a Virtual Musical Instrument

Bachelor's thesis 33 pages  
December 2012

---

The purpose of this thesis was to develop a virtual musical instrument working on Apple iPad 2 tablet device and to teach author to program on that platform.

Subjects in this thesis was the introduction to device and operating system, software development to iOS in general, including development environment, testing in simulator and in a device and the Objective-C programming language. The last part of the thesis is about implementing the example application.

Applications for iOS are developed with Objective-C programming language. Objective-C is a superset to C language that adds objects to it. Objective-C is easy to learn for those who know C and C++ already.

An example application in this thesis is a musical instrument that has button on screen and user can touch them to play music. Buttons are arranged differently than usual, helping player to play good sounding music with only a little experience in playing. Because of the topic of this application it was necessary to talk a little about music theory.

---

Key words: iOS, Xcode, Objective-C, iPad, touch screen, music

## SISÄLLYS

1	Johdanto.....	7
2	Kohdeympäristö iOS ja Apple iPad 2 .....	8
2.1	iOS .....	8
2.2	iPad .....	10
3	Ohjelmistokehitys ja työkalut.....	11
3.1	Xcode .....	11
3.2	iOS-simulaattori.....	12
3.3	Testaus laitteessa.....	12
3.4	Objective-C –kieli.....	13
3.4.1	Metodien kutsumisen syntaksi .....	13
3.4.2	Osoittimet.....	14
3.4.3	Property-ominaisuudet .....	15
3.4.4	Muistinhallinta ja ARC .....	16
3.4.5	Periytyminen .....	16
3.5	Cocoa Touch -kehykset.....	16
4	Musiikin teoriaa.....	18
4.1	Sävelet ja sävellajit .....	18
4.2	Soinnut .....	19
4.3	MIDI-arvot.....	20
5	Ohjelman suunnittelu.....	21
5.1	Ohjelman kuvaus ja idea .....	21
5.2	Ominaisuudet .....	23
6	Toteutus .....	25
6.1	Ohjelman rakenne .....	25
6.1.1	NoteButtonView- ja ChordButtonView –luokat.....	25
6.1.2	PlayView-luokka.....	26
6.1.3	TuningManager-luokka.....	26
6.1.4	TouchInstrumentViewController-luokka.....	26
6.2	Äänten soittaminen .....	26
6.3	Kosketusten seuraaminen.....	27
6.4	Sävellajien ja sointujen määrittely ja käsittely.....	28
6.4.1	Sävellajitiedoston muoto .....	28
6.4.2	Sointutiedoston muoto .....	29
6.4.3	Sävelen nimen muuntaminen midi-arvoksi.....	29
6.4.4	Sävellajin vaihtaminen .....	30
6.4.5	Soinnun sävelten määrittäminen .....	30

7 Yhteenveto.....	32
LÄHTEET.....	33

**TERMIT**

iOS	Applen mobiililaitteille tekemä käyttöjärjestelmä
Xcode	Kehitysympäristö, jota käytetään iOS-järjestelmälle ohjelmointiin.
Objective-C	Olio-ohjelmointikieli, joka on C-kielen laajennos. Käytetään iOS:n ja OS X:n sovellusten ohjelmointiin.
iPad	Applen valmistama tablet-laite. Laitteesta on useampia versioita ja tässä työssä sillä tarkoitetaan iPad 2 –laitetta.
UIKit	iOS:n sovellusten käyttöliittymän toteuttamiseen käytetty sovelluskehys.
Cocoa Touch	Kokoelma iOS:n sovelluskehysistä. Tehty erityisesti kosketusnäyttöllisiä laitteita varten.

# 1 Johdanto

Tämän työn tarkoituksena on tutustua ohjelmistokehitykseen Applen iOS-käyttöjärjestelmälle ja sitä käyttäville kosketusnäytöllä varustetuille laitteille. Kyseistä käyttöjärjestelmää käyttäviä laitteita ovat iPhone, iPod Touch ja iPad. Samaa käyttöjärjestelmää käyttää myös televisioon kytkettävä AppleTV, mutta sitä ei kuitenkaan tässä työssä käsitellä.

Työssä käsitellään iOS-järjestelmään tarkoitettujen sovellusten kehitykseen tarvittavia työkaluja, Objective-C –ohjelmointikieltä ja itse ohjelman toteutusta. Työssä tutustutaan myös iOS-käyttöjärjestelmän ja sitä käyttävien laitteiden ominaisuuksiin ja niiden historiaan. Työssä selvitetään myös iOS-käyttöjärjestelmän ominaisuuksia ja niiden historiaa.

Esimerkkinä tässä työssä toteutetaan kosketusnäytöllä toimiva musiikki-instrumentti iPad 2 -tablet-laitteelle. Työn aiheen takia ja esimerkkiohjelman toiminnan ymmärtämisen kannalta on tarpeellista käsitellä hieman myös musiikin teoriaa.

Luvussa 2 esitellään kohdeympäristö eli iPad-tablet ja sen iOS-käyttöjärjestelmä. Luvussa 3 käsitellään sovellusten kehittämistä iOS-ympäristöön. Aiheina on kehitysympäristö Xcode, ohjelmointikielen tärkeimmät ominaisuudet ja erot C++:aan sekä muutama sana iOS:n käyttämästä Cocoa Touch -sovelluskehiksestä. Luvussa 4 on hieman musiikin teoriaa niiltä osin, mitä tämän työn esimerkkisovelluksen toiminnan ymmärtämiseksi tarvitaan. Luvussa 5 esitellään toteutetun ohjelman käyttöliittymä ja ominaisuudet. Luku 6 käsittelee ohjelman toteutusta ja siinä kerrotaan miten kaikki ohjelman luokat on toteutettu.

## 2 Kohdeympäristö iOS ja Apple iPad 2

Tässä luvussa esitellään lyhyesti käytettävä laite ja käyttöjärjestelmä.

### 2.1 iOS

iOS on Applen käyttöjärjestelmä sen omille mobiililaitteille. Se on suunniteltu kokonaan kosketusnäytöllä käytettäväksi ja yhden kosketuksen lisäksi se tukee montaa yhtäaikaista kosketusta ja erilaisia kosketuseleitä, jotka helpottavat käyttöä. Mobiililaitteita, jotka käyttävät iOS-käyttöjärjestelmää, ovat iPhone, iPod Touch ja iPad. Kuviossa 1 nähdään laitteiden ulkonäköä. (The Verge: iOS: A visual history)



Kuvio 1: iPod Touch, iPad ja iPhone (Apple: What is iOS)

iOS:n perusnäkökulma eli kotiruutu on pysynyt alusta asti samanlaisena. Se sisältää ainoastaan ohjelmakuvakkeita, joita koskettamalla avataan ohjelma. Kotiruutuja voi olla useita ja niiden välillä siirrytään pyyhkäisemällä ruutua sormella sivulle.



iOS:ssä on moniajo-ominaisuuksia, mutta ne on toteutettu hieman erilailla kuin tietokoneissa on totuttu. Useimmat ohjelmat menevät pysäytettyyn tilaan suljettaessa ja ne avautuvat samaan näkymään uudelleen käynnistettäessä ja tämä tapa toimiikin useimpien ohjelmien kanssa hyvin. Jotkin ohjelmat, kuten nettipuheluohjelmat pystyvät toimimaan taustalla ja osa ohjelmista käyttää push-viestejä palvelimelta asioiden ilmoittamiseen käyttäjälle. Useimmiten kehittäjän ei tarvitse ottaa huomioon ohjelman tehojen käyttöä taustalla olon aikana, ellei ohjelman nimenomaan tarvitse pystyä tekemään toimintoja myös silloin. (The Verge: iOS: A visual history)

iOS-käyttöjärjestelmä esiteltiin ensimmäisen kerran ensimmäisen iPhone mallin julkistuksen yhteydessä vuonna 2007 nimellä iPhone OS. Silloin siihen ei vielä pystynyt asentamaan kolmannen osapuolen ohjelmistoja. Kehittäjien oli mahdollista kuitenkin tehdä selaimella toimivia pikkuohjelmia, jotka muistuttivat toiminnaltaan oikeaa sovelusta. (The Verge: iOS: A visual history)

Vuonna 2008, versiossa 2.0 käyttöjärjestelmään julkaistiin kehitysympäristö eli SDK, joka mahdollisti myös ulkopuolisten kehittäjien tekemän omia ohjelmiaan. Ohjelmien jakelu on iOS:ssä rajoitettua ja ainoa virallinen ja tuettu jakelukanava on Applen ohjelmakauppa App Store. App Storea voi käyttää iOS-laitteista löytyvän ohjelman avulla tai tietokoneella iTunes-ohjelmalla. App Storessa kehittäjien on mahdollista myydä kehittämiään ohjelmia helposti tai jakaa niitä ilmaiseksi. Myytyjen ohjelmien tuloista Apple ottaa itselleen 30% ja loput 70% jää kehittäjälle. Ohjelmien sisällä voi olla maksullisia lisäosia tai lisäsisältöä, joiden tuotot myös kulkevat App Storen kautta. (The Verge: iOS: A visual history, Apple Developer: Distribute your App - iOS Developer Program)

Vuonna 2010 iPhone OS nimi vaihdettiin iOS:ksi. Nimenvaihdos tehtiin, koska iPhone ei ole ainoa samaa käyttöjärjestelmää käyttävä laite, vaan iOS löytyy myös iPadista ja iPod Touchista. iOS löytyy myös AppleTV:stä televisiokäyttöön soveltuvalla käyttöliittymällä. (The Verge: iOS: A visual history)

Kirjoittamishetkellä iOS on versiossa 6. (Apple: iOS 6)

## 2.2 iPad

iPad 2 on 9,7 tuuman kosketusnäytöllä varustettu tablet-laite. iPad 2 on julkaistu maaliskuussa 2011. Näytön tarkkuus on 1024x768 pikseliä. Akkukesto on noin 10 tuntia. Langattomina yhteyksinä siinä on versiosta riippuen pelkkä WiFi tai sekä WiFi, että 3G-datayhteys ja Bluetooth-yhteys molemmissa versioissa. Tallennustilaa laitteessa voi olla 16, 32 tai 64 Gt. 3G-mallissa on myös GPS-paikannin.

iPadia ohjataan sormilla kosketusnäytöllä. Hiirtä ei ole vaan kaikki asiat tehdään suoraan näyttöä koskettamalla. Kirjoitettaessa käytetään ruudulle tarvittaessa ilmestyvää virtuaalista näppäimistöä, mutta iPadin kanssa voi käyttää myös tavallisia Bluetooth-näppäimistöjä. Ohjaukseen voidaan käyttää esimerkiksi joissain peleissä myös laitteessa olevaa asentotunnistinta ja gyroskooppia, jolloin vaikkapa autoa tai lentokonetta voi ohjata kääntelemällä laitetta eri suuntiin. (Apple – iPad 2 – View the technical specifications for iPad 2)

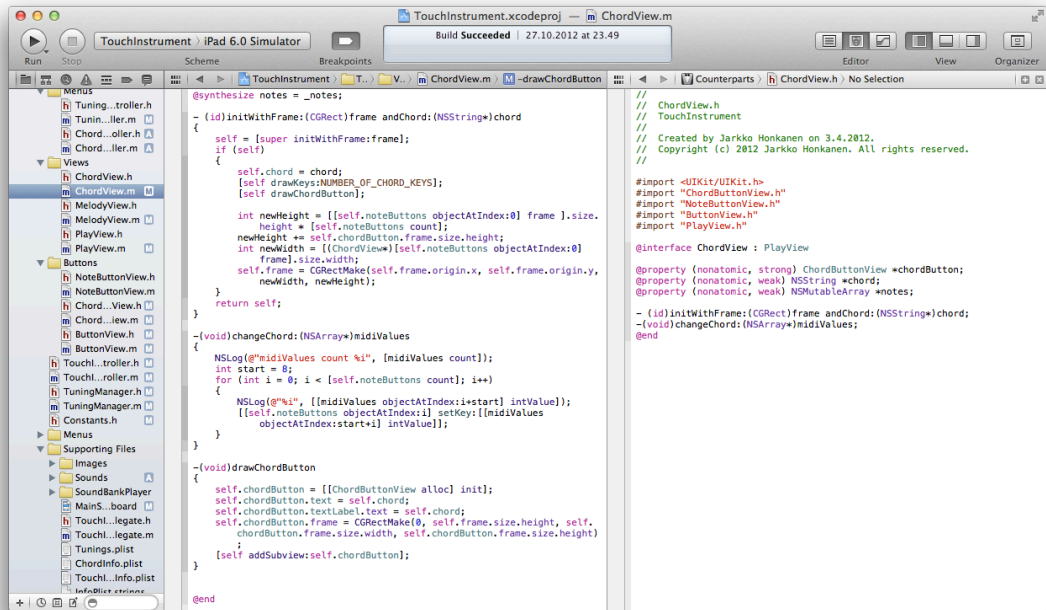
### 3 Ohjelmistokehitys ja työkalut

Tässä kappaleessa esitellään lyhyesti iOS –alustalle ohjelmoimiseen tarvittavia työkaluja sekä kerrotaan hieman niiden käytöstä. Kappaleessa esitellään myös käytettävä ohjelmointikieli, Objective-C.

#### 3.1 Xcode

Kehitysympäristönä iOS:lle on Xcode. Se sisältää koodieditorin, debuggerin, käyttöliittymäsuunnittelutyökalun ja kaiken tarvittavan iOS:lle ohjelmoimiseen. Nykyinen versio koostuu yhdestä isosta ikkunasta, jonka sisällä kaikki tapahtuu, kun vanhemmissa versioissa useimmille osioille oli omat ikkunansa. Mielestäni nykyinen tapa on parempi ja selkeämpi. Xcodessa on myös versionhallinta sisäänrakennettuna ja se toimii suoraan Xcoden käyttöliittymästä. Xcoden versionhallinta tukee Subversion- ja Git-protokollia. Xcoden ominaisuuksiin voi tutustua osoitteessa <https://developer.apple.com/xcode/>.

Xcoden päänäkymässä vasemmalla on kansiohierarkia, jossa projektin tiedostot sijaitsevat. Kansioden nimillä ei ole ohjelmoinnin kannalta merkitystä vaan ne ovat olemassa vain ohjelmoijaa varten työtä selkeyttämässä. Kansioissa olevat tiedostot löytyvät tiedoston nimen perusteella riippumatta siitä, missä ne sijaitsevat. Kansiohierarkian oikealla puolella on tekstieditori, jossa ohjelmointi suoritetaan. Sen oikealle puolelle saa vielä auki Assistant Editorin, jonka voi asettaa näyttämään automaattisesti aina valitun tiedoston ”vastakappaleen”, eli valittaessa .m-tiedosto, Assistant editoriin aukeaa .h-tiedosto. Jos käyttäjä valitsee käyttöliittymätiedoston, tekstieditorin tilalle aukeaa käyttöliittymäsuunnittelutyökalu. Kuviossa 2 Xcoden käyttöliittymä Assistant editor avoinna.



Kuvio 2: Xcoden käyttöliittymä

### 3.2 iOS-simulaattori

Osana Xcode-kehitysympäristöä on simulaattori, jolla voidaan testata tehtyjä sovelluksia tietokoneella ilman itse laitteita. Kun ohjelmaa halutaan testata simulaattorissa, simulaattori täytyy valita Run-napin vierestä olevasta pudotusvalikosta. Run-nappia klikkaamalla Xcode kääntää ohjelman, käynnistää simulaattorin ja avaa ohjelman simulaattoriin. Simulaattorissa voi kääntää laitetta eri asentoihin 90 asteen välein ja valita mitä laitetta simuloidaan. iPad 2:n näytön tarkkuus on 1024x768 pikseliä. Uudemmissa iPad-versioissa näytön tarkkuus on kaksinkertainen eli 2048x1536 pikseliä. Simulaattorista voidaan valita, kumpaa näyttöä simuloidaan.

Kosketusnäyttölaitteen simuloiminen hiirellä käytettävällä tietokoneella asettaa rajoituksia ja monia ominaisuuksia testattaessa se ei riitä vaan ohjelmaa on pakko testata oikealla laitteella. Esimerkiksi monen yhtäaikaisen kosketuksen testaaminen, sekä asentotunnistimen ja gyroskoopin testaus ei luonnollisesti tietokoneella onnistu.

### 3.3 Testaus laitteessa

Oikealla laitteella testaaminen tapahtuu lähes samalla tavalla kuin simulaattorissa testaaminen. Xcoden pudotusvalikosta valitaan laitteen nimi ja painetaan Run -nappulaa.

Laitteen täytyy olla kytketty USB-johdolla tietokoneeseen kiinni. Ennen laitteessa testausta vaaditaan kuitenkin valmisteluja.

Laitteessa testaamiseen vaaditaan aina Applen kehittäjäohjelmaan liittyminen, mikä on maksullista. Ohjelman jäsenyys ostetaan vuodeksi kerrallaan ja sen hinta on yksittäiselle kehittäjälle 99\$. Yrityksille on tarjolla monelle käyttäjälle tarkoitettu ohjelma, joten jokaiselle työntekijälle ei tarvitse erikseen sitä hankkia. Opiskelijoille on tarjolla oma ilmainen ohjelma, mutta se ei mahdollista ohjelmien julkaisua, eikä myyntiä. Liittymisen jälkeen kehittäjä saa pääsyn Applen Member Centeriin, jonka Provisioning Portalissa hallitaan testaamiseen ja julkaisemiseen tarvittavia sertifikaatteja ja laitteita. Kaikki laitteet, joilla ohjelmia aiotaan testata, täytyy rekisteröidä Provisioning Portalissa ja kehittäjän täytyy asentaa kehittäjäsertifikaatti tietokoneelleen. Testattavalle ohjelmalle täytyy luoda App ID. (Apple Developer: Choosing an iOS Developer Program)

Kaikki edellä mainitut toimenpiteet tuovat hieman lisätyötä ja ovat välillä aiheuttaneet pieniä ongelmia tässä ja muissa tekemissäni projekteissa. Ongelmia ei enää alkuasetusten kuntoon saamisen jälkeen ole ollut ja testaaminen ei enää sen jälkeen vaadi lisätoimenpiteitä. Olen hieman oikaissut käyttämällä kaikissa projekteissa samaa App ID:tä, koska se toimii normaalisti, jos kyseessä on vain omassa laitteessa testaaminen, eikä julkaisu.

### **3.4 Objective-C –kieli**

iOS:ssä, kuten myös Applen tietokoneiden OS X –käyttöjärjestelmässä, pääasiallisena ohjelmointikielenä on Objective-C. Objective-C on C-kielen laajennos, joka tuo siihen olio-ominaisuuksia. Se on täysin yhteensopiva C-kielen kanssa ja Objective-C –kääntäjä pystyykin kääntämään C-kieltä normaalisti. Tässä kappaleessa esitellään Objective-C:n olennaisimpia ominaisuuksia ja verrataan hieman eroavaisuuksia C++:aan. Tämän luvun koodiesimerkit ovat suoraan tai vähän muokattuina tämän työn esimerkkiohjelmasta. (Apple Developer: The Objective-C Programming Language: Introduction)

#### **3.4.1 Metodien kutsumisen syntaksi**

Ensivilkaisulla Objective-C –koodi näyttää erilaiselta kuin C++. Suurin syy tähän on ero olioiden metodikutsuissa. Olioiden metodikutsut ovat aina hakasulkeiden sisässä.

```
[noteButtons insertObject:noteButton atIndex:3];
```

`noteButtons` on tässä esimerkissä taulukko-olio. Metodin nimi on `insertObject:atIndex:`. Jos metodin kutsussa on useampi kuin yksi parametri, niille kaikille täytyy antaa nimi. Tämä voi usein aiheuttaa hyvinkin pitkiä koodirivejä, mutta hyötynä tästä on se, että kyseisen rivin toiminnan varmasti ymmärtää helposti sellainenkin lukija, joka ei ole kyseistä oliota tai metodia ennen käyttänyt. C++:lla vastaava rivi voisi olla esimerkiksi:

```
noteButtons.insertObject(noteButton, 3);
```

### 3.4.2 Osoittimet

Objective-C:ssä olioiden kanssa käytetään aina osoittimia, eikä niitä ole mahdollista luoda ilman osoitinta. Tavallisten C-tyylisten muuttujien kanssa osoittimia ei tarvita. Metodi myös palauttaa aina osoittimen, jos palautettava tietotyyppi on olio. Tyypillinen tapa luoda olio on:

```
NoteButtonView *noteButton = [[NoteButtonView alloc] init];
```

Sisempien hakasulkeiden sisällä varataan muistia oliolle. Sen jälkeen syntyneestä oliosta ajetaan alustusmetodi, `init`, joka palauttaa osoittimen itseensä. Tämä osoitin tallennetaan `noteButton` –osoittimeen.

Koska olioiden kanssa käytetään aina osoittimia, sijoitusoperaattorin käyttäminen ei kopioi oliota vaan ainoastaan osoittimen. Jos olio halutaan kopioida, sitä varten valmiista luokista löytyy `copy`-metodi, joka luo oliosta kopion ja palauttaa osoittimen siihen.

Objective-C:ssä on olemassa tietotyyppi `id`, joka on osoitin minkä tahansa tyyppiseen olioön. Tämän tietotyypin kanssa ohjelmoijan kannattaa olla varovainen ja tarkistaa ennen metodien kutsumista, että osoitin todella osoittaa oletetun tyyppiseen olioön. Jos `id` osoittaa eri tyyppiseen olioön kuin on oletettu, se ei ehkä vastaakaan metodikutsuun, mikä voi johtaa ohjelman kaatumiseen. Tätä voidaan myös käyttää hyväksi, koska metodeita voidaan kutsua ilman varmuutta, että osoitin edes osoittaa minnekään. Jos

metodia kutsutaan osoittimesta, joka ei osoita mihinkään olioon, mitään ei tapahdu. Näin voidaan jättää tiettyjä tarkistuksia pois jos tällainen toimintatapa sopii kyseiseen ohjelman kohtaan. Oliolta voidaan myös kysyä, minkä luokan se toteuttaa.

### 3.4.3 Property-ominaisuudet

Propertyt ovat Objective-C:n ominaisuus muuttujien helpompaan käsittelyyn. Property voi olla olio tai tavallinen tietotyyppi. Propertyn luomiseen tarvitaan seuraavanlaiset kaksi riviä:

```
@property (nonatomic) NSArray *chordInfo;

@synthesize chordInfo
```

Ensimmäinen rivi kirjoitetaan header-tiedostoon (.h) ja synthesize-komento toteutustiedostoon (.m). Suluissa oleva nonatomic tarkoittaa, että propertyä ei ole turvallista käyttää säikeiden kanssa vaan se voi johtaa kaatumiseen, jos useampi säie yrittää kirjoittaa siihen samaan aikaan. Tätä kuitenkin yleensä käytetään jos säikeitä ei tarvita, koska se on nopeampi tapa. Synthesize –komennolla kääntäjä luo propertylle automaattisesti setter ja getter –metodit. Näiden metodien käyttö ei normaalisti eroa normaalien muuttujan käsittelystä, mutta jos näihin metodeihin tekee omia lisäyksiä, ne tapahtuvat aina kun muuttujan arvoa muutetaan tai luetaan. Kun muuttujasta tehdään property, nämä metodit suoritetaan myös silloin kun muuttujaan viitataan normaalilla pistenotaatiolla.

```
self.melodyView.instrumentNotes =
self.tuningManager.wholeInstrumentNotes;
```

wholeInstrumentNotes on tässä tapauksessa taulukko, jonka osoitin vain normaalisti kopioitaisiin. Jos instrumentNotes on kuitenkin esitelty propertynä, sille on olemassa oma setter-metodi, joka arvon asetuksen lisäksi ajaa propertyn sisältävässä luokassa olevan metodin. Näin luokka reagoi automaattisesti muuttujan arvon vaihtumiseen. Toiminta muistuttaa C++:n ominaisuutta, jossa operaattoreiden toimintaa voidaan olioissa muuttaa. Objective-C:ssä vastaava ei ole kuitenkaan mahdollista muiden operaattoreiden kuin sijoituksen kanssa.

Propertyt ovat julkisia, jos ne esitellään otsikkotiedostossa ja yksityisiä jos ne esitellään toteutustiedostossa. Objective-C:ssä voi olla myös tavallisia C-tyylisiä muuttujia.

#### 3.4.4 Muistinhallinta ja ARC

Objective-C:n muistinhallinta toimii siten, että oliolla on retain-count, jonka mennessä nolnaan sen muisti vapautetaan. Ohjelmoija voi hallita tätä arvoa metodeilla retain, joka kasvattaa arvoa yhdellä ja release, joka vähentää määrää yhdellä. Osa olioista voi olla ns. autorelease –olioita, jotka automaattisesti käyttävät release komentoa jossain vaiheessa suoritusta, kun oliota ei enää tarvita.

ARC eli Automatic Reference Counting on Objective-C:n automaattinen muistinhallinta, joka muistuttaa Javan roskien keruuta. Toisin kuin Javan roskien keruu, ARC ei toimi aktiivisesti ohjelman ajon aikana vaan se lisää kääntämisvaiheessa koodin sekaan tarvittavat retain ja release –komennot, jolloin ohjelmoijan ei enää tarvitse huolehtia muistin hallinnasta. Tämän työn esimerkkiohjelmassa on hyödynnetty ARC:tä, joten olioiden vapauttamista ei ole tarvinnut itse miettiä. ARC ei ole pakollinen ominaisuus vaan sen käyttö on ohjelmoijan päätettävissä. Jos ARC on päällä, ohjelmoija ei pysty itse käyttämään retain- ja release –metodeita.

#### 3.4.5 Periytyminen

Objective-C:ssä luokka voi periytyä vain yhdestä luokasta. Se ei siis tue C++:sta tuttua moniperiytymistä. Kun valmiiden luokkien periytymistä tarkastellaan riittävän pitkälle, huomataan, että lähes kaikkien luokkien kantaluokkana on NSObject. Jos ohjelmoijan oman luokan ei ole syytä periytyä mistään muusta valmiista luokasta, sen täytyy periytyä NSObject-luokasta. On kyllä mahdollista tehdä luokka, joka ei periydy NSObject-luokasta, mutta se ei ole järkevää, koska NSObject sisältää tärkeitä ominaisuuksia muistin hallintaan ja luokkien kopioimiseen ja niiden toteuttaminen itse olisi työlästä ja virheeltistä.

### 3.5 Cocoa Touch -kehykset

Cocoa Touch on iOS:ssä kokoelma sovelluskehyskiä. Se sisältää sovelluskehyskiä esimerkiksi äänen ja videon toistamiseen ja käsittelyyn, karttojen, verkkoselaimen, verk-



koyhteyksien ja laitteiden, kuten GPS-paikantimen ja gyroskoopin käyttöön. Siinä on myös työkalut ohjelman sisäisen tietokannan luomiseen ja käyttämiseen, mikä on toteutettu SQLiteillä.

UIKit on iOS:ssä käytettävä sovelluskehys käyttöliittymän ohjelmoimiseen. Se on suunniteltu nimenomaan kosketusnäytöisille laitteille. Kaikkien siihen kuuluvien luokkien nimet alkavat UI-etuliitteellä. Kaikki tämän työn esimerkkiohjelman käyttöliittymän näkyvät elementit on toteutettu UIKitin luokkien avulla. Näkyvien käyttöliittymäelementtien lisäksi UIKit sisältää luokkia piirtämiseen sekä kosketusten ja eleiden käyttämiseen.

## 4 Musiikin teoriaa

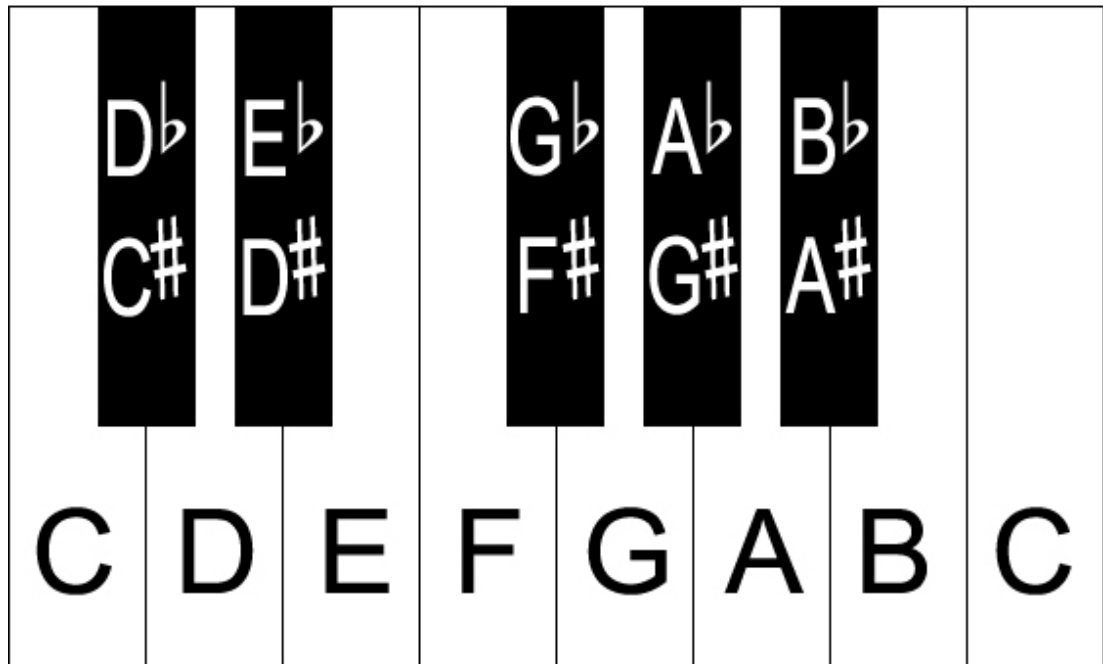
Tässä työssä oleellisessa osassa on musiikkiin liittyvät asiat. Koska siihen liittyviä termejä käsitellään tässä työssä, on tarpeen käsitellä muutamia musiikin teorian aiheita. Tämän kappaleen asioiden ei ole tarkoitus olla musiikin teorian kannalta täysin oikeaoppisia, vaan tarkoituksena on selvittää asioita siten, että musiikin teoriaa tuntematon lukija pystyisi ymmärtämään riittävästi tämän työn esimerkkiohjelman toimintaa. Teoriaa on käsitelty käyttäen pianoa esimerkkinä, koska se on monille ihmisille tuttu soitin ja sen koskettimisto on selkeä ja sopiva asioiden havainnollistamiseen.

### 4.1 Sävelet ja sävellajit

Yleisesti ottaen musiikkia käsitellään seitsemän sävelen avulla. Sävelten nimet ovat C, D, E, F, G, A, B. Ensin mainitun sävelen ääni on matalin ja viimeksi mainitun ääni korkein. Sävelten luetteleminen yleensä aloitetaan C-sävelestä, eikä aakkosten alusta. Nämä seitsemän säveltä muodostavat yhden oktaavin. Kun tämä sävelsarja on käyty läpi, siirrytään seuraavaan oktaaviin, jossa sävelten nimet ovat samat. Usein oktaavia merkitään numerolla sävelen perässä. Sävelet C2 ja C3 ovat siis seitsemän sävelen päässä toisistaan. Oktaavista toiseen siirtyminen tarkoittaa sitä, että saman sävelmän voi soittaa mistä tahansa oktaavista ja se kuulostaa oikealta, kunhan se aloitetaan samasta sävelestä (sama kirjain). Sävelmän korkeus on tällöin eri.

Edellä mainitut seitsemän säveltä vastaavat pianon valkoisia koskettimia. Pianossa on myös mustia koskettimia valkoisten välissä. Näitä kutsutaan korotetuiksi tai alennetuiksi säveliksi. Pianon koskettimet ovat puolen sävelaskeleen päässä toisistaan. Jokainen perussävel voidaan korottaa tai alentaa. Korotusta merkitään kirjoittamalla sävelen perään # ja alennusta kirjoittamalla sävelen perään b. Kun sävelestä C mennään puoli sävelaskelta ylöspäin, saadaan sävel C#. Kun sävelestä D mennään puoli sävelaskelta alaspäin, saadaan sävel Db. Kuten kuvioista 3 voi päätellä, nämä sävelet ovat itse asiassa sama asia. Tilanteesta riippuu kumpana se merkitään. Sävelten B ja E jälkeen ei tule mustaa kosketinta. Ne voidaan silti kuitenkin tarvittaessa merkitä korotettuna, jolloin seuraava kosketin on valkoinen. E# tarkoittaakin siis samaa kuin F. Vaikka pianon valkoisilla koskettimilla voikin soittaa hyvin monenlaista musiikkia, myös mustia tarvitaan monissa kappaleissa saamaan aikaan haluttu tunnelma. Aiemmin mainittiin oktaavissa

olevan seitsemän säveltä. Tämän työn kannalta tärkeämpi tieto on se, että oktaavissa on 12 puolisisävelaskelta. Tätä tietoa käytetään esimerkkiohjelman laskennassa.



Kuvio 3: Pianon koskettimisto sävelten nimet merkittyinä (John Classick: Names of Notes and Intervals)

Valintaa siitä, mitkä sävelet soitetaan sellaisenaan tai korotettuna tai alennettuna kutsutaan sävellajiksi. Monille tuttuja sävellajeja ovat duuri ja molli. Duuria usein sanotaan iloisen kuuloiseksi ja mollia surullisen kuuloiseksi. Sävellajilla on aina jokin juurisävel, minkä suhteen sen korotukset ja alennukset on tehty. Jos sama kappale soitetaan esimerkiksi D-molli –sävellajissa tai F-molli sävellajilla, ne kuulostavat samalta, mutta ovat eri korkeudella. Juurisävelen erosta johtuen näiden sävellajien korotukset ja alennukset ovat eri vaikka ovatkin saman tyyppisiä sävellajeja.

## 4.2 Soinnut

Soinnut ovat kokoelma säveliä tietyn kaavan mukaan. Yleisimmin käytettyihin perussointuihin kuuluu kolme säveltä. Erikoisempiin sointuihin voi kuulua myös useampia säveliä. Soinnun sävelet määritellään yhden oktaavin alueella. Soittajan päätettäväksi jää, kuinka monen oktaavin alueelta hän sävelet soittaa. Esimerkiksi D-molli -sointuun kuuluu sävelet D, F ja A. Soittaja voi soittaa pianolla haluamastaan kohtaa nämä kolme lähes vierekkäin olevaa säveltä tai hän voi halutessaan soittaa jokaisen soittimesta löytyvän sointuun kuuluvan sävelen. Sointuja soitetaan usein soittamalla soinnun sävelet

samaan aikaan, mutta niitä voi soittaa myös yksitellen soittajan haluamassa järjestyksessä.

Kuten sävellajeilla, myös soinnuilla on juurisävel. Tässä työssä toteutettu esimerkkiohjelma laskee sointujen sävelet soinnun tyypin ja juurisävelen perusteella. Soinnun sävelet voidaan määritellä ilmoittamalla, kuinka monta puoliaskelta pitää siirtyä juurisävelestä eteenpäin, niin että juurisävel on numero 1. Silloin saman tyyppinen sointu voidaan muodostaa mille tahansa annetulle juurisävelelle helposti. Tämän työn esimerkkiohjelmassa soinnun tyypit on määritelty ilmoittamalla sarja lukuja, joiden perusteella lasketaan sointuun kuuluvat sävelet, kun juurisävel on ilmoitettu.

### **4.3 MIDI-arvot**

MIDI on standardi, jota käytetään sähköisissä musiikkilaitteissa. Jokaisella puolisävelellä on oma MIDI-arvonsa, jonka ilmoittamalla, mikä tahansa MIDI-standardin mukainen laite tai ohjelma osaa soittaa oikean sävelkorkeuden. Tämän työn esimerkkiohjelmassa käytetään näitä arvoja. Käyttämällä lukuja sävelten ilmoittamiseen, on myös sointujen sävelten laskennallinen määrittäminen helppoa. Arvot menevät järjestyksessä sävelen korkeuden mukaan siten, että C0-sävelen arvo on 12. C#0 on 13, D0 14 jne.

## 5 Ohjelman suunnittelu

Tässä luvussa esitellään tätä työtä varten tekemäni esimerkkiohjelma, jota toteuttaessani opettelini iOS-ohjelmoinnin.

### 5.1 Ohjelman kuvaus ja idea

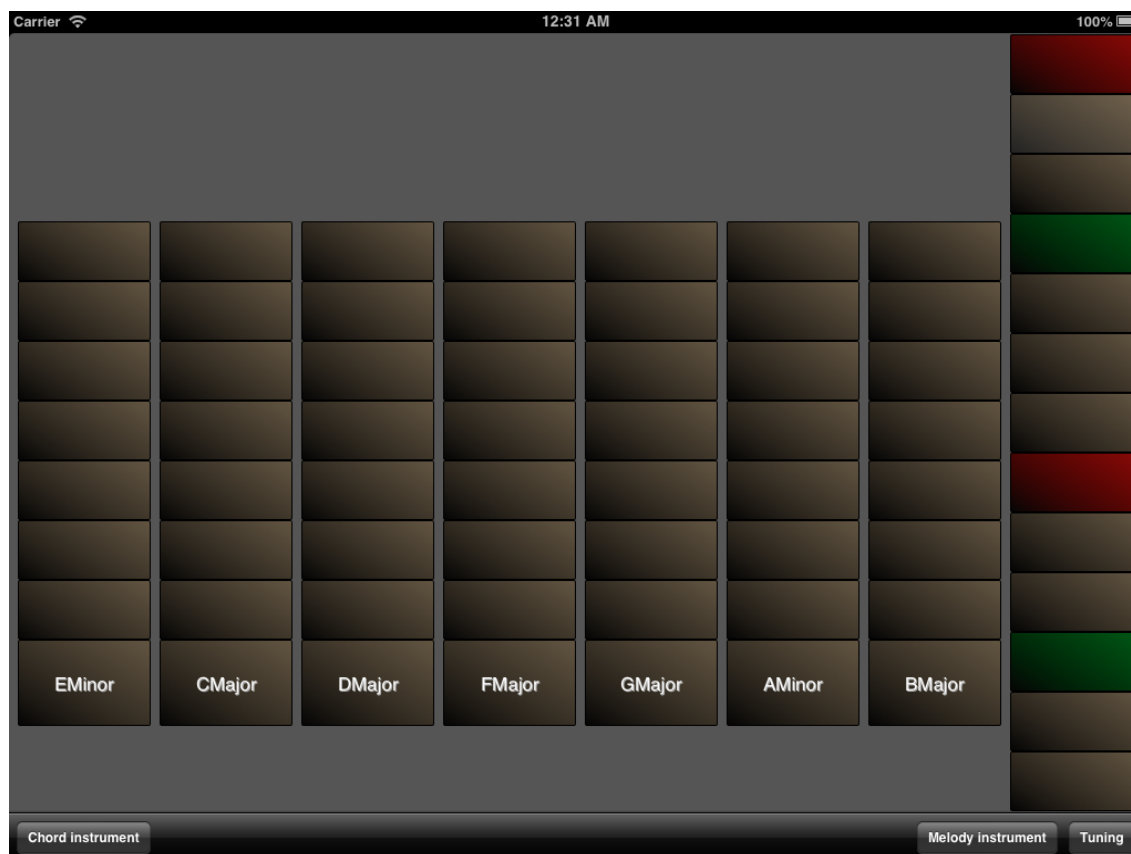
Toteuttamani ohjelma on musiikki-instrumentti, jota soitetaan koskettamalla näytölle piirrettyjä nappuloita. Ohjelman ulkoasuun ja käyttöliittymän kauneuteen ei ole panostettu, koska se ei ole tämän työn kannalta oleellista.

Valitsin tämän tyyppisen ohjelman, koska itseäni kiinnostaa musiikki ja ajattelin, että tämän tyyppinen ohjelma olisi kiinnostavaa toteuttaa. Samalla tässä työssä ei ole kyse pelkästä puhtaasta ohjelmoinnista vaan lisäksi on pitänyt miettiä, miten toteutukseen tarvittavat musiikilliset asiat saa yhdistettyä ohjelmointiin.

Olen tutkinut musiikin tuottamiseen soveltuvien ohjelmien tarjontaa App Storessa ja niiden valikoima on laaja. Ohjelmia löytyy hyvin yksinkertaisista ilmaisista ohjelmista suuriin ja monipuolisiin ohjelmiin, jotka voivat maksaa paljonkin. Kosketusnäyttö tarjoaa mahdollisuuksia monen tyyppisen käyttöliittymän toteuttamiseen, mikä ei ole perinteisillä soittimilla mahdollista. Kuitenkin pianon koskettimistolla varustetut soittimet näyttävät olevan todella yleisiä App Storessa. Onhan se monille tuttu soitin ja monet osaavat sitä soittaakin. Itse halusin toteuttaa hieman erilaisella käyttöliittymällä varustetun soittimen.

Kosketusnäyttö asettaa rajoitteita musiikin soittamiselle. Tuntuma on aina oikeaan soittimeen verrattuna huonompi, koska soittimen nappeja tai muita osia ei voi tuntea vaan pitää luottaa kokonaan näköhavaintoon. Toiseksi soiton voimakkuuden hallinta ei toimi normaalisti kosketusnäytöllä. Voimakkuuteen reagoinnin toteutus ei ole kuitenkaan mahdotonta. Se on tehty Applen GarageBand-ohjelmassa kiihtyvyysanturia hyväksi käyttäen, mutta testaukseni perusteella sen toiminta ei ole niin hyvä kuin toivoisi. (Apple Special Event March 2011, 49:50)

Kuviossa 4 näkyy ohjelman käyttöliittymä. Sen oikeassa reunassa on pystyssä sarja nappuloita, joita voi käyttää normaaliin melodian soittamiseen. Syy pystyasennolle, pianotyyllisen vaaka-asennon sijaan oli se, että minusta pystyasennossa soittaminen on helpompaa käden asennon takia. Nappuloiden äänet vastaavat oletusasetuksella pianon valkoisia koskettimia, mutta sävellaji on käyttäjän vaihdettavissa.



Kuvio 4: Toteutetun ohjelman käyttöliittymä

Suurimman osan ruudusta täyttää osio, jossa on seitsemän sarjaa nappuloita. Jokainen näistä sarjoista on jokin sointu (engl. chord). Sarjan alin nappula soittaa kokonaisen soinnun ja muut nappulat vain yhden sointuun kuuluvan sävelen. Tämän järjestelyn tarkoituksena on helpottaa säestämistä. Helpoin tapa säestää on käyttää ainoastaan koko soinnun sisältävää nappulaa. Käyttäjä voi myös näppäillä tai pyyhkäistä sormella eri rytmeillä soinnun nappuloiden yli samaan tapaan kuin kitaralla soitettaessa. Kokema-tonkin soittaja pystyy tällaisella järjestelyllä saamaan helposti aikaan jotain hyvän kuu-loista. Oikean puolen osiolla taas pystyy soittamaan normaaliin tapaan ja se mahdollis-taa monipuolisemman soittamisen. Soitto-osiot on järjestetty pystyyn, koska se on tun-tunut helpolta asennolta käsille soittaa.

## 5.2 Ominaisuudet

Oikeassa reunassa olevan melodiaosion sävellaji on käyttäjän vaihdettavissa. Ruudun alareunan palkin oikeanpuoleisimmasta napista (Tuning) aukeaa pudotusvalikko, jossa on lista valittavista sävellajeista. Käytän tässä suomeksi sanaa sävellaji, vaikka ohjelmassa käytetään sanaa tuning, mikä tarkoittaa viritystä, jolla viitataan esimerkiksi jonkin kielisoittimen viritukseen tiettyyn sävellajiin. Kuviossa 5 sävellajin valintavalikko.



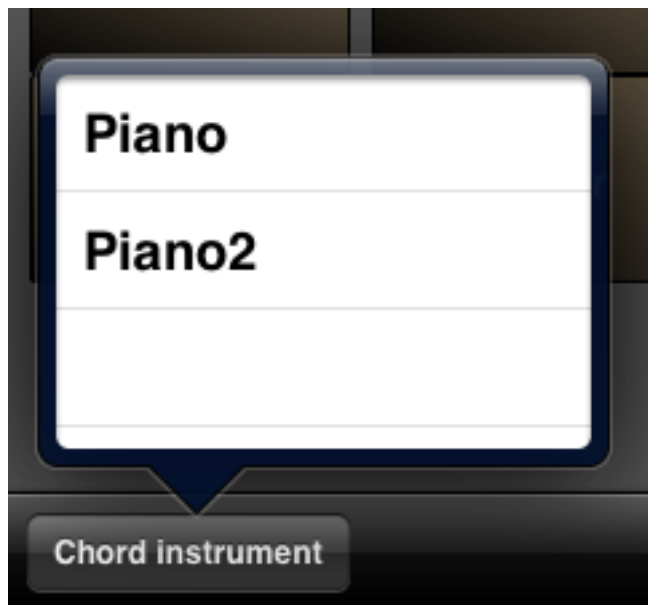
Kuvio 5: Sävellajin valinta ohjelmassa

Melodiaosiossa on merkitty eri väreillä C- ja G-sävelet. C-sävelet ovat punaisia ja G-sävelet vihreitä. Tämä helpottaa soittamista, koska näin soittaja näkee helposti, missä kohtaa tietyt sävelet ovat. Sävelten nimiä en halunnut kirjoittaa nappeihin koska se ei näyttäisi hyvältä ja olisi sekavan näköistä. Tällainen värikoodaus on myös soittajalle nopeampaa tulkita. Jos sävellajin vaihdos aiheuttaa näiden sävelten paikan vaihtumisen, myös värien paikat seuraavat mukana.

Kun käyttäjä koskettaa jotain soittimen nappia, äänen kuulumisen lisäksi sen väri vaihtuu. Kun käyttäjä irrottaa sormensa nappulalta tai liu'uttaa sen toiselle nappulalle, väri vaihtuu takaisin alkuperäiseksi. Näin soitin ei ole täysin staattinen vaan se myös näyttää reagoivan käyttäjän tekemiseen.

Kun käyttäjä koskettaa sointuosion alinta, eli soinnun kaikki sävelet soittavaa nappia ja sointuosiossa on vielä muiden sävelten soiminen kesken, muut sävelet sammutetaan. Sointuja nopeaan tahtiin soitettaessa soinnut saattaisivat muuten soida päällekkäin ja mennä sotkuisen kuuloiseksi. Melodiaosion ääniin tämä ei vaikuta.

Sointu- ja melodiaosion soitinäänet voidaan vaihtaa toisistaan riippumatta. Alapalkin vasemmanpuoleisimmalla (Chord Instrument) napilla voidaan vaihtaa sointuosion äänet ja keskimmaisella napilla (Melody Instrument) melodiaosion äänet. Kuviossa 6 soitinäänen vaihtovalikko.



Kuvio 6: Soittimen valinta ohjelmassa



## 6 Toteutus

Tässä luvussa käsitellään ohjelman rakennetta sekä esitellään ohjelman luokat ja olennaisten asioiden toteutustavat.

### 6.1 Ohjelman rakenne

Käyttöliittymästä ainoastaan alhaalla oleva palkki ja siinä olevat napit on toteutettu käyttöliittymäeditorissa. Itse soitto-osion käyttöliittymä on toteutettu koodissa.

#### 6.1.1 `NoteButtonView`- ja `ChordButtonView` –luokat

Peruskomponenttina ohjelmassa on yksi nappula. Nappulaluokkia on kahdentyyppisiä: yhden sävelen soittava nappula, `NoteButtonView`, ja kokonaisen soinnun soittava nappula, `ChordButtonView`. Nämä molemmat periytyvät samasta luokasta `ButtonView`, joka taas periytyy UIKitin luokasta `UIImageView`. Nappulat itsessään eivät soita ääniä vaan niihin on toteutettu ainoastaan nappulan kuvan näyttäminen. `NoteButtonView`iin on tallennettu tieto nappulan midi-arvosta ja sävelen nimi. `ChordButtonView`iin on toteutettu soinnun nimen tulostaminen nappulan päälle.

`UIImageView` on yksinkertainen luokka, jonka tarkoitus on nimensä mukaisesti esittää kuva. Luokan voi asettaa `highlighted`-tilaan, jolloin luokka näyttää normaalin kuvan sijaan toisen kuvan, jonka kehittäjä voi asettaa luokan alustuksessa tai myöhemminkin. Tällä tavalla on helppo toteuttaa nappi, jonka ulkoasu muuttuu sitä painettaessa.

`ButtonView` itsessään ei lisää `UIImageView`iin muita ominaisuuksia kuin kytkee päälle `userInteractionEnabled` –ominaisuuden, jota ilman kosketusten seuraaminen ei toimi. Se on haluttu kuitenkin tehdä yhteiseksi kantaluokaksi erityyppisille nappuloille, koska niitä asetetaan samaan taulukkoon ja käsitellään piirtämisvaiheessa niin, että eroa ei tehdä. Tässä voisi käyttää suoraan `UIImageView`iä tai Objective-C:n id-tietotyyppiä, joka voi olla osoitin minkä tahansa tyyppiseen luokkaan, mutta näin koodiin saadaan selkeästi näkymään mitä siinä käsitellään.

### 6.1.2 PlayView-luokka

PlayView kokoaa ryhmän nappuloita yhdeksi kokonaisuudeksi. Ohjelmassa käytettyjä erillisiä kokonaisuuksia ovat melodiaosio ja sointuosion yhteen sointuun liittyvät napit. PlayView-luokka hoitaa kosketusten hallinnan ja ohjaa TouchInstrumentViewController-luokassa olevaa äänen soittamista.

PlayView-luokkaa ei käytetä sellaisenaan vaan siitä perityvät luokat MelodyView ja ChordView. Suurin osa toiminnallisuudesta on toteutettu PlayView-luokassa. MelodyView luokka lisää tähän värikorostuksen lisäyksen. ChordView lisää PlayViewiin sointuosion sointunapin piirtämisen.

### 6.1.3 TuningManager-luokka

TuningManager lukee soinnut ja sävellajit tiedostoista ja hoitaa niiden muodostamiseen liittyvän laskemisen. Muualta ohjelmasta voidaan kysyä TuningManagerilta koko melodiaosion sävelet tai tietyn soinnun sävelet. TuningManagerin toimintaa on tarkasteltu tarkemmin myöhemmin.

### 6.1.4 TouchInstrumentViewController-luokka

TouchInstrumentViewController on ohjelman pääluokka, joka kokoaa kaikki tarvittavat elementit ohjelman käyttöliittymään. Lisäksi se luo SoundBankPlayerit ja vastaanottaa playView-olioiden lähettämät viestit ja hoitaa niiden perusteella äänen soittamisen hallinnan.

Käyttöliittymän alapalkki ja sen napit on luotu käyttöliittymäeditorilla. TouchInstrumentViewController sisältää metodit näiden nappien painalluksiin reagointiin ja pudotusvalikoiden piirtämiseen.

## 6.2 Äänen soittaminen

Äänen toistamista varten on tähän työhön valittu erillinen SoundBankPlayer –kirjasto. Äänen toistaminen ilman tätäkin, iOS:n omilla ominaisuuksilla, on hyvin helppoa, mutta kirjaston tarkoituksena on pienentää ohjelman kokoa ja vähentää tarvittujen äänitie-

dostojen määrää. (Hollance: SoundBankPlayer: Using OpenAL to Play Musical Instruments in Your iOS App)

SoundBankPlayer tarvitsee vain osan käytetyistä äänitiedostoista. Loput äänet se muodostaa muuttamalla lähimpänä olevan äänitiedoston korkeutta. Tämä huonontaa äänenlaatua jonkin verran, mutta ei merkittävästi, kunhan ääniä on sopivan pienin välein. Suoriin tiedostonimiin ei viitata vaan haluttu sävel ilmoitetaan SoundBankPlayerille MIDI-numeroarvoina. Tämä helpottaa sävellajien ja sointujen muodostamista, koska ne pystytään laskemaan, eikä niiden jokaista säveltä tarvitse listata erikseen.

SoundBankPlayeriin voi lisätä säveliä myös jonoon ja sen jälkeen ne soitetaan kerralla `playQueuedNotes` –metodilla. Tätä on käytetty hyväksi kokonaisen soinnun soittamisessa.

SoundBankPlayer luodaan ohjelmaan normaalina oliona. Alustamisen jälkeen sille täytyy ilmoittaa, mitä äänitiedostoja käytetään, jonka jälkeen se on valmis käytettäväksi. SoundBankPlayer-olioita on ohjelmassa kaksi. Toinen on melodiaosiolle ja toinen sointuosiolle. Näin osioille voidaan valita eri äänet, eikä melodiaosion äänten toisto häiriinny sointuosion mahdollisesta kaikkien äänten sammuttamisesta.

### 6.3 Kosketusten seuraaminen

Käyttäjä pystyy käyttämään kaikkia kymmentä sormeaan soittamiseen. Koska kosketuksia voi olla samaan aikaan useita, niiden seuraamiseen tarvitaan enemmän koodia kuin vain yhden kosketuksen seuraamiseen.

Kun kosketus havaitaan, tarkistetaan osuuko se mihinkään nappiin. Jos kosketus on napin alueella, se korostetaan ja soitetaan nappiin kuuluva ääni. Jos kosketus ei osu mihinkään nappiin, ei reagoida mitenkään.

iOS luo jokaisesta havaitusta kosketuksesta `UITouch`-tyyppisen olion, joka säilyy koko kosketuksen elinkaaren ajan. Kosketusten seuraamista varten tarvitaan uusi tietotyyppi, joka sisältää osoittimet kosketettuun nappiin ja kosketuksen olioon. Tätä tietotyyppiä tarvitaan, kun kosketus liikkuu ruudulla ja halutaan reagoida tilanteeseen, jossa kosketus siirtyy alkuperäisen napin päältä toiselle. Joka kerta kosketuksen liikkuesssa tarkiste-

taan, onko kosketus vielä saman napin päällä, joka tietotyyppiin on merkitty. Jos nappi on eri, poistetaan vanhan napin korostus ja tehdään uuden napin kohdalla samat toimenpiteet kuin uuden kosketuksen osuessa nappiin.

Kosketuksen loppuessa eli sormen irrotessa ruudulta, tarkistetaan, painoiko se jotain nappia ja poistetaan napin korostus. Sen jälkeen poistetaan kyseiseen kosketukseen liittyvä keyDownInfo-olio.

## 6.4 Sävellajien ja sointujen määrittely ja käsittely

Tässä kappaleessa tarkastellaan sointu- ja sävellajitiedostojen muotoa sekä tapoja, joilla niitä käsitellään. Ohjelmassa on TuningManager -niminen luokka, joka hoitaa kaiken tähän liittyvän toiminnallisuuden. Toisin kuin musiikin teorialuvussa on mainittu, ohjelman toteutuksessa sävelen oktaaviluku merkitään ensimmäiseksi merkiksi eikä vasta viimeiseksi. Tämä helpottaa sävelten käsittelyä ohjelmassa. Kaikki tämän kappaleen aliluvuissa mainitut metodit sijaitsevat TuningManager-luokassa.

### 6.4.1 Sävellajitiedoston muoto

Sävellajitiedosto Tunings.plist sisältää tiedot sävellajeista Dictionary-tyyppisessä muodossa eli avain-arvo -parina. Avaimena on soinnun nimi ja arvona sävelet yhdessä merkkijonossa pilkuilla eroteltuna. Sävel-merkkijono sisältää vain sävelten nimet ilman tietoa oktaavista. Kuviossa 7 on esimerkki sävellajitiedoston sisällöstä, johon on tallennettu kolme eri sävellajia.

Key	Type	Value
▼ Root	⊕ Dictionary	⬆ (3 items)
CMajor	String	C,D,E,F,G,A,B
DMajor	String	C#,D,E,F#,G,A,B
GPentatonicMinor	String	C,D,F,G,A#

Kuvio 7: Sävellajitiedoston sisältö

### 6.4.2 Sointutiedoston muoto

Sointutiedosto sisältää soinnun tyypin ja siihen kuuluvat sävelet. Sointu on ilmoitettu ainoastaan tyyppinä ilman juurisäveltä ja sävelet numeroina, jotka ilmoittavat, monesko puoliaskel juurisävelestä laskettuna se on. Juurisävel on numero yksi. Näin riittää, että sointujen tyypit kirjataan tiedostoon ja kaikki saman tyyppiset soinnut ovat laskettavissa koodissa mille tahansa juurisävelelle. Kuviossa 8 on esimerkki sointutiedoston sisällöstä, johon on tallennettu kahden eri sointutyyppin tiedot.

Key	Type	Value
▼ Information Property List	Dictionary	(2 items)
Major	String	1,5,8
Minor	String	1,4,8

Kuvio 8: Sointutiedoston sisältö

loadList-metodi avaa parametrina annetun tiedoston ja tallentaa sen NSDictionary-muotoon. Samaa metodologia käytetään sekä sointu-, että sävellajitiedoston lukemiseen.

### 6.4.3 Sävelen nimen muuntaminen midi-arvoksi

TuningManager-luokassa on yksi luokkametodi, noteToMidiValue, joka muuntaa sävelen nimen midi-arvoksi. Koska se on luokkametodi, sitä voi käyttää myös ilman, että luokasta on tehty oliota.

Sävelen nimi on merkkijono, jonka ensimmäinen merkki on oktaavin numero. Toinen merkki on sävelen nimi. Merkkijonossa voi olla vielä kolmas merkki, '#' tai 'b', jotka tarkoittavat ylennystä tai alennusta.

Ensimmäiseksi oktaavilukuun lisätään luku 1, koska oktaavien numerointi alkaa siten, että sävelen C0 midi-arvo on 12, eikä 0 kuten voisi olettaa. Saatu luku kerrotaan luvulla 12, jotta saadaan kyseisen oktaavin C-sävelen midi-arvo.

Jokaiselle oktaavin sävelelle on annettu arvo, joka kertoo kuinka monen puoliaskelen päässä se on ensimmäisestä eli C-sävelestä. C-sävelen arvo on 0. Seuraavaksi edellisen

laskun tuloksena saatuun lukuun lisätään määritettävän sävelen arvo. Näin on saatu kyseisen sävelen MIDI-arvo laskettua.

Jos sävelessä on vielä kolmantena merkkinä ylennys- tai alennusmerkki, lisätään ylenyksessä MIDI-arvoon luku yksi ja alennuksessa vähennetään luku 1.

#### **6.4.4 Sävellajin vaihtaminen**

TuningManagerin `changeTuning` –metodilla voidaan vaihtaa soittimen sävellaji. Metodi tyhjentää ensin TuningManagerin taulukon, joka sisältää nykyisen sävellajin sävelet. Sen jälkeen metodi laskee taulukkoon uuden sävellajin mukaisia säveliä tarpeeksi monta kattamaan melodiaosion tarvitsemat sävelet. Tulokseksi saadusta taulukosta melodiaosio voi ottaa sopivasta kohdasta pätkän säveliä nappeihinsa.

Esimerkiksi sävellaji `DMajor` sisältää sävelet `C#, D, E, F#, G, A, B`. Näitä säveliä käydään läpi järjestyksessä monta kertaa ja jokaisella kierroksella sävelen eteen laitetaan numero joka on edellistä kierrosta yhtä suurempi. Numerointi aloitetaan nolasta. Käsittelyn jälkeen TuningManager sisältää taulukon, jonka sisältö voi olla esimerkiksi `2C#, 2D, 2E, 2F#, 2G, 2A, 2B, 3C#, 3D, 3E, 3F#, 3G, 3A, 3B`.

Melodiaosio ottaa tehdystä taulukosta sävelet nappeihinsa ja muuntaa ne `noteToMidiValue`-metodilla midi-muotoisiksi. Samalla melodiaosio tarkastaa, ovatko nappien korostusvärit oikeilla paikoilla sävellajin vaihdoksen jälkeen ja asettaa ne oikein.

#### **6.4.5 Soinnun sävelten määrittäminen**

TuningManagerin `getChord`-metodi palauttaa taulukon, joka sisältää sointuun kuuluvia säveliä midi-arvoina. Parametrina se ottaa pyydetyn soinnun nimen. Nimestä tarkistetaan ensin ensimmäinen merkki, joka on juurisävel. Loput nimestä on soinnun tyyppi, jonka tiedot löytyvät sointutiedostosta.

Esimerkkinä tässä on sointu `CMinor` eli C-molli-sointu. Kuviosta 8 nähdään, että sointutiedostossa sen säveliksi on merkitty 1, 4, 8. Aluksi muodostetaan ensimmäinen sointuun kuuluva sävel oktaaviin 0 eli juurisävelen ollessa C ensimmäinen sävel olisi 0C. Tämä muutetaan midi-arvoksi, jonka jälkeen seuraava sävel muodostetaan ensimmäisen

sävelen midi-arvosta lisäämällä siihen sointutiedostossa seuraavana oleva luku. Kolmas sävel muodostetaan taas lisäämällä ensimmäiseen arvoon kolmas luku. Tässä esimerkissä soinnussa on kolme säveltä, mutta niitä voi olla enemmänkin. Kun kaikki kolme säveltä ovat muodostettu muodostetaan juurisävel seuraavasta oktaavista eli 1C ja tehdään samat toimenpiteet kuin edelliselle oktaaville.

## 7 Yhteenveto

Aloittaessani tätä opinnäytetyötä en tiennyt mitään iOS-laitteille ohjelmoinnista. Tämän opinnäytetyön rinnalla olen myös tehnyt toista projektia iOS-käyttöjärjestelmälle, jossa käytin UIKitin sijaan Cocos2D-sovelluskehystä, joka on tarkoitettu pelikehitykseen. Näin olen oppinut hyvin ja laajasti käyttämään iOS:lle tarkoitettuja ohjelmointityökaluja ja oppinut monenlaisia toimintatapoja.

Aluksi Objective-C vaikutti sekavalta ulkoasunsa vuoksi. Siinä käytetty syntaksi metodien kutsumiseen poikkeaa siitä, mihin olen muissa kielissä tottunut. Kun sitä kuitenkin oppi lukemaan ja siihen tottui, huomasin, että se tuntuu mukavalta kieleltä käyttää. Eri-tyisesti pidän siitä, että kaikilla parametreilla on nimi, jonka vuoksi rivit ovat usein luettavissa selkeinä lauseina ja helposti ymmärrettävissä.

Kehitysympäristöstä olen myös pitänyt, koska se on ollut helppo käyttää. Sisäänrakennettu versionhallinta on ollut hyvä lisä. Huonona puolena voisi sanoa laitteilla testaamisen hankaluuden, koska sitä varten tarvitsee tehdä monta toimenpidettä. Sen kanssa on ollut myös ongelmia, eikä asia ole ensimmäisellä yrittämällä lähtenyt toimimaan. Onneksi kaikki on toiminut sen jälkeen kun asiat on kerran laittanut kuntoon.

Koska iPadille ohjelmoidessa ei tarvitse ottaa huomioon kuin yksi näytön tarkkuus, käyttöliittymän suunnittelu ja toteutus on helppoa. Minulla on jonkin verran kokemusta myös Android-laitteille ohjelmoinnista ja mielestäni sen kanssa käyttöliittymän toteuttaminen oli hankalampaa, koska laitteissa voi olla lähes mikä tahansa näytön tarkkuus. Kehitysympäristö ei omien kokemusten perusteella myöskään toiminut Androidille ohjelmoidessa ongelmattomasti.

Esimerkkinä tekemäni ohjelma onnistui odotusten mukaisesti. Vaikka onkin hieman vaatimaton ominaisuuksiltaan, se tekee sen mitä pitää hyvin, enkä ole havainnut ohjelmassa virheitä. Julkaisukelpoinen se ei nykytilassaan ole ulkoasunsa takia. Julkaisukelpoiseksi saattamiseen projektiin pitäisi saada graafikko suunnittelemaan ja piirtämään käyttöliittymän kuvat. Ohjelman tekeminen toi kuitenkin hyvää kokemusta musiikin tuottamiseen soveltuvan ohjelman tekemisestä. Aihe kiinnostaa minua, joten tämä työ saattaa ehkä toimia tulevaisuudessa pohjana jollekin isommalle projektille.



## LÄHTEET

Apple Developer: Choosing an iOS Developer Program | Luettu 13.10.2012  
<https://developer.apple.com/programs/start/ios/>

Apple Developer: Distribute your App - iOS Developer Program | Luettu 13.10.2012  
<https://developer.apple.com/programs/ios/distribute.html>

Apple Developer: The Objective-C Programming Language: Introduction | Luettu 12.11.2012  
<https://developer.apple.com/library/mac/#documentation/Cocoa/Conceptual/ObjectiveC/Introduction/introObjectiveC.html>

Apple: iOS 6 | Luettu 10.10.2012  
<http://www.apple.com/ios/>

Apple – iPad 2 – View the technical specifications for iPad 2 | luettu 13.11.2012  
<http://www.apple.com/ipad/ipad-2/specs.html>

Apple Special Event March 2011, 49:50 | Katsottu 13.11.2012  
<http://www.apple.com/apple-events/march-2011>

Apple: What is iOS | Luettu 10.11.2012  
<http://www.apple.com/ios/what-is/>

Hollance: SoundBankPlayer: Using OpenAL to Play Musical Instruments in Your iOS App | Luettu 10.10.2012  
<http://www.hollance.com/2011/02/soundbankplayer-using-openal-to-play-musical-instruments-in-your-ios-app/>

John Classick: Names of Notes and Intervals | Luettu 13.11.2012  
<http://www.johnclassick.com/names-of-notes-and-intervals/>

Macworld: How iOS multitasking really works | Luettu 5.11.2012  
[http://www.macworld.com/article/1164616/how\\_ios\\_multitasking\\_really\\_works.html](http://www.macworld.com/article/1164616/how_ios_multitasking_really_works.html)

The Verge: iOS: A visual history | Luettu | 11.10.2012  
<http://www.theverge.com/2011/12/13/2612736/ios-history-iphone-ipad>