KARELIA-AMMATTIKORKEAKOULU
Tietojenkäsittelyn koulutus

Petteri Anttila

Musiikin rytmihavainnon käyttö pelisovelluksen rakentamisessa: BEATRUNNER-PELISOVELLUS

Opinnäytetyö
Joulukuu 2015

Karelia AMMATTIKORKEAKOULU

OPINNÄYTETYÖ
Joulukuu 2015
Tietojenkäsittelyn koulutus
Karjalankatu 3
80220 JOENSUU
+358 50 468 3679

Tekijä
Petteri Anttila

Nimeke
Musiikin rytmihavainnon käyttö pelisovelluksen rakentamisessa: BEATRUNNER-PELISOVELLUS

Tiivistelmä

Lähes kaikissa tietokonepeleissä taustamusiikki ja ääniefektit on yhdistetty peliin, mutta rytmihavainnon käyttö pelisovelluksen rakentamisessa on harvinaisempaa. Tämän opinnäytetyön aiheena oli tutkia, miten rytmihavaintoa voidaan käyttää pelisovelluksen rakentamisessa.

Opinnäytetyön osana kehitettiin tietokonepelin prototyyppi, jota hyödynnettiin rytmihavainnon tutkimuksessa. Rytmihavaintoon perustuva tasohyppelypelin prototyyppi lisäsi työhön käytännön näkökulmaa. Proto-ohjelma rakennettiin Unity-ympäristöön käyttäen Visual Studio -ohjelmointiympäristöä C#-kieleltä hyödyntäen. Työssä käytettiin hyväksi verkosta löytyviä ohjelmistokirjastoja.

Opinnäytetyön teoriaosuudessa tarkastellaan ensin digitaalisen musiikin teoriaa ja eri tallennusmuotoja. Tämän jälkeen työssä käsitellään rytmihavainnon alkeita sekä esitellään rytmihavainnon matemaattinen kaava.

Opinnäytetyön käytännön osuudessa esitellään tarkemmin pelikehitysprosessia ja siinä käytettyjä työkaluja. Opinnäytetyöprosessin tuloksena syntyi pelin prototyyppi, jonka testituloksia sekä käyttökelpoisuutta arvioidaan raportin lopussa.

Kieli
suomi

Sivuja 24

Asiasanat
peliprojekti, BEATRUNNER, Unity, Bass.Net, rytmihavainto, musiikki, digitaalinen ääni

THESIS
Desember 2015
Business Information Technology
Karjalankatu 3
80220 JOENSUU
+358 50 468 3679

Author
Petteri Anttila

Title
Using beatdetection in building video game program: BEATRUNNER-Gameprogram

Abstract
In almost all computer games background music and sound effects are connected to game, but the use of the beat detection in dynamic games (games that build its levels in game and doesn't use premade levels) is less common. The theme of this theses is to research how the beat detection can be used in building games.. The aim of this study was to examine how the rhythm of discovery can be used in the game application in construction.

I developed a game prototype which enables research on the topic of utilization of beat detection in platformer games. Prototype was build using Unity as an environment and Visual Studio as a programming environment using C# programming language and BASS library.

The theoretical part of the thesis we first learn about digital music theory and different formats. After this, we look at the basics of beat detection and present the mathematical formula of beat detection.

The practical part of the thesis we take a look at how the prototype was made and what tools were used to achieve it and at the end we evaluate prototypes test results and overall usefulness.

Language
Finnish

Pages 24

Keywords
Game project, BEATRUNNER, Unity, Bass.Net, rhythm detection, music, digital audio

Sisällys

1 Johdanto

Tämän projektin tavoitteena on selvittää, kuinka musiikin avulla on mahdollista luoda pelikenttä, jossa esteet sijoittuvat ja määrittyvät musiikin mukaan. Projektissa käytetään kappaleita mahdollisimman monesta musiikin lajityypeistä: rockista, klassisesta musiikista, dubstep, heavy metal. Projektissa tehdään pelikenttä reaaliajassa siten, että musiikin rytmin havainto muodostuu sekunnin ennen musiikin kuuntelua.

Opinnäytetyössä kehitetään pelin prototyyppi, jossa luodaan pelikenttä musiikin rytmin mukaan ja pelaajan on tarkoitus selviytyä esteistä musiikin loppuun asti. Esteet muodostuvat suorakulmioista ja kuutioista, jotka luodaan musiikista rytmisignaalin mukaan. Este näkyy pelaajalle noin sekunnin ajan ennen kuin sitä pitää väistää. Pelissä hahmona on Pacman-testihahmo.

Tässä työssä tarkastellaan eri tekniikoita musiikin signaalin havainnointiin. Tämän jälkeen testataan tekniikoita eri musiikkilajeilla, joita hyödynnetään pelin prototyypissä. Opinnäytetyön lopputuloksena valmistuu prototyyppi, jota myöhemmin voidaan hyödyntää mallina ja ohjeena erilaisissa projekteissa, joissa tavoitteena on tuottaa sisältöä tai toimintoa musiikin avulla.

Prototyypin rakentamisessa käytin verkosta yksityiseen käyttöön ladattavaa ilmaista Bass.Net-kirjastoa rytmihavainnon suorittamiseen. Ohjelman rakensin Unity-ympäristöön käyttäen Visual Studio -ohjelmointiympäristöä C#-kielellä.

Opinnäytetyöraportin alussa tarkastellaan digitaalista musiikkia yleisellä tasolla. Tämän jälkeen luvussa kolme käsitellään rytmihavainnon perusteet, lyhyesti Fast fourier transformia sekä lopuksi koodausympäristöä ja valmiita tarjolla olevia koodeja. Luku neljä käsittelee peliprototyypin käyttöympäristöä ja siinä käytettyjä tekniikoita. Lopuksi esitellään ja arvioidaan tutkimuksen tulokset sekä opinnäytetyön prosessin kulkua.

2 Digitaalinen musiikki

Digitaalisella musiikilla on kaksi erilaista tiedostotyyliä: häviötön ja häviöllinen. Häviötön pitää äänen laadun enimmäkseen alkuperäisenä CD-tasoisena, kun taas häviöllinen pakkaa musiikin pienempään tilaan ja samalla menettää hiukan laadussa. Häviöttömään tyyliin kuuluvat muun muassa WAV, AIFF, FLAC, APE, ALAC ja häviölliseen tyyliin MP3, AAC, OGG, ja WMA. Suurin osa nykyajan musiikista on digitaalisessa muodossa. Digitaalisessa musiikissa on kyse siitä, että analoginen signaali muutetaan digitaaliseksi eli tietokoneen ymmärtämään muotoon. Ensimmäisiä versioita digitaalisesta musiikista tuli CD-levyn muodossa vuonna 1982. (Ruotsala 2014; Whitson 2012.)

Ihmisen kuuloaistia rajoittavat kaksi päätekijää: korvan kuulokynnys ja nk. peittoilmiö, joka tarkoittaa, että tietyt perusäänet peittävät heikompia äänen osasignaaleja. Ääniaaltoja on mahdollista muokata ja käsitellä, esim. ihminen voi laittamalla pumpulitupot korviinsa vaimentaa tulevia ääniaaltoja ja megafonin avulla vahvistaa. Eri instrumenteista tulevat äänet ovat tulos fysikaalisista ilmiöistä, kuten ilmapatsaan tai kielen värähtelystä. Koska ääni on ilmanpaineen vaihtelua ilmassa, voidaan se muuttaa esim. jännitevaihteluksi. (Keskinen, Hoikkala & Korkala 1999.)

Elektroninen media tarjoaa suuria mahdollisuuksia äänen muokkaamiseen. Elektronisessa muodossa ääntä voidaan mm. suodattaa, muokata sen aika-arvoja, kääntää takaperin, vertailla toiseen informaatioon, lajitella, kasvattaa tai pienentää intensiteettiä ja säröttää. Digitaalista informaatiota on mahdollista prosessoida eri prosessointilaitteilla, kuten esim. *sekvenssereillä* (music sequencers) ja *kovalevyäänittimillä* (direct-to-disk audio recorders). (Holm 1998.)

Eräs elektronisen signaalin muokkaamisen perusparametreista on ääniaaltojen *amplitudin* (äänenvoimakkuus) muokkaaminen. Tämä tapahtuu *linjavahvistimien* avulla. Lisäksi ohjelmasignaalia on mahdollista muokata lukemattomilla muilla tavoilla. Tätä varten on *suotimia* (filters), jotka toimivat eri medioita yhdistävinä

prosessoreina. Nämä valikoivat tarvittavan informaation, äänen tapauksessa halutut taajuudet. Suotimia ovat mm. *sulkusuotimet* (cut-off filters), *yli- ja alipäästösuotimet* (high- and low-pass filters) sekä *kaistansuotimet* (band-pass filters). (Holm 1998.)

Signaalia pystytään myös tuottaa ja syntetisoida elektronisesti. Tästä voi olla joskus hyötyäkin. Esimerkiksi edullinenkin laite voi tuottaa laaja-alaisemman äänen kuin yksikään mekaaninen instrumentti tai luonnollinen äänilähde. Elektronisia signaaleja tuottava komponentti on nimeltään *oskillaattori* (oscillator), joka on nimetty muodostamiensa oskilloivien aaltojen mukaan. Monimuotoisemmat signaalit muodostetaan modulaatiotekniikalla. Siinä signaaliin vaikutetaan toisen aallon tiedossa olevilla parametreillä (taajuus, amplitudi, pulssinleveys). Syntetisaattoripiirien äänenprosessointi perustuu pitkälti juuri tähän. (Holm 1998.) Äänen korkeus muodostuu sen värähtelytaajuudesta. Mikäli näytteitä otetaan esim. 11 000 kertaa sekunnissa eli näytetiheys on 11 kHz, mutta äänen taajuus on vaikka 15 kHz eli 15 000 värähdystä sekunnissa, ei kaikista värähdyksistä saada ollenkaan näytettä. Mikäli äänen taajuus on sama kuin näytetiheys niin kustakin ääniaallosta saadaan vain yksi näyte, joka ei riitä. Vasta kun jokaisesta aallosta saadaan vähintään kaksi näytettä (positiivinen ja negatiivinen amplitudin taso), muodostuu niistä suurin piirtein oikea kuva. (Holm 1998.)

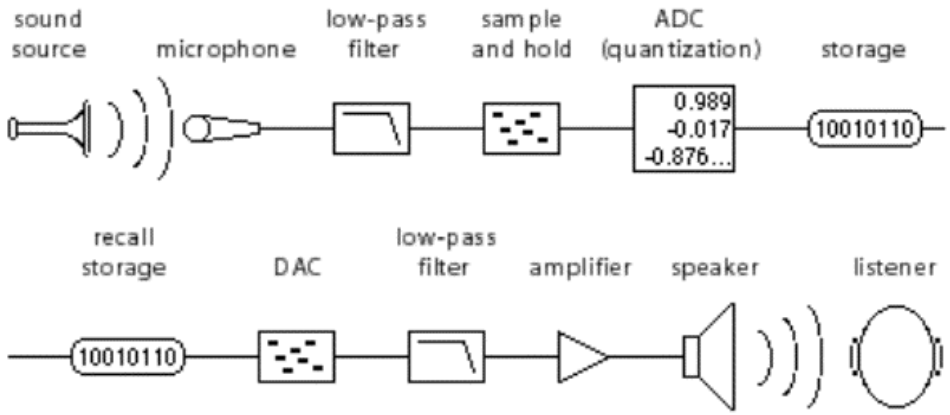
Nyquistin teoreeman mukaan tietyn taajuisen äänen toistumiseen vaaditaan ainakin kaksinkertainen ja korkealaatuiseen nelinkertainen näytteistystaajuus. Näytteistystaajuudet esitetään (Taulukko 1) aina kanavakohtaisesti laskettuina, joten jos esimerkiksi äänitettäessä stereodataa 8000 näytettä sekunnissa, on näytteitä oikeastaan 16000 sekuntia kohden. (Holm 1998.)

Taulukko 1. PCM-standardin näytteistystaajuudet (Holm 1998).

44,1 kHz	CD-äänilevyn standardi
22,1 kHz	keskiaaltoradion taso
11,025 kHz	puheen tallennus, hyvän puhelinlinjan taso
5,5 kHz	vastaa huonon puhelinlinjan tasoa

2.1 Äänen tallennus

Tietokoneessa äänenkäsittely on aina digitaalista. On mahdollista että ääni tulee useasta eri äänilähteestä, jotka voidaan yhdistää tietokoneessa. Analogisen audiosignaalin lukemiseen tarvitaan esimerkiksi äänikortti. Digitaalisessa aaltomuodossa tallennetun äänen voidaan ajatella olevan kuin äänilähteestä tulevien ääniaaltojen kuva. Digitaalinen ääni talletetaan arkistoissa valittuun tallennusmuotoon. Tallennus ja toisto tapahtuvat tietokoneen äänijärjestelmän toimiessa digitaalisen ääninauhurin tapaan. Ääniaallot, puhe, musiikki tai hälyäänet muutetaan sähköiseksi, analogiseksi värähtelyksi, joka johdetaan tietokoneen äänijärjestelmän analogia/digitaalimuuntimeen. (Holm 1998.)



Kuva 1. Äänen tallennus (Mentholorange 2013).

Ääni digitoidaan tietokoneeseen liitetyllä A/D-muuntimella (Analog/Digital Converter). Tämä on mikropiiri, jolla otetaan värähtelystä näytteitä tasavälein (ts. aalto 'katkaistaan') ja siitä otetaan 'poikkileikkaus', joka kertoo aallon korkeuden (amplitudin) katkaisuhetkellä. Tämä tieto tallennetaan numeerisessa muodossa. Äänen digitoinnissa vallitsevat yleiset digitoinnin pelisäännöt. Näytteenottotaajuus vaikuttaa äänen kirkkauteen. A/D-muunnoksen tasojen määrä (kvantisointi) määrää digitoidun äänen dynamiikan eli hiljaisten ja voimakkaiden äänien suhteen. Molemmat vaikuttavat digitoidun äänen sisältämään tiedon määrään ja tiedoston kokoon. Kun näytteitä otetaan riittävän tiheästi, esimerkiksi 22000 kertaa sekunnissa, niistä saadaan hyvä kuva alkuperäisestä ääniaallosta. (Holm 1998.)

2.2 Yleiset tallennusmuodot

MP3

Kehittäjä: Motion Picture Experts Group

MP3 on lyhenne MPEG audio layer 3 -tyylistä. Layer 3 on yksi kolmesta koodaustavasta äänen pakkaamiseen. Layer 3 käyttää käsitteellistä audiokoodausta ja psykoakustista pakkausta poistamaan kaikki tarpeettomat tiedot ja merkityksettömät osat äänisignaalista, kuten asiat, joita ihmisen korva ei kuule. Layer 3 myös lisää MDCT:n (Modified Discrete Cosine Transform), joka implementoi suodatinpankin (filter bank). MP3:n lisenssimaksu on 0,75 dollaria MP3:n purkuun pystyvältä tai kaksinkertainen myös pakkaukseen pystyvältä laiteelta. Vähimmäismaksu on 15 000 dollaria vuodessa. (Vangie 2005.)

MP3 on vuonna 1991 kehitetty säästökoodausmenetelmä, joka mahdollistaa lähes CD-tasoisen äänen siirtämisen järkevässä ajassa nykyisillä tiedonsiirtonopeuksilla. MP3 on häviöllinen pakkausmenetelmä, joka pystyy pienentämään näytteen koon noin neljäsosaan alkuperäisestä. Paitsi matematiikkaan ja tietojenkäsittelyyn, MP3-koodaus perustuu myös akustiikkaan ja kuulotutkimuksen havaintoihin. (Keskinen ym. 1999.)

MP3-koodaus vähentää ääni-informaatiosta 5-20 prosenttia poistamalla kuulokynnystä heikommat sekä perusäänten peittämät signaalit. Se siis pelkistää signaalin hallitusti jättäen jäljelle alkuperäisen signaalin tuottamat kuuloelämykseen tarvittavat tekijät. (Keskinen ym. 1999.)

MP3-koodaus tapahtuu pääpiirteittäin seuraavasti:

Signaalista poistetaan kuulokynnyksen alapuolelle jäävät osat. Karsitaan signaalit, jotka sijoittuvat voimakkaiden äänien peittämälle taajuudelle. Jaetaan käytettävissä olevat bitit taajuussisällön ja voimakkuuden mukaan; voimakkaat signaalit ohjelmoidaan tarkasti ja heikompien signaalien tarkkuutta supistetaan. Ohjelmoitaessa stereoääntä voidaan lisätä enemmän bittejä sille kanavalle, joka on voimakkaampi tai jonka taajuuskirjo on laajempi. Lopuksi signaalille tehdään Huffman-koodaus, joka pakkaa bittidatan pienempään tilaan. Huffman-koodaus on yleisesti tietotekniikassa käytetty pakkausperiaate. Käytettävissä olevat bitit

osoitetaan ensisijaisesti signaalin merkittävimpien ja selvimmin erottuvien osuuksien koodaamiseen, muu osa signaalia koodataan pienemmällä bittimäärällä. (Keskinen ym. 1999.)

WMA – Windows Media Audio (.wma)

Kehittäjä: Microsoft

Kehitetty vuonna 1999

WMA on Microsoftin ensimmäinen äänitiedon pakkausmuoto, jonka Microsoft kehitti kilpailemaan suositun MP3-tyypin kanssa. Ensimmäinen WMA codec perustui aikaisempaan työhön, jonka kehitti Henrique Malvar ja hänen tiiminsä. WMA kulki MSAudio-nimellä projektin aikana, jonka jälkeen se nimettiin MSAudio 4.0:ksi. Lopulta se julkaistiin nimellä Windows Media Audio (WMA) osana Windows Media Technologies 4.0 -ohjelmistoa. Microsoft vakuutti että WMA pystyi kehittämään äänitiedostoja, jotka olivat puolta pienempiä kuin MP3-tiedosto sekä melkein CD-laatuista ääntä, mutta hifiharrastajat hylkäsivät väitteen CD-laatuisesta äänestä. RealNetworks halusi Microsoftin todistavan väitteensä WMA:n erinomaisesta äänenlaadusta verrattuna RealAudioon. (Wikipedia 2015a.)

WAV (.wav)

WAV on Microsoftin ja IBM:n vuonna 1991 kehittämä formaatti, jota käytetään äänen tallentamiseksi tiedostoon. Koska wav-formaatti ei ole pakattu, sen koko on suuri verrattuna esimerkiksi pakattuun MP3-formaattiin. Tästä syystä wav-tiedoston suosio internetissä jakamisessa on pieni. (Wikipedia 2015b)

Real Audio (.ra .ram .rm)

Real Audio on RealNetwork-yrityksen vuonna 1995 patentoima formaatti, jonka avulla voidaan siirtää äänidataa ja joka mahdollistaa digitaalisen äänitiedoston kuuntelemisen reaaliajassa. Tämän tiedoston käyttämiseen tarvitaan RealPlayer-ohjelmaa. Real Audio on yksityisomistuksellinen formaatti. (Wikipedia 2015c.)

Progressive Networkin RealAudio -teknologia mahdollistaa RealAudion player- ja server-ohjelmien välisen interaktiivisen linkityksen, jonka avulla voidaan toteuttaa äänilähetyksiä koko Internet-verkkoon. Lähetyspäässä tarvitaan palvelinohjelmisto, joka soveltuu laajamittaiseen verkossa tapahtuvaan

lähetystoimintaan. Lähetystoimintaan tarvitaan oma kotisivu ja kiinteä liittymä Internet-verkkoon. RealAudio Server -ohjelmisto on yhteensopiva minkä tahansa MIME-koodia tukevan palvelinohjelman kanssa, joita ovat esim. Netscapen Netsite ja Macin HTTPD. (Holm 1998.)

Riippuen äänen pakkaustiheydestä yksittäisen kanavan kaistanleveyden pitää olla 10 - 22 kb sekunnissa. RealAudio-ohjelmisto vaatii keskimäärin 2 MB kiintolevytilaa. Riippuen pakkaustiheydestä RealAudio-formaatissa olevat tiedostot vaativat keskimäärin 1.1-2.4 kb jokaista äänisekuntia kohden. Täten esimerkiksi yhden tunnin pituinen lähetys vaatii levytilaa 3.6 MB:stä 8 MB:aan. (Holm 1998.)

RealAudio encoder on ohjelma, jolla voidaan muuntaa eri ääniformaatissa olevia tiedostoja (esim. wav, au, pcm jne.) RealAudio-formaattiin. Koodattu RealAudio-materiaali voidaan soittaa Internetissä reaaliaikaisesti, käyttämällä asianmukaisia ohjelmia (player ja server). Koodaus voidaan suorittaa kahden vaihtoehtoisen algoritmin mukaisesti. RealAudio 14.4 -algoritmi mahdollistaa monofonisen AM-tasoisen äänen, joka vaatii vähintään 14.4 Kbps modeemiyhteyden Internetiin. 28.8 -algoritmi mahdollistaa lähes FM-tasoisen äänen ja vaatii vähintään kaksi kertaa nopeamman yhteyden. RealAudio-kooderi on suunniteltu toimimaan Windows 95 ja Windows NT -ympäristöissä sekä Macintoshissa. (Holm 1998.)

14.4-algoritmissa tarvitaan vapaata levytilaa 1 K jokaista koodattavaa sekuntia kohti. 28.8 -algoritmi vaatii vastaavasti 1.8 K vapaata levytilaa sekuntia kohti. "LIVE" -koodaus edellyttää vähintään Pentium-tasoisen suorittimen ja enemmän keskusmuistia. (Holm 1998.)

MIDI - Musical Instrument Digital Interface (.mid)

MIDI on lyhenne sanoista Musical Instrument Digital Interface, joka tarkoittaa musiikkisoittimien digitaalista liitäntää tai rajapintaa, joka on suunniteltu välittämään viestejä sähköisten musiikkilaitteiden välillä. MIDI on standardoitu vuonna 1983 ja sitä ylläpitää MIDI Manufacturers Association (MMA). (Wikipedia 2015d.)

MIDI yleistyi nopeasti, koska sitä on voitu käyttää (hyödyntää) monenlaisissa tilanteissa. MIDI:n alkuvaiheilla se tosin tarkoitti lähinnä syntetisaattoreiden ja niiden ohjaamiseen käytettyjen sekvenssereiden yhdistämistä. Ajan myötä mukaan ovat tulleet myös muut soittimet, kuten kitara, viulu ja puhaltimet. MIDI mahdollistaa myös uudenlaisia musiikillisia ilmaisutapoja. MIDI-soittimena voi käyttää vaikka datakäsinettä, joka lukee käden liikkeitä ja muuttaa ne musiikilliseksi tiedoksi: nuoteiksi ja ohjainkäskyiksi. MIDI-laite voi tulkita digitaalisen lämpömittarin mittaamia lukemia ja muuttaa kappaleen tempoa mittarin mukaan. (Henrin 2004.)

MMC-koodin (MIDI Machine Control) avulla MIDI:ä voi käyttää vaikka valoshown tai lavatekniikan ohjauksessa. MIDI:n käyttö on yksinkertaista. Soittimesta on tavallisesti seuraavat liittimet: MIDI IN, MIDI OUT, ja usein myös MIDI THRU. MIDI IN vastaanottaa MIDI-viestejä ja MIDI OUT taas lähettää niitä. MIDI THRU lähettää viestejä mutta toisin kuin MIDI OUT, MIDI THRU ei muodosta viestejä, vaan se toistaa kaiken MIDI IN -portista tulevan. (Henrin 2004.)

Ogg (.ogg)

Ogg on äänen pakkausmuoto, jota Xiph.Org Fundation ylläpitää. Vuonna 1993 se alkoi yksinkertaisena äänen pakkausprojektina ja se tunnettiin silloin nimellä OggSquish mutta vuoden 1997 jälkeen se nimettiin Ogg:ksi. Toisin kuin muut tiedostotyypit tähän mennessä Ogg on ilmainen ja se käyttää Vorbis-pakkausjärjestelmää. Ogg Vorbis -tiedostojen äänenlaatu on erittäin hyvä, jopa parempi kuin MP3-tiedostojen, varsinkin pienillä bittinopeuksilla. Ogg Vorbis ei ole vielä kuitenkaan läheskään yhtä suosittu kuin MP3. Avoimen lähdekoodin ansiosta Ogg Vorbis ei ole sidottu pelkästään Windows-ympäristöön, vaan tuki löytyy myös esimerkiksi Linuxille ja Macintoshille. Ogg Vorbisin haittana voidaan pitää sitä, että pakkauksen purku vaatii MP3-pakkausmenetelmään verrattuna enemmän suoritintehoa. Täten esimerkiksi Ogg Vorbis -tiedostoja käyttävien kannettavien laitteiden toistoaika on MP3-tiedostoja käyttäviä laitteita lyhyempi. (Hertell 2005; Wikipedia 2015e.)

3 Beat Detection / Onset Detection

Internetistä löytyy paljon erilaisia oppaita ja valmiita koodeja eri kielille kuten C++, C#, Java, Python, Processing Language.Unity asset storesta löytyy maksullisia rytmihavaintokoodeja.

“Onset” viittaa musiikin alkunuottiin tai ääneen, jonka amplitudi muuttuu nollasta tämän nuotin/äänen alustavaan huippuun. ”Onset” liittyy transient-konseptiin: kaikilla nuoteilla voi olla ”onset”, mutta niillä ei välttämättä ole alustavaa tranientia. Transient tarkoittaa korkeata amplituudia, lyhytaikaista ääntä ääniaallon alussa (Wikipedia 2015f; Wikipedia 2015g).

Rytmin havainto yksinkertaisimmillaan on kahden matemaattisen kaavan käyttöä. Ensin lasketaan instant energy 'e' seuraavalla kaavalla. (Patin 2013):

$$e = e_{stereo} = e_{right} + e_{left} = \sum_{k=i_0}^{i_0+1024} a[k]^2 + b[k]^2$$

Sen jälkeen lasketaan local avarage energy '<E>' seuraavalla kaavalla:

$$<E> = \frac{1024}{44100} \times \sum_{i=0}^{44032} (B[0][i])^2 + (B[1][i])^2$$

Seuraavaksi verrataan, onko 'e' suurempi kuin 'C * <E>' , missä C on kaavion herkkyys, ja C:n pitää olla väliltä 1-2 esim. 1,3. Jos 'e' on suurempi kuin 'C * <E>' on löydetty rytmi. Tämä on alkeellinen kaava, jota kannattaa käyttää vain silloin kun yksinkertaiselle kaavalle on tarve. Edellä mainitussa kaavassa täytyy kaavion herkkyys laitta itse käsin, joka aiheuttaa rytmin havainnon tarkkuuden radikaalia vaihtelua . Tätä voidaan parantaa muilla kaavioilla. (Patin 2003.)

3.1 Fast Fourier Transform(FFT)

FFT on algoritmi, jolla voi laskea DFT:n eli Discrete Fourier Transformin nopeammin. Tavallinen DFT-laskenta on nopeudeltaan $2n^2$, kun taas FFT on nopeudeltaan 2nlog(n). FFT julkaistiin vuonna 1965 ja tämän algoritmin julkaisijat olivat Cooley ja Tukey. Fourier analyysi konvertoi ajan etäisyyden taajuudeksi eli aaltoluvuiksi, ja päinvastoin. Matemaattinen kaava, jolla DFT analysoi, on alla olevan mukainen. Ensimmäinen kaavio on ajan konvertointi taajuudeksi ja sen alla oleva kaavio on taajuuden muuttaminen ajaksi. (Wikipedia 2014; Smith 1998.)

$$F(\omega) = \int_{-\infty}^{\infty} f(t) e^{-i\omega t} dt$$

$$f(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} F(\omega) e^{i\omega t} d\omega$$

FFT-kaava (Proxymacentauri360 2012).

3.2 Käyttökohteet

Beat detectionia voidaan käyttää missä tahansa yhteydessä, jossa käytetään musiikkia. Esimerkiksi diskojen valaistuksessa voidaan valoja laittaa pois ja päälle musiikin tahdissa. Beat detectionia voidaan myös hyödyntää musiikin tunnistamisessa niin, että koodi ”kuuntelee” musiikin tahdin ja muita tietoja kuten vokaalit ja sen perusteella tarkistaa tietokannasta, löytyykö kappale näillä tiedoilla. Beat detectionia voidaan käyttää samalla lailla kuin Audiosurf- sekä Beat Hazard -peleissä. Audiosurf (2008) on musiikin pohjalta tehty peli, jossa pelaaja kontrolloi rakettia. Pelaaja kerää saman värisiä palikoita riviin tai jonoon saadakseen pisteitä sekä väistelee harmaita palikoita. Peli hyödyntää musiikkia palikoiden sijainnin sekä värin lisäykseen sekä pelin nopeuden vaihteluun. Beat Hazard (2010) on Asteroids- pelin tyylinen peli, jossa pelaaja tuhoaa vihollis-aluksia sekä asteroideja.

Pelissä hyödynnetään musiikkia vihollisten lisäykseen, pelaajan ammusten tehokkuuden ja visuaalisten tehosteiden muokkaamiseen.

3.3 C# -koodausympäristö

Koodausympäristönä projektissa käytin Visual Studio 2013 C# -sovellusta. Microsoft Visual Studio 2013 tarjoaa monia uusia ominaisuuksia tukemaan kehitystä uusimmille alustoille sekä moderneja työkaluja helpottamaan lisäarvon tuottamista. Visual Studio 2013 on paras ratkaisu nykyaikaisten sovellusten kehitykseen sekä sovelluksen elinkaaren hallintaan.

Välineellä voidaan tehdä esimerkiksi Windows-, web- ja mobiilisovelluksia ja siihen voidaan integroida monien eri valmistajien täydennyksiä, kuten esimerkiksi Intel Visual Fortran. Visual Studiolla graafisten käyttöliittymien luominen on erityisen helppoa, mikä selittää suurelta osin sen suuren suosion. (Wikipedia 2015i.)

4 BEATRUNNER-projekti

Työkaluina tässä projektissa käytin Unitya, Visual Studiota, LibreOfficea ja Mozilla Firefoxia.

Unity käyttöympäristönä oli tähän projektiin hyvä. Ainoana ongelmana oli se, että Unity ei ymmärrä MP3-tietotyyppiä, koska se on maksullinen tietotyyppi. Tästä johtuen jouduin käyttämään Ogg-tiedostotyyppiä. Muuten Unity on helppokäyttöinen ja muutenkin hyvä pelien tekoon. Unityn mukana tuleva MonoDevelopment on hyvä, yksinkertainen ja ilmainen ohjelmointiympäristö.
Unityllä voi testata hyvin pelin ominaisuuksia. Jos lataa Unity pluginin Visual Studioon niin saa Visual Studion oman testausympäristön käyttöön Unityn

testausympäristön lisäksi.

4.1 un4seen BASS.NET-kirjasto

BASS.NET on .Net-versio BASS:n audiokirjastosta ja sen jokaisesta lisäosasta. Sitä voi käyttää .Net-ympäristössä ja kaikilla .Net kielillä kuten C#, VB.Net, Jscript, F#. BASS.NET on maksullinen mikäli sillä tekee maksullisia tuotteita. Omassa projektissani käytin sivustoa bass.radio42.com/help/, joka sisältää alla kuvatut luokka-avaruudet, luokat ja funktiot. Tämä sivusto sisältää kaikkiaan 33 erilaista luokka-avaruutta.

Namespace:

Un4seen.Bass
Un4seen.Bass luokka-avaruus sisältää kaikki luokat, delegaatit ja enumeraatit bass.dll-kirjastosta. BASS on audiokirjasto Windows- ja Mac OSX- ohjelmistoihin. Sen tarkoituksena on tarjota kehittäjille tehokkaita työkaluja (funktioita) äänen hyödyntämiseen ohjelmistoissa.

Un4seen.Bass.AddOn.Fx
Tämä luokka-avaruus sisältää kaikki luokat, delegaatit ja enumeraatit bass_fx.dll kirjastosta. BASS_FX on Jobnikin tekemä laajennus, joka tarjoaa useita DSP-funktioita, kuten tempo- ja taajuusasetuksia. Kaikki BASS_FX-kutsut ovat pääluokan BassFx sisällä ja ne on toteutettu staattisina menetelminä.

Class:

BassNet
Rekisteröintiluokka, jota käytetään käyttäjän BASS.NET API -lisenssin rekisteröintiin.

BassFx
BASS.NET-kirjaston BassFx luokka käyttää BASS_FX.DLL sisältöä.

Bass

BASS.NET-kirjaston Bass luokka käyttää BASS.DLL sisältöä.

Funktiot:

BASS_Init

BASS_Init luo musiikin tuottolaitteen.

BASS_FX_BPM_BeatGetParameters

Nykyisen rytmin parametriarvon saamiseen käytettävä funktio.

Registration

BASS.NET-lisenssin rekisteröintifunktio.

BASS_StreamCreateFile

Funktio mahdollistaa MP3, MP2, MP1, OGG, WAV, AIFF tai mahdollisten lisäosien käyttämien tiedostojen sisältämän musiikin muuttamisen käyttökelpoiseksi dataksi.

BASS_FX_BPM_BeatDecodeGet

Käytetään rytmin sijainnin palauttamisen sekunnilleen kappaleen alusta.

BASS_FX_BPM_BeatFree

Resurssien vapauttamisfunktio.

Delegaatit/Enumeraatit:

BASSInit

Alustaa function “lippu” arvon.

BASSFlag

Lippujen luomiseen tarkoitettu enumeraatti.

BASSFXBpm

BPM/rytmin purkuasetukset.

BPMBEATPROC

Delegaatti, johon käyttäjä määrittelee function-nimen, jota kutsutaan kun rytmin sijainti on löydetty.

Esimerkkejä yllä kuvattujen osien käytöstä ja hyödyntämisestä:

```
using Un4seen.Bass;
using Un4seen.Bass.AddOn.Fx;
BassNet.Registration("tunnus", "rekisteröintikoodi");

Bass.BASS_Init(-1, 44100, BASSInit.BASS_DEVICE_DEFAULT, IntPtr.Zero);

//musiikin aloitus kohta
int stream = Bass.BASS_StreamCreateFile(“tiedoston_nimi”, 0L,

datan pituus        Decode the sample data, without outputting it
0L,                 BASSFlag.BASS_STREAM_DECODE);

//new BPMBEATPROC lähettää MyBeatProc funktiolle tiedot
BPMBEATPROC _beatProc = new BPMBEATPROC(MyBeatProc);

//funktio                              rytmin sijainti sekunnilleen
void MyBeatProc(int channel, double beatpos,                IntPtr user)

//decodaamisen aloitus sekunnilleen
BassFx.BASS_FX_BPM_BeatDecodeGet(stream, 0.0,

//decodaamisen lopetus sekuntti   työn teko piilossa          audio.clip.length,
BASSFXBpm.BASS_FX_BPM_BKGRND,

//funktio rytmin sijainnin löytämiseen
_beatProc,                IntPtr.Zero);

//vapauta resurssit
BassFx.BASS_FX_BPM_BeatFree(stream);
```

Lisätietoa BASS.NET-koodeista löytyy osoitteesta: http://bass.radio42.com/help/

4.2 Prototyypin toteutus ja testaus

Rytmihavainnon toteutus

Rytmihavainto tapahtuu kutsumalla BASS.NET-kirjaston funktiota BassFx.BASS_FX_BPM_BeatDecodeGet. Tämän sisällä kutsutaan MyBeatProc-funktiota, jolle lähetetään rytmin sijainti sekunnilleen. MyBeatProc- funktiossa tehdään esteen, alustan ja kuilun lisäys.

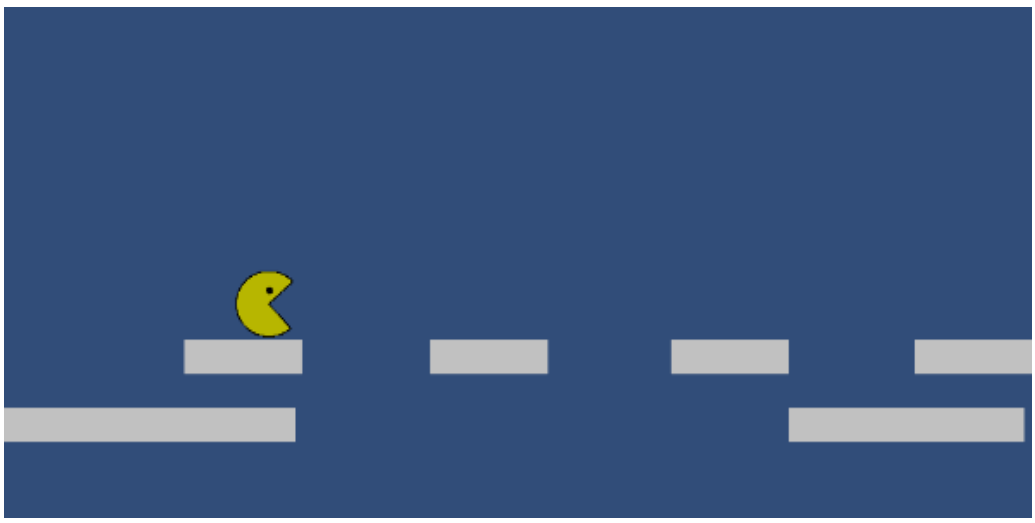
Prototyypissäni tasojen lisääminen tapahtuu, kun rytmihavainto löytää rytmin sijainnin. Jos rytmin löytämisen aikana kehitetty satunnaisnumero on alle 40, lisätään palikka esteeksi laittamalla palikka pelaajan korkeudelle. Mikäli numero on 40:n ja 60:n välillä, lisätään kuilu. Mikäli kuiluja tulee kolme peräkkäin, pakotetaan tason lisäys satunnaisluvusta huolimatta. Jos luku on yli 60, palikka jatkaa alustaa, jonka päällä pelaaja juoksee. Palikan sijainti x-akselilla määräytyy rytmihavainnon sijainnin perusteella. Palikan korkeus sekä leveys määritellään palikan elementissä (prefab). Elementti sisältää objektin tarvittavat tiedot kuten sijainnin, koon sekä materiaalin. Jokaisen esteen lisäyksen aikana tarkastetaan, kuinka monta samaa estettä on tehty, jotta voidaan varmistaa, ettei tule liikaa kuiluja tai estepalkkeja, joiden takia pelistä tulisi mahdoton tai tylsä pelata.

Haasteet

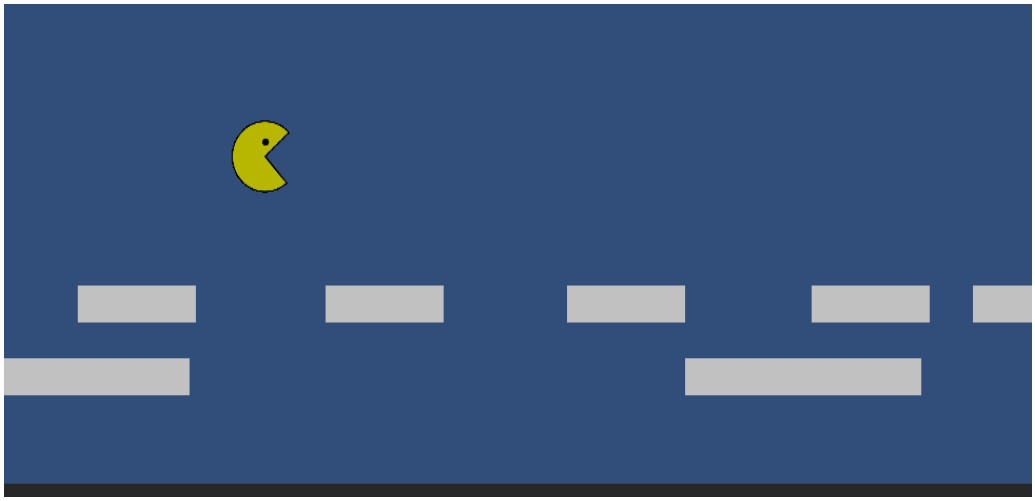
Prototyypin haasteina olivat rytmihavainnon kehittäminen, koska se oli minulle kokonaan uusi asia, sekä helposti ymmärrettävän materiaalin löytäminen. Omassa rytmihavaintokoodissa ongelmana oli sen hitaus, koska piti odottaa kunnes rytmihavaintokoodi oli käynyt koko kappaleen läpi. Kirjaston käyttö oli helppoa, koska se oli kehitetty C#-ympäristöön mutta sen rytmihavainnon tarkkuuden kanssa oli ongelmia. Kirjaston oppimiseen meni aikaa. Projektin ongelmien sekä oman osaamiseni takia aikataulu ei pitänyt.

Testaus

Testauksessa käytin klassista musiikkia, hevy metallia ja rokkia. Testauksen aikana huomasin, että käyttämäni kirjasto löytää paremmin rytmin hevy metal -musiikista kuin muista testaamistani musiikin lajeista. Kuten Mario Zechner (2010) kirjoittaa blogissaan, klassisesta musiikista on todella vaikea löytää rytmihavaintokoodille oikea rytmi. Koska toisin kuin muissa musiikeissa, klasissessa musiikissa rytmi vaihtelee laajasti, jopa huomaamattomasti, mikä tekee siitä todella vaikean havaita (GraemeG 2010). Klassisessa musiikissa kenttä muodostui enimmäkseen pitkistä kuiluista, esteistä tai alustasta, mikä teki pelistä tylsän tai jopa mahdottoman pelata. Tämä voi johtua siitä, että BASS.NET-kirjasto löytää rytmin helpommin bassoa sisältävästä musiikista, jossa on paljon ”meteliä”. Koska rytmihavainnon täytyy tietää herkkyys, jonka perusteella rytmi tarkistetaan, niin uskoisin että tässä kirjastossa on käytetty matalaa herkkyys-astetta, minkä takia rytmin löytäminen on helpompaa heavy metal -musiikista kuin klassisesta tai rokista. Herkkyysaste rytmihavainnossa tarkoittaa sitä, kuinka helppoa rytmin löytäminen musiikista on. Esimerkiksi rap-musiikista rytmin pystyy kuulemaan, kun taas heavy metal -musiikista sitä on vaikeampi kuulla.Testauksessa käytin Unityä sekä Visual Studion testausympäristöä. Testaushahmona käytin Pacman-hahmoa. Kuvat 2, 3 ja 4 kuvaavat miten prototyyppi kehittää kentän rytmihavainnon mukaan.



Kuva 2. Hahmon liikkuminen sekä kentän dynaaminen kehitys.



Kuva 3. hahmon hyppy seuraavalle tasolle.

Kuva 4. Epäonnistunut hyppy.

Kuva 5 esittelee rytmihavainnon tuloksia sekä satunnaislukua, jota käytän ohjelmassani kentän luomiseen. Esimerkistä näkee, että rytmihavainto voi tapahtua usean kerran sekunnin sisällä. Tämä voi aiheuttaa ongelmia kentän luomisessa, kuten kuvassa 5 näemme. Tasoja on samassa aikakohdassa eri korkeudella. Satunnaislukugeneraattori tässä ohjelmassa toimii eri tahdissa rytmihavainnon kanssa. Tämä mahdollistaa myös sen, että tasoja syntyy limittäin.

```
file.WriteLine("Beat at: " + beatpos + " Rand: " + rand);
```

```
Beat at: 2.61514739229025 Rand: 56
Beat at: 3.13607709750567 Rand: 56
Beat at: 3.30537414965986 Rand: 26
Beat at: 3.81884353741497 Rand: 26
Beat at: 4.34755102040816 Rand: 26
Beat at: 4.85936507936508 Rand: 26
Beat at: 5.37453514739229 Rand: 26
Beat at: 5.54895691609977 Rand: 40
Beat at: 5.88902494331066 Rand: 40
Beat at: 6.06226757369615 Rand: 40
Beat at: 6.58625850340136 Rand: 40
Beat at: 7.09927437641723 Rand: 54
Beat at: 7.44560090702948 Rand: 54
Beat at: 7.61702947845805 Rand: 54
```

Kuva 5. Rytmihavainnon tulokset

5 Yhteenveto

Kokonaisuudessaan olen tyytyväinen opinnäytetyöprosessin lopputulokseen, vaikka alkuperäinen visio, tehdä toimiva peli musiikin pohjalta, jäi protoversion tasolle. Alkuperäinen tavoite opinnäytetyön valmistumiselle oli toukokuu 2015, mutta peliprojektin aikana tuli yllättäviä teknisiä ongelmia ja myös raportin kirjoittaminen osoittautui odotettua haastavammaksi.

Projektin alkuvaiheessa ongelmana oli rytmivaihdon nopeuden ja tarkkuuden käsittely. Oman rytmihavaintokoodini alku oli hyvä muokattavuudeltaan, mutta nopeudeltaan hidas. Kappale piti kuunnella kokonaan ennen kuin pelin pystyi aloittamaan. BASS.NET-rytmihavainto on nopeudeltaan todella hyvä mutta tarkkuus vaihtelee eri musiikkilajeilla suuresti ollen esimerkiksi klassisella huono mutta kohtuullisen hyvä heavy-musiikilla. Edellä mainitut kokemukset johtivat siihen, että tavoitteena ollut toimiva peli muuttui pelin prototyypiksi, jossa keskitytään kentän luomiseen rytmihavainnon mukaan. Aiheeseen liittyvää materiaalia oli niukasti saatavilla ja löytynyt materiaali oli englanninkielistä.

Projekti opetti paljon rytmihavainnosta sekä pelin tekemisestä. Uskon että tulen tekemään tämän pelin loppuun mutta omalla rytmihavaintokoodilla. Nykyinen

toteutus ei ole niinkään monipuolinen tai tarkka eri musiikin lajeissa. Myöhemmin opinnäytetyön aikana löysin Rythmtool-nimisen valmiin rytmihavaintokoodin Unity Asset -kaupasta hintaan 25€. Tätä suositeltiin paljon mutta itse en kuitenkaan kokeillut kyseistä koodia omassa toteutuksessani, joten en pysty sanomaan kuinka hyödyllinen se on.

Opinnäytetyön valmistumisen venyminen johtui projektin viivästymisestä. Projektin kannalta minun oli opiskeltava paljon kokonaan uusia asioita. Raportin kirjoittamiseen keskityin todenteolla vasta prototyypin luomisen jälkeen. Haasteina raportin teossa oli sopivien lähteiden löytäminen. Vaikka lähteitä löytyikin, niin tieto oli suppeaa. Raportissa katsoin parhaaksi käyttää myös lähteiden suoria lainauksi, koska suuri osa lähteistä sisälsi paljon tarkkaa teknistä tietoa, ja tekstin muuttaminen omin sanoin olisi saattanut menettää sisällön ymmärtämisen.

Opinnäytetyössä käytetyt rytmihavaintoon liittyvät lähteet ovat kaikki englannin kielellä. Työn toteutuksessa käytetty toiminnallinen projektimainen menetelmä soveltui hyvin tutkimuksen suorittamiseen. Vaikka opinnäytetyöllä ei ollut toimeksiantajaa, niin mielestäni opinnäytetyössä syntynyttä pelin prototyyppiä pystyy hyödyntämään pohjana rytminhavaintoa käyttävään sovellukseen huolimatta siitä, että lopputuloksena ei syntynytkään täydellistä peliä.

Oman oppimisen kannalta tämä projekti opetti paljon uutta tietoa äänen ja rytmihavainnon käytöstä peleissä. Unityn osalta opin paljon pelien luomisesta sekä UnityVS Visual Studio –lisäosasta, joka lisää mahdollisuuden testaukseen ja virheiden jäljittämiseen. Kirjallista osuutta kirjoittaessani opin, että raportin kirjoittamiseen pitää varata paljon aikaa. Raportissa ei voi ilmaista asioita lyhyesti ja ytimekkäästi, vaan perustellusti ja lähteitä käyttäen. Mielestäni opinnäytetyön oppimistavoitteet toteutuivat omalta kohdaltani.

Lähteet

Audiosurf. 2008. Dylan Fitterer.

Beat Hazard. 2010. Cold Beam Games.

Cycling 74. How Digital Audio Works.
http://cycling74.co/docs/max5/tutorials/msp-tut/mspdigitalaudio.html. 21.06.2015.

Digiwiki. 2011. Äänitteiden digitointi.
http://www.digiwiki.fi/fi/index.php?title=%C3%84%C3%A4nitteiden_digitointi. 30.06.2015

GraemeG. 2010. Rhytms used in Classical Music? Talk Classical.
http://www.talkclassical.com/8844-rhythms-used-classical-music.html. 01.12.2015

Henrin. 2004. Mikä on MIDI?.
http://www.henrin.net/musiikki/tekniikka/midi. 01.07.2015.

Hertell, J. 2005. Äänen streamaus.
https://publications.theseus.fi/bitstream/handle/10024/10600/TMP.objres.51.pdf?sequence=2. 01.02.2015.

Holm, J-M. 1998. Audioformaatit.
http://www.mit.jyu.fi/opiskelu/seminaarit/bak/audioformaatit/#2.%20%C3%84%C3%A4ni%20tietokoneessa. 15.06.2015.

Keskinen, A & Hoikkanen, A & Korkalo, O. 1999. MP3-tekniikka.
http://www.netlab.tkk.fi/opetus/s38118/s99/htyo/16/tekniikka.shtml. 20.06.2015.

Klapuri, A. 2013. Digitaalinen audio.
http://www.cs.tut.fi/~digaudio/PDF/johdanto.pdf. 01.05.2015.

Mentholorange. 2013. How Digital Audio Works (from Max/MSP basic tutorial).
http://orangelearnof.files.wordpress.com/2013/03/image015.gif. 20.06.2015.

Mäkelä, J. 2011. Äänitteiden digitointi.
http://www.digiwiki.fi/fi/index.php?title=%C3%84%C3%A4nitteiden_digitointi. 25.04.2015.

Ohjelmistot.com. 2015. Microsoft-Visual Studio 2015.
http://www.ohjelmistot.com/Comersus/store/comersus_viewProductFamil

y.asp?idproductfamily=255&gclid=Cj0KEQjwmNuuBRDTu5rDjr2kxJsBEiQAWlm6UrAy1JIf7wg7Bit-X8JfqcOQUdS9pOi_o9_VLkhfr-8aAtPs8P8HAQ. 10.08.2015.

Patin, F. 2003. Beat Detection Algorithm. http://www.gamedev.net/page/resources/_/technical/math-and-physics/beat-detection-algorithms-r1952. 05.10.2014

Proximacentauri360. 2012. Fourier Transform. https://proximacentauri360.wordpress.com/2012/08/15/fourier-transform/. 20.07.2015.

Radio42. 2015. Help Dokumentti. http://bass.radio42.com/help/. 01.08.2015.

Ruotsala, A. 2014. Digitaalinen ääni. http://emute.edu.fi/taustatieto/aihealueet/yleista-tietoa/3.-digitaalinen-aani. 01.06.2015.

Smith, S W. 1998. The Scientist and Engineer's Guide to Digital Signal Processing. http://www.dspguide.com/ch12/2.htm. 20.11.2014.

Uimonen, J. 2006. Äänen tallennus. http://www.music.helsinki.fi/tmt/opetus/aanitys/luento3/pruju3.html. 12.04.2015.

Vangie, B. 2005. Common Audio Formats. http://www.webopedia.com/DidYouKnow/Computer_Science/digital_audio_formats.asp. 17.05.2015.

Whitson, G. 2012. What’s the Difference Between All These Audio Formats, and Which One Should I Use?. http://lifehacker.com/5927052/whats-the-difference-between-all-these-audio-formats-and-which-one-should-i-use. 12.06.2015.

Wikipedia. 2014. Fast Fourier Transform. http://en.wikipedia.org/wiki/Fast_Fourier_transform. 11.11.2014.

Wikipedia. 2015a. Windows Media Audio https://en.wikipedia.org/wiki/Windows_Media_Audio. 01.02.2015

Wikipedia. 2015b. WAV https://en.wikipedia.org/wiki/WAV. 05.03.2015

Wikipedia. 2015c. RealAudio https://en.wikipedia.org/wiki/RealAudio. 07.03.2015

Wikipedia. 2015d. MIDI
https://en.wikipedia.org/wiki/MIDI. 15.05.2015

Wikipedia. 2015e. Ogg
https://en.wikipedia.org/wiki/Ogg. 20.04.2015

Wikipedia. 2015f. Onset (audio).
https://en.wikipedia.org/wiki/Onset_%28audio%29. 25.01.2015.

Wikipedia. 2015g. Transient (acoustics)
https://en.wikipedia.org/wiki/Transient_%28acoustics%29. 22.04.2015

Wikipedia. 2015h. Digital Audio.
https://en.wikipedia.org/wiki/Digital_audio. 04.01.2015.

Wikipedia. 2015i. Microsoft Visual Studio.
https://fi.wikipedia.org/wiki/Microsoft_Visual_Studio

Zechner, M. 2010. Onset Detection Part 4.5: What to expect. Badlogic Games.
http://www.badlogicgames.com/wordpress/?cat=18. 01.12.2015