TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond
Tarkvarateaduse instituut

Tanel Dunaiski 142849 IAPB

GENERATIIVNE MUUSIKA UNITY 3D ABIL

bakalaureusetöö

Juhendaja: Jaagup Irve
Tehnikateaduste magister

Tallinn 2017

Autorideklaratsioon

Kinnitan, et olen koostanud antud lõputöö iseseisvalt ning seda ei ole kellegi teise poolt varem kaitsmisele esitatud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on töös viidatud.

Autor: Tanel Dunaiski

22.05.2017

Annotatsioon

Bakalaurusetöö eesmärgiks on luua rakendus, mis loob generatiivselt mängu meeleolule vastavat muusikat.

Töös käsitletakse erinevaid muusika loomise viise mängudele ning tutvustatakse põhilisi generatiivse muusika loomiseks kasutatavaid algoritme. Lisaks räägitakse ka nende algoritmide realiseerimiseks vajaminevast muusikateooriast.

Töö praktilise osana valmis generatiivset muusikat loov rakendus, mille arendamiseks kasutati Unity 3D-d ning C#-i. Rakenduse poolt loodav muusika muutub ka vastavalt mängu meeleolule – kasutajal on võimalik lisada erinevatele aladele skript, mille alusel programm muudab loodava muusika helistikku, tempot ning kas suurendab või vähendab instrumentide arvu.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 30 leheküljel, 7 peatükki, 11 joonist.

Abstract

Generative music using Unity 3D

The purpose of this thesis is to create a program that generates music in real time and which also has the property of changing generated music in relation to the game state.

This document addresses different ways of creating music for games in general and introduces different algorithms used for creating generative music. Also basic music theory that is needed for using these algorithms correctly is discussed.

As a result of this work, a generative music application was created. Unity 3D and C# were used for developing this program. Music created by this application also responds to the changes in game state – user has the possibility of adding certain scripts to different areas, based on which program can change the generated music's key, tempo and add or take away different instruments.

The thesis is in Estonian and contains 30 pages of text, 7 chapters, 11 figures.

Sisukord

1 Sissejuhatus....8
2 Teoreetiline taust....9
2.1 Muusika mängudes....9
2.1.1 Tsükliline muusika....9
2.1.2 Adaptiivne muusika....9
2.2 Generatiivne muusika....10
2.2.1 Deterministlikud ja stohhastilised protsessid....10
2.2.2 Stohhastilised protsessid....11
2.3 Muusika teoreetiline taust....13
2.3.1 Helistik....13
2.3.2 Akord ja kolmkõla....14
2.3.3 Rütm....14
3 Kasutatud tehnoloogiad....16
3.1 Unity 3D....16
3.2 Microsoft Visual Studio....16
3.3 C#....17
4 Rakenduse ülesehitus....18
4.1 Peamised kasutuses olevad Unity komponendid....18
4.1.1 Scene....18
4.1.2 GameObject....18
4.1.3 Collider....19
4.2 Olulisemad loodud komponendid....20
4.2.1 Skriptimine....20
4.2.2 MusicGenerator....20
4.2.3 Melody....21
4.2.4 Percussion....21
4.2.5 ZoneProperties....21
4.2.6 ZoneChange....21

4.2.7 Rakenduse struktuur....21
5 Rakenduse kirjeldus....23
6 Hinnang tulemusele....27
7 Kokkuvõte....28
Kasutatud kirjandus....29

Jooniste loetelu

Joonis 1. Näide: Kalduvuse maskid [9].....11
Joonis 2. Näide: Tõenäosuse muutumine [9].....12
Joonis 3. Näide: Markovi ahel tabeli ning graafi kujul [9].....13
Joonis 4. Näide: Helipikkuste jagunemine takti [2].....15
Joonis 5. Tühi stseen.....18
Joonis 6. Mänguobjekti ülesehitus.....19
Joonis 7. Trigger tüüpi kastikujuline collider.....20
Joonis 8. Rakenduse struktuur.....22
Joonis 9. Kolmkõla nootide otsimine helifailidest.....24
Joonis 10. Noodi pikkuse genereerimine vahemikule 4-7 arvestades tempot.....25
Joonis 11. Järgmise noodi leidmine kasutades Brown noise-i ideed.....26

1 Sissejuhatus

Töö eesmärgiks on luua Unity 3D abil ja C# keelt kasutades programm, mis loob generatiivselt mängu meeleolule vastavat muusikat. Olemas on küll programmid, mis Unity 3D-le generatiivset muusikat loovad, kuid neil puudub mängu situatsioonile vastavalt generatiivsuse muutmise omadus.

Kasutajal on võimalik valida endale sobiv helistik, tempo ja kasutatavad instrumendid. Seejärel programm valib välja standardjärgnevuse bassikäiguks ning hakkab selle peale looma meloodiat. Meloodia luuakse bassi noodist ülesehitatud kolmkõla nootidest. Kui mängu olukord muutub pingelisemaks, siis programm tõstab tempot ning lisanduvad uued instrumendid, nagu näiteks löökpillid. Mängu meeleolu muutmiseks peab kasutaja lisama Unity 3D-s *collider*-i ala ning sellele lisama skripti, millega saab alale valida omadused. Kui mitu ala on omavahel kattuvad, siis helistikuks valitakse sisenetud ala helistik ning tempoks nende kahe aritmeetiline keskmine.

Bakalaurusetöö peatükkides kaks ja kolm tutvustatakse koostatud rakenduse teoreetilist tausta. Räägitakse generatiivse muusika loomise erinevatest tehnikatest ning tuuakse välja üldine muusikateooria põhi.

Bakalaurusetöö peatükkides neli ja viis räägitakse konkreetse loodud rakenduse erinevatest osadest ning selle töökäigust.

2 Teoreetiline taust

2.1 Muusika mängudes

Üheks kõige populaarsemaks ning efektiivsemaks emotsioonide muutjaks, ükskõik millises meediapõhises loo jutustamises, on muusika. Muusikal on võime mõjutada inimese psüühikat ilma, et inimene ise sellest aru saaks. Samasugune on muusika võime ka mängudes – see aitab mängijal sisse elada ja mängu rohkem nautida. Üheks suureks probleemiks mängude jaoks muusika loomisel on fakt, et traditsiooniliselt on muusika lineaarne kunstivorm – sellel on kindel algus, keskosa ja lõpp. Tänapäevastel mängudel seevastu on kalduvus olla mittelineaarne ning ettearvamatu [17].

2.1.1 Tsükliline muusika

Mitmeid aastaid oli selle probleemi kõige tüüpilisemaks lahenduseks nn tsükliline muusika, kus iga muusika tükk oli kirjutatud nii, et selle lõpp läheb sujuvalt üle sama tüki algusesse. See võimaldab mängida muusikat tsükliliselt ning täpselt nii kaua kui vaja. Seda tehnikat kasutati näiteks kõige esimestes mängudes, mis sisaldasid muusikat (nt “Space Invaders”) ning kasutatakse veel ka tänapäeval [17].

2.1.2 Adaptiivne muusika

Muusika mängudes ei peaks aga mitte ainult korduma vaid ka muutuma vastavalt mängu meeleolule. Sellist muusikat nimetatakse adaptiivseks või dünaamiliseks muusikaks. Kaks kõige tüüpilisemat lähenemist selle teostamiseks on vertikaalne taasorkestratsioon (ingl. *vertical re-orchestration*) või horisontaalne taasjärjestamine (ingl. *horizontal re-sequencing*). Vertikaalne taasorkestratsioon kasutab põhjana muusikat, mis on võimalikult neutraalne ja lisab või võtab ära emotsiooni muutmiseks instrumente, harmonisatsioone ning rütmilisi elemente. Horisontaalse taasjärjestamise jaoks kirjutatakse selline muusika, mis võib hüpata ükskõik millisesse muusika osasse

ükskõik mis ajal ja mis järjestuses ilma, et mängija sellest aru saaks. Lähenemine, millel on aga kõige suurem potentsiaal paindlikkusele ja adaptiivsusele, on muusika, mis koostab ennast ise reaalajas [17].

Näiteks on enamikud rollimängu tüüpi arvutimängud sellised, kus kasutaja veedab mitmeid tunde looduses ringi rännates, kuulates vaikset taustamuusikat. Kui aga tekib vastane, muutub mängu muusika pingelisemaks olenemata vastase tüübist ning sellest, kas mängija võitleb temaga või on lihtsalt läheduses. Paljudes mängudes on muusika liiga must-valge, mistõttu ei anna mängu hetkesituatsiooni kohta piisavalt täpset informatsiooni. Siinkohal tulevadki hästi välja adaptiivsete süsteemide eelised. Näiteks võiks muusika muutuda pingelisemaks alles siis, kui mängija ja vastane konkreetselt omavahel võitlevad ning muusika erineva tugevusega vastaste vahel võiks samuti olla erinev [3].

2.2 Generatiivne muusika

Generatiivne muusika võimaldab kirjutada ja mängida muusikat reaalajas mingite etteantud parameetrite ja piirangute põhjal. Kuigi seda kasutatakse vähem kui eelnevalt salvestatud muusikat, on sellel väga suur potentsiaal tulevikus just tänu paindlikkusele ja adaptiivsusele. Kuna generatiivse muusika loomisel on vaja teha väga palju valikuid, on selle jaoks tehtud palju keerukaid algoritme, mistõttu kutsutakse seda stiili ka algoritmiliseks muusikaks. Lisaks algoritmidele on sellise muusika loomiseks vajalik ka ettemääratud elementaarne muusikaline materjal, mis sisaldab erinevaid helistikke, noodipikkusi, akordide järgnevusi ning helifaile, mida mängida. Generatiivse süsteemi musikaalsus sõltub paljuski algoritmi tüübist, mida kasutatakse, kuid ka ettemääratud muusikalisest materjalist [10].

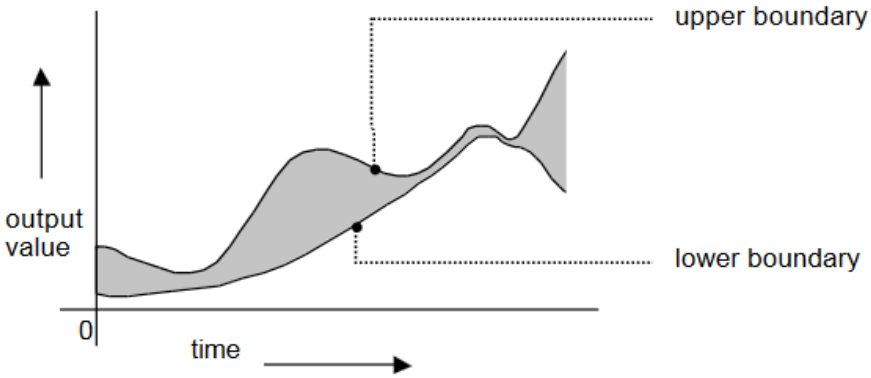
2.2.1 Deterministlikud ja stohhastilised protsessid

Arvutil on kaks vastandlikku viisi otsuste tegemiseks: deterministlik või stohhastiline. Deterministlik protsess jõuab lahenduseni viies läbi kindlaksmääratud, kuid mõnikord keerulisi ülesandeid, mis ei sisalda juhuslikku valikut. Stohhastilised protsessid aga integreerivad juhuslikkuse oma otsuste tegemisse. Mõlemat tehnikat võib kasutada generatiivse muusika loomiseks, kuid neil on erinevad eesmärgid. Antud töö raames on

keskendutud stohhastilistele algoritmidele, kuna nendega on suurem võimalus variatsiooniks ja ettearvamatuseks. Stohhastilise meetodi kasutamine vähendab ka andmete kogust, kuna võrreldes deterministlikke protsessidega, on stohhastilistes meetodites vaja vähem helifaile sama mitmekesise muusika loomiseks. Kuid sellel on ka oma negatiivne pool – kuna stohhastilistes meetodites kasutatakse palju juhuslikku valikut, võib selle loodud muusika anda vähem ühtse multimeedia kogemuse [10].

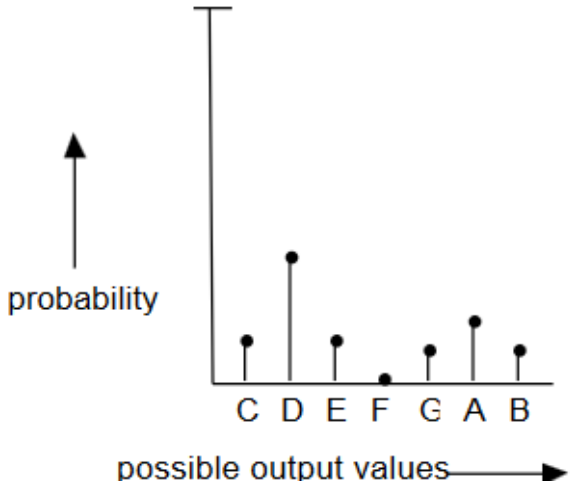
2.2.2 Stohhastilised protsessid

Kõige lihtsamad stohhastilised tehnikad on aleatoorsed protsessid. Aleatoorne protsess genereerib juhusliku numbri etteantud maksimaalse ja minimaalse väärtuse vahele. Muutes neid piire on võimalik luua nn kalduvuse maskid (ingl. *tendency masks*)(vt joonis 1), mida kasutades on võimalik kontrollida näiteks harmoonilise järgnevuse noote või löökpillide dünaamilist intensiivsust. Seda tehnikat on laialdaselt kasutatud granulaarse helisünteesi tehnikates ning sobib hästi ka generatiivse muusika loomiseks.



Joonis 1. Näide: Kalduvuse maskid [9].

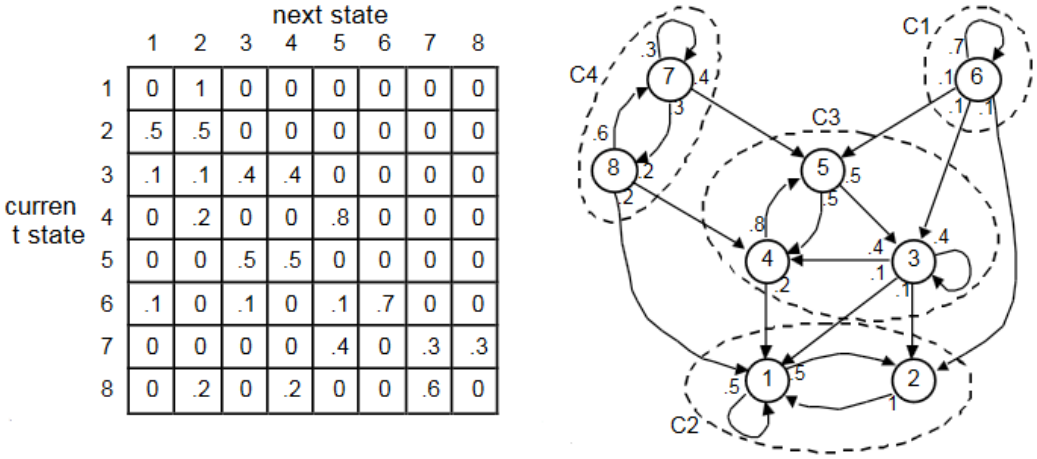
Lisaks aleatoorsele protsessile, kus igal väärtusel on samasugune esinemistõenäosus, kasutatakse tihti veel ka tõenäosuse jaotamist. Sellega saab teha kindlaks, et mingisugune väärtus esineb teistest rohkem ning selle abil on võimalik koostada näiteks fundamentaalne toon. See on väga kasulik ka interaktiivses keskkonnas, kus kasutaja tegevused võivad mõjutada sündmuse juhtumise tõenäosust. Joonis 2 demonstreerib tõenäosuse muutumist.



Joonis 2. Näide: Tõenäosuse muutumine [9].

Tüüpiline juhuslikkuse vorm on veel *Brown noise*, mida kutsutakse mõnikord ka juhuslikuks kõnnakuks. See erineb eelnevalt mainitud meetoditest selle poolest, et tema väljund sõltub eelnevast väljundist. Tavaliselt on järgmine väljund kas sama, ühe võrra suurem või väiksem, mis põhjustab järk-järgult muutuva tulemuse.

Üks keerulisemaid stohhastilisi meetodeid on Markovi ahel. Ka sellel on järgmine väljund sõltuv eelnevast, kuid erinevalt *Brown noise*-ist, on järgneva väljundi tõenäosus sõltuv väljunditest, mis on lähiminevikus tulnud samast seisundist. Hetkeseisundile vastavale tõenäosuste reale toetudes valitakse järgmine seisund, järgnevale seisundile vastavale tõenäosuse reale toetudes järgmine seisund jne – sellest tulenebki antud algoritmi ahela aspekt (vt joonis 3). Esimese tasandi Markovi ahel vaatab tagasi ainult ühe astme võrra, teise tasandi ahel vaatab tagasi kahe astme võrra jne. Mida kõrgema tasandi Markovi ahelaga on tegemist, seda sidusamad on loodavad mustrid, kuid need muutuvad ka väga etteaimatavaks ning nõuavad töötlemiseks palju ressursse. Markovi ahelaid saab generatiivse muusika loomisel kasutada väga erinevatel juhtudel. Muutes ahela tasandit on võimalik luua kas väga juhuslikke järgnevusi või ennustatavaid perioodilisi meloodiaid [10].



Joonis 3. Näide: Markovi ahel tabeli ning graafi kujul [9].

Antud töös on kasutatud tõenäosuse jaotamist ning ka lihtsustatud versiooni *Brown noise*-ist. Tõenäosuse jaotamine on kasutusel noodipikkuste valimisel ning *Brown noise*-i ideed kasutatakse mängitava noodi valimiseks, kus see peab olema kas samas, eelmises või järgmises oktaavis.

2.3 Muusika teoreetiline taust

2.3.1 Helistik

„Helistik on kindla algusnoodiga helilaad.” [13, lk 204] Muusikas mõõdetakse erinevate kõrgustega helide vahelisi kauguseid toonides. Teineteise kõrval olevate astmete vahel võib olla kas pooltooni või tervetoon. Pooltoonid on muusikas III ja IV astme ning VII ja I astme vahel. Tervetoon on kõikide ülejäänud astmete vahel. Samasugust helirida on võimalik moodustada kõikidelt erinevatelt helikõrgustelt ning valitud helikõrguselt ülesehitatud helirida nimetataksegi helistikuks. Helistiku I aste kannab nime toonika, mis on helistiku kõige tähtsamaks astmeks. Üldjuhul algavad meloodiad toonikast ning ka lõppevad sinna [11].

Helistiku astmed jagunevad püsivateks ning püsimatuteks. Püsivad astmed on I III ja V ning ebapüsivad II IV VI ja VII aste. Püsimatud astmed lahenevad üldjuhul püsivatesse naaberastmesse välja arvatud VII aste korral, mis laheneb tõusvalt esimesse astmesse

[7]. Püsivus on meloodias oluline, sest annab loole helistikutunnetuse ning nendesse lahendatakse paljud akordid.

Modulatsiooniks nimetatakse ühe helitöö jooksul helistiku vahetust. Helindi uuest helistikust on võimalik aru saada nii meloodilise(lineaarse) kui ka akordilise(vertikaalse) muutuse järgi. Võrreldes eelneva helistikuga saavad helikõrgused pärast modulatsiooni uued heliastmenimetused ning nendest moodustunud akordid omandavad uue funktsiooni [6].

2.3.2 Akord ja kolmkõla

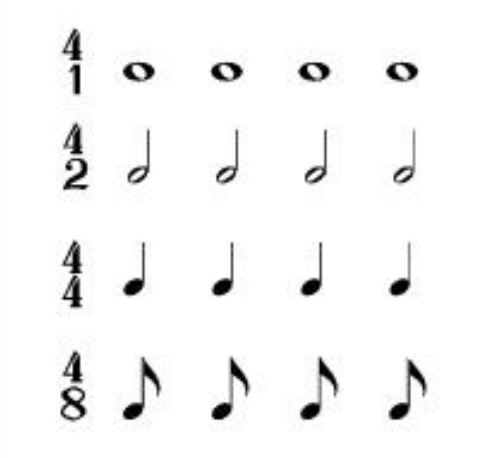
„Akordiks nimetatakse kooskõla vähemalt kolmest helikõrgusest, mida saab omavahel asetada tertside vahekorda." [6, lk 119] Kõige lihtsamaks akordi tüübiks on kolmkõla, mida saab, nagu teisigi akorde, ehitada helistiku igalt astmelt. Kõla poolest võib kõiki kolmkõlasid liigitada kahte suurde rühma: duur- ja mollkolmkõlad. Iga kindla helilaadi astmelt üles ehitatud kolmkõla omab alati kindlat funktsiooni kõigepealt helilaadi põhitooni ja selle kolmkõlaga, kuid ka ülejäänud astmetega ning nendest moodustatud akordidega [6].

Kolmkõlad jaotatakse kolme põhifunktsiooni: toonika-, dominant- ja subdominantfunktsiooni. Toonika on I astme funktsioon ning sellelt ehitatud kolmkõla, toonikakolmkõla, on helistikus kõige püsivam akord. Toonikakolmkõla moodustavad helistiku I III ja V aste, mis on ka kõik püsivad astmed. Toonika kõlab duuris ja mollis erinevalt. Dominant on V astme funktsioon. Sellelt astmelt ehitatud kolmkõla on püsimatu ning kõlab ühtemoodi nii duuris kui mollis. Dominantkolmkõla koosneb V VII ja II astmest. Subdominant on IV astme funktsioon ning koosneb astmetest IV VI ja I. Nii nagu toonika, kõlab subdominantkolmkõla duuris ja mollis erinevalt ning on dominantkolmkõlast iseloomu poolest rahulikum [7].

2.3.3 Rütm

Rütm on helide paigutamine ajas. See on muusika üks olulisemaid osasid, milleta meloodia ei saaks eksisteerida. Erinevatest osadest koosnevatel nootidel on erinev helipikkus. Helipikkused jagunevad järgmiselt: täisnoot, poolnoot ehk kahendik, veerandnoot ehk neljandik ning kaheksandiknoot. Igal heliteosel on määratud ka

taktimõõt, mis näitab, mitu ja millist helipikkust mahub ühte takti. Joonise 4 esimesel real on näidatud, kuidas ühe takti sisse mahub neli täisnooti. Joonise teine rida näitab, kuidas ühte takti mahub neli poolnooti jne [2].



Joonis 4. Näide: Helipikkuste jagunemine takti [2].

3 Kasutatud tehnoloogiad

3.1 Unity 3D

Unity 3D on Unity Tehnologies poolt arendatud mängu mootor, millega on võimalik luua videomänge nii arvutitele, konsoolidele, mobiilidele kui ka veebi. Unity 3D sisaldab redigeerimisprogrammi, millega on võimalik luua ning disainida sisu, ja mängu mootorit, millega saab loodud produkti käivitada. See on tuntud just oma võime poolest luua lihtsalt mänge kõikidele populaarsetele mänguplatvormidele.

Unity 3D on võitnud mitmeid auhindu, nagu näiteks Wall Street Journali 2010. aasta tehnoloogia innovatsiooni auhinna ning sellega on loodud mitmeid populaarseid videomänge. Tuntumateks mängudeks on näiteks Blizzardi poolt arendatud Hearthstone ning 2011. aastal firma Squad poolt loodud mäng Kerbal Space Program [16].

Antud töö arendamiseks sai valitud Unity 3D, kuna see võimaldab lihtsalt kontrollida loodud muusika dünaamilisust ning annab võimaluse ka järgnevatel arendajatel integreerida valminud skripte oma loomingutesse. Eeliseks on ka Unity 3D arenduskeskkonna tasuta kättesaadavus ning Pro litsentsi ei ole vaja osta kuni arendatud mängu käive ei ole üle saja tuhande dollari aastas [8].

3.2 Microsoft Visual Studio

Antud töös kasutatavad skriptid on kirjutatud Microsoft Visual Studio IDE-s. Tegemist on Microsofti poolt arendatud programmiga, millega saab luua videomänge ning erinevaid veebi- ja mobiilirakendusi. Microsoft Visual Studio toetab C, C++, C++/CLI, VB.NET, C#, F# ja TypeScript keeli ning lisateenuseid paigaldades on võimalik toetama ka veel mitmeid teisi. Microsoft pakub kõigile programmist tasuta versiooni, mis toetab ka kõiki lisasid [9].

3.3 C#

Unity toetab programmeerimist kolmes keeles: Javascript, C# ja Boo [8]. Antud töö kõik skriptid on kirjutatud C#-is, kuna see pakub antud valikust kõige rohkem võimalusi ning oli autorile kõige arusaadavam.

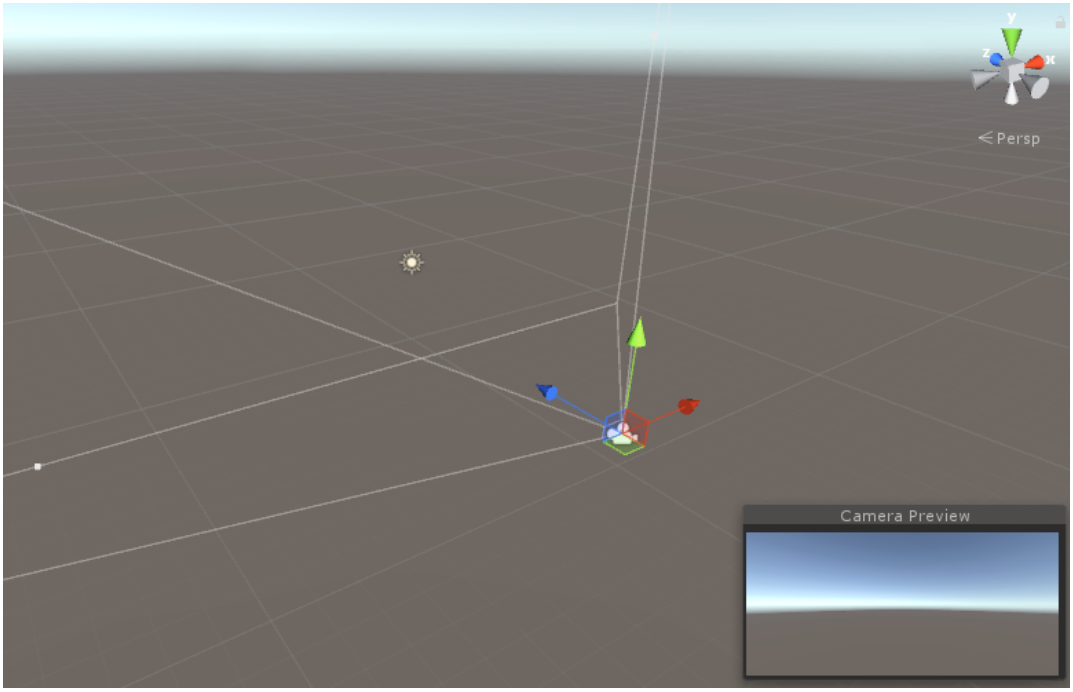
C# on tüübikindel (ingl. *type-safe)* objektorienteeritud programmeerimiskeel, mis võimaldab arendajatel luua erinevaid turvalisi ning robustseid .NET raamistikul jooksvaid rakendusi. Kuna tegemist on objektorienteeritud keelega, toetab C# ka kapseldamist (ingl. *encapsulation*), pärimist (ingl. *inheritance*) ja polümorfismi (ingl. *polymorphism*). Kõik muutujad ja meetodid on kapseldatud klassi definitsiooni sisse. Klass võib otseselt pärida ainult ühelt klassilt, kuid võib implementeerida mitmeid liideseid [5].

4 Rakenduse ülesehitus

4.1 Peamised kasutuses olevad Unity komponendid

4.1.1 Scene

Stseenid (ingl. *scenes*) sisaldavad Unity-s kõiki mängu objekte. Neid saab kasutada nii menüü, erinevate tasemete kui ka kõige muu loomiseks. Igal stseenil peab olema vähemalt üks kaamera ja üks valgustuse objekt, nagu on näha joonisel 5 [12].

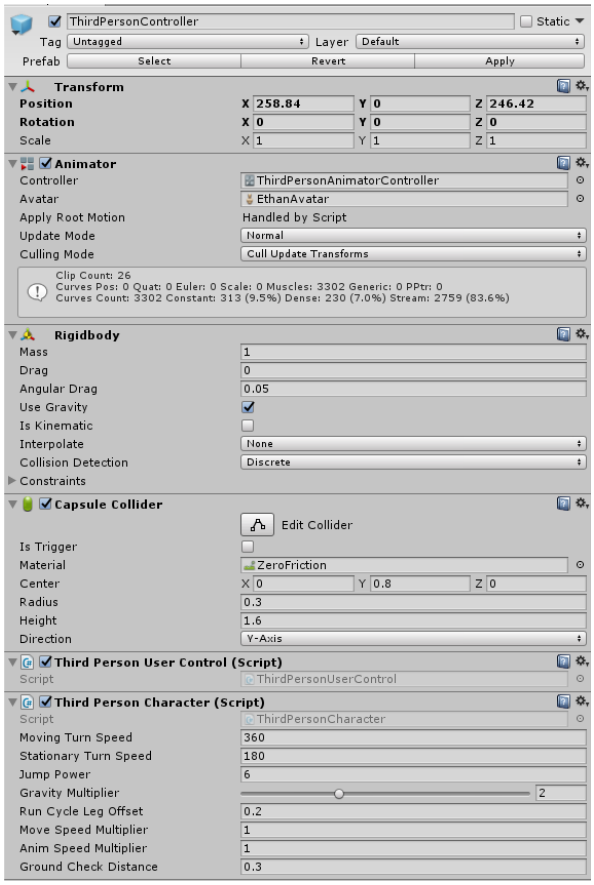


Joonis 5. Tühi stseen.

4.1.2 GameObject

Mänguobjekt (ingl. *GameObject*) on kõige olulisem Unity komponent. Iga objekt, mis on Unity-ga loodud mängus, on mänguobjekti tüüpi. See töötab konteinerina teistele komponentidele ja iseseisvalt ei tee midagi, vaid vajab teiste objektide omadusi ning

efekte. Ühele mänguobjekti konteinerile saab lisada mitu teist objekti, nagu näha joonisel 6 [4].



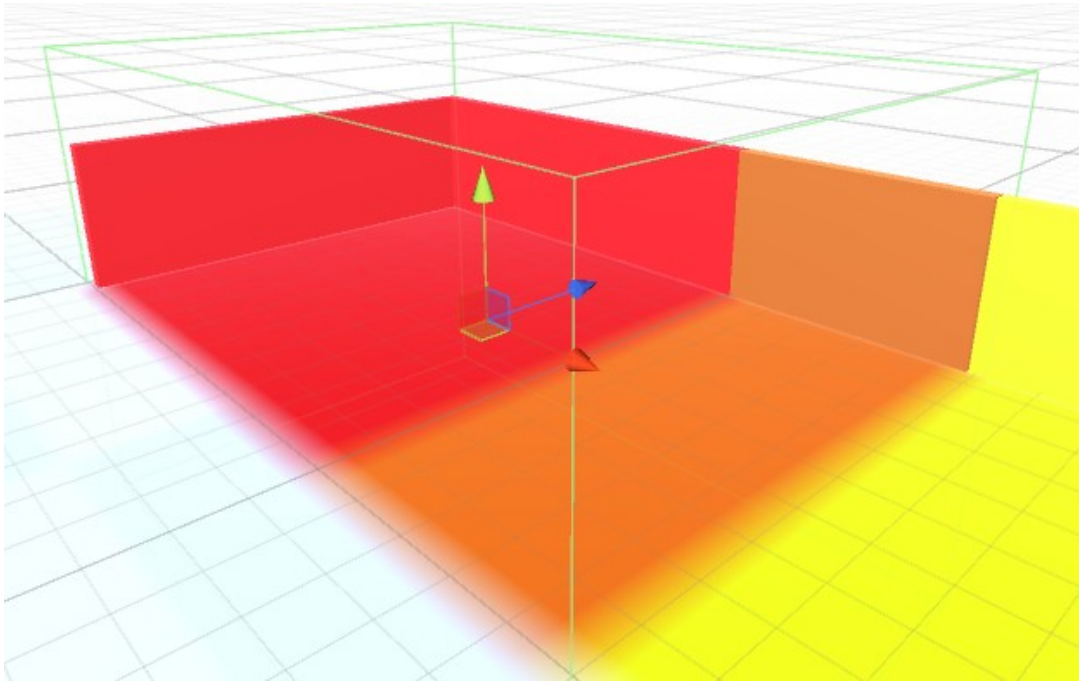
Joonis 6. Mänguobjekti ülesehitus.

4.1.3 Collider

Collider komponent defineerib objekti kuju füüsilise kokkupõrke tuvastamiseks. See ei pea olema täpselt samades mõõtmetes, mis reaalne objekt, vaid on tavaliselt ligikaudsetes mõõtmetes. Kõige lihtsamad, primitiivset tüüpi *collider* komponendid on kasti, kera ja kapsli kujulised. Neid võib lisada ühele objektile nii palju kui vaja, et saavutada soovitud tulemus. Unity toetab ka aga *mesh collider* tüüpi, mida saab vormida täpselt oma vajaduste järgi.

Collider-eid saab lisada mänguobjektidele, millel ei ole *rigidbody* komponenti, et luua näiteks põrandaid, seinu ja muid liikumatuid elemente. Selliseid *collider-eid* kutsutakse staatilisteks *collider-iteks*. Dünaamilisteks *collider-iteks* kutsutakse *rigidbody* komponendiga olevate objektide *collider-eid*. Need kaks *collider-i* tüüpi saavad oma vahel küll suhelda kuid staatilised *collider-id* kokkupõrke tulemusel ei liigu.

Collider-i saab teha Unity-s ka *trigger* tüüpi (vt joonis 7). *Trigger* tüüpi *collider-id* ei käitu kui tahked objektid vaid neisse saab siseneda ja väljuda. Kui sellisesse *collider-isse* siseneb või väljub teine *collider-i* objekt, kutsutakse välja teatud funktsioonid [1].



Joonis 7. *Trigger* tüüpi kastikujuline *collider*.

4.2 Olulisemad loodud komponendid

4.2.1 Skriptimine

Mänguobjekti üheks komponendiks võib olla ka skript. Skriptimine on väga oluline osa mängude loomisel. Need võimaldavad mängul vastata kasutaja sisenditele ning korraldada mängusiseseid sündmusi toimuma siis, kui vaja. Unity võimaldab skripte kirjutada kolmes keeles: C#, Javascript ning Boo. Lisaks tavalistele skriptimise võimalustele pakub Unity ka veel oma poolt suure hulga funktsioone, mis hõlbustavad mängude loomist ning objektidega suhtlemist.

4.2.2 MusicGenerator

MusicGenerator on antud rakenduse kõige kesksem skript. See on vaja lisada mänguobjektile, millest soovitakse muusikalist väljundit ning see kontrollib kogu rakenduse tööd.

4.2.3 Melody

Melody skript tegeleb igale instrumendile eraldi meloodia genereerimisega. Lisaks toimub selle kaudu ka mängu tööle pannes esialgne helifailide importimine mällu.

4.2.4 Percussion

Percussion skript on üpriski sarnane Melody skriptile, kuid tegeleb löökpilli genereerimisega.

4.2.5 ZoneProperties

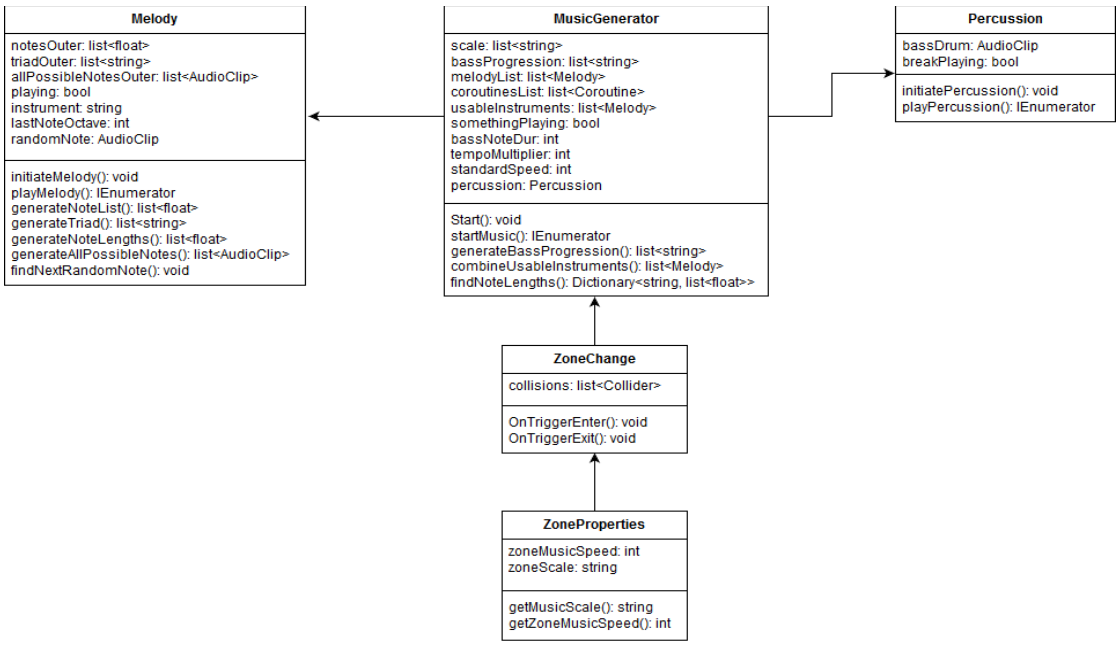
ZoneProperties skript on vaja lisada igale *collider-ile*, millega soovitakse muuta genereeritava heli omadusi. See võimaldab valida uuele alale teise helistiku ning ka tempo.

4.2.6 ZoneChange

ZoneChange skript kontrollib, mis juhtub, kui keskset MusicGenerator-i skripti omav objekt siseneb ZoneProperties skripti omava *trigger* tüüpi *collider-i* alasse. Võtab kasutaja valitud andmed ZoneProperties skripti jaoks ning edastab need MusicGenerator-isse.

4.2.7 Rakenduse struktuur

Joonisel 8 on näidatud kuidas klassid omavahel suhtlevad ning millised on nende peamised muutujad ja funktsioonid.



Joonis 8. Rakenduse struktuur.

5 Rakenduse kirjeldus

Kõigepealt on vaja lisada soovitud mänguobjektile MusicGenerator skript. See võimaldab kasutajal valida endale sobiva standardse helistiku, tempo ning bassi noodi pikkuse. Lisaks sisaldab see ka instrumentide kontrollimiseks vajalikke välju. On võimalik valida, kas kasutatakse dünaamilisi instrumente ehk, kas muusika intensiivsuse kasvades või vähenedes suurendatakse või vähendatakse instrumente ning nende arvu või on see koguaeg konstantne. Veel on võimalik valida löökpillide olemasolu ning ka viie instrumendi kasutatavust ja intensiivsust.

Mängu tööle pannes laetakse kõigepealt sisse kõikide valitud instrumentide helifailid ning omistatakse igale instrumendile kolm helikanalit. Helifailidena on kasutusel Briti orkestri Philharmonia Orchestra poolt salvestatud helinäidised [15]. Bassikäik järgib kõige tuntumat akordide järgnevust, milleks on I - IV - V. Bassinooti valitakse iga muutumise korral juhuslikkuse põhjal. Igal valimisel on 1/10 võimalus tulla ka lisaks seitsmes helistiku noot, pärast mida järgneb iga kord bassinoodiks esimene (vt jaotist 2.3.1).

Meloodia genereerimisel lähtutakse hetkel aktiivsest olevast bassinoodist, mille peale formuleeritakse kolmkõla noodid (vt jaotist 2.3.2) ning otsitakse nendele vastavad helifailid.

```
private List<AudioClip> generateAllPossibleNotes(List<string> triad) {
	List<AudioClip> notes = new List<AudioClip>();
	Regex regex = new Regex(instrument + @"_[" + triad[0] + triad[1] +
				triad[2] + @"]\d");
	for (int i = 0; i < allAudioClips.Count; i++) {
		Match match = regex.Match(allAudioClips[i].name);
		if (match.Success) {
			notes.Add(allAudioClips[i]);
		}
	}
	return notes;
}
```

Joonis 9. Kolmkõla nootide otsimine helifailidest.

Noodipikkuste kombineerimisel arvestatakse kõigepealt tempot ning seejärel ühe bassinoodi pikkust. Esimesena arvutatakse välja etteantud noodipikkused, milleks on veerand-, pool-, täis-, poolteist- ning topeltnoot ja seejärel sobitatakse need juhuslikkuse alusel bassinoodi pikkuse sisse, mis käitub antud programmis taktimõõduna (vt jaotist 2.3.3). Siinkohal tuleb mängu ka kasutaja valitud tempo. Tempo väärtused võivad olla vahemikus 1-10 ning see vahemik on omakorda jaotatud kolmeks. Kui tempo väärtus on piirides 1-3, on tegemist kõige kiirema muusikaga ning kasutatakse vaid veerand- ja poolnoote. Mida lähemal on tempo väärtus 1-le, seda eksponentsiaalselt suurem on tõenäosus veerandnoodi valikuks. Tempo vahemikus 4-7 omab kõiki võimalikke noodipikkuseid. Seekord muudetakse eksponentsiaalselt pool- ja täisnootide tõenäosusi (vt joonis 10). Viimases vahemikus 8-10 kasutatakse ainult täis-, poolteist- ja topeltnoote, millest eksponentsiaalselt suureneb poolteistnoodi tõenäosus. Samadel instrumentidel on kasutusel samasugused noodipikkuste *listid*, kuna muidu läheb mitmete instrumentide lisamisel muusika liiga kaootiliseks.

```
} else if (MusicGenerator.tempoMultiplier >= 4 &&
       MusicGenerator.tempoMultiplier <= 7) {
       double wholeNoteCount = Math.Pow(2, MusicGenerator.tempoMultiplier-2);
       double halfNoteCount = Math.Pow(2, 8 -
                                 MusicGenerator.tempoMultiplier);
       double wholeAndHalfNoteCount;
       double quarterNoteCount;
       if (MusicGenerator.tempoMultiplier >= 6) {
              wholeAndHalfNoteCount = Math.Pow(2, 2);
              quarterNoteCount = Math.Pow(2, 1);
       } else {
              quarterNoteCount = Math.Pow(2, 2);
              wholeAndHalfNoteCount = Math.Pow(2, 1);
       }
       generatedList = addXTimes(generatedList, halfNote, halfNoteCount);
       generatedList = addXTimes(generatedList, quarterNote,
                           quarterNoteCount);
       generatedList = addXTimes(generatedList, wholeNote, wholeNoteCount);
       generatedList = addXTimes(generatedList, doubleNote, wholeNoteCount);
       generatedList = addXTimes(generatedList, wholeAndHalfNote,
                           wholeAndHalfNoteCount);
}
```

Joonis 10. Noodi pikkuse genereerimine vahemikule 4-7 arvestades tempot.

Kõiki eelmainitud listide genereerimisi tehakse ühes tsoonis püsides muusika sujuvuse tagamiseks muusika mängimise ajal. Erandiks on esmakordne käivitamine või tsooni vahetus, mille korral tuleb kõik need andmed uuesti genereerida.

Uue bassinoodi ilmnedes käivitatakse kõikide instrumentide lõimed genereeritud *listidega*. Meloodia mängimiseks võetakse ette noodipikkuste *list*, mis käiakse järjest läbi ning igale noodipikkusele pannakse vastavusse juhuslikult valitud noot. Noodi valimisel on kasutusel *Brown noise*-i idee – valitud noot peab olema kas samas, üks madalam või üks kõrgem oktaavis. See hoiab ära liigse meloodia ebaloomuliku hüppamise (vt joonis 11).

```
LastNoteOctave = (int)System.Char.GetNumericValue(randomNote.name
                    [randomNote.name.Length – 1]);
float oneOctaveDown = lastNoteOctave - 1;
float oneOctaveUp = lastNoteOctave + 1;
Regex usableRegex = new Regex(instrument + @"_\w[" + oneOctaveDown +
                          lastNoteOctave + oneOctaveUp + @"]");
for (int count = 0; count < allPossibleNotes.Count; count++) {
       Match usableMatch = usableRegex.Match(allPossibleNotes[count].name);
       if (usableMatch.Success) {
              usableNotes.Add(allPossibleNotes[count]);
       }
}
randomNoteInt = MusicGenerator.rnd.Next(usableNotes.Count);
randomNote = usableNotes[randomNoteInt];
```

Joonis 11. Järgmise noodi leidmine kasutades Brown noise-i ideed.

Kirjeldatud noodipikkuste listi ei kasuta kaks instrumenti: klaver ja kitarr. Need instrumendid on jäetud taustapillideks, et muusika oleks rohkem struktuursem ja mitte nii kaootiline. Need pillid mängivad kas korraga või väga väikeste vahedega bassinoodi akorde.

Löökpill tuleb mängu ainult tempo vahemikus 1-3. See mängib ühel helikanalil ning sõltuvalt tempost kas 1, 2 või 4 lööki sekundis.

Kui mängija siseneb uute alasse, mis sisaldab ZoneProperties skripti, kutsutakse välja Unity enda funktsioon OnTriggerEnter. Selles funktsioonis võetakse kasutaja poolt määratud omadused uuele tsoonile ning edastatakse need MusicGenerator-i skripti. Seejärel teatatakse kõiki instrumentide lõimi, et on toimunud tsooni vahetus ning need lõpetavad pärast mängitava noodi lõppemist oma töö. Siis genereeritakse uued listid, ning töö jätkub. Kui ollakse samal ajal kahes kattuvas ZoneProperties skripti sisaldavas tsoonis, siis leitakse mängitavaks tempoks nende kahe tsooni tempode aritmeetiline keskmine ning helistikus tuleb hiljuti sisenetud tsooni helistik. Kui kasutaja lahkub alast, teostatakse sarnased tegevused, ainult väljakutsutavaks funktsiooniks on OnTriggerExit.

6 Hinnang tulemusele

Rakenduse eesmärgiks oli luua generatiivselt mängu meeleolule vastavat muusikat, mis ka õnnestus. Kõik tingimused, nagu näiteks helistiku valik, tempo muutus ning ka instrumentide dünaamiline muutumine, said edukalt täidetud. Küll aga ei ole programmi poolt loodav muusika nii loomulik kui sooviti ning arvatavasti tulevastes mängudes või Unity 3D lisade poes kasutust ei leia.

Programmil on palju edasiarendusruumi. Antud bakalaureusetöö raames loodud skriptid töötavad hästi generatiivse muusika loomise baasina, kuid vajaksid parema muusika loomiseks keerukamaid algoritme. Näiteks võiks tulevikus implementeerida meloodia genereerimisse Markovi ahelaid, et muusika tunduks terviklikum, ning teha rütmipillide sektsioon mitmekesisemaks erinevate trummide või muude instrumentide osadega.

7 Kokkuvõte

Bakalaurusetöö eesmärgiks oli luua generatiivset muusikat loov rakendus, mis muudab muusikat vastavalt mängu meeleolule, lisades või ära võttes instrumente, ning kiirendades või aeglustades tempot.

Rakenduse loomisel sai kasutatud Unity mängumootorit, mis pakkus erinevate alade tuvastamiseks funktsioone ning võimaldas lihtsasti kontrollida loodava muusika dünaamilisust. Rakenduse kasutajal on võimalik valida nii standardne kui ka igale alale spetsiifiline muusika tempo ning helistik ja kasutatavad pillid. Rakenduse käivitamisel hakkab programm looma generatiivselt muusikat.

Töös toodi välja erinevad generatiivse muusika loomise algoritmid ning põhilised teadmised muusikateooriast, mille alusel rakendus loodi. Võrreldi omavahel stohhastilisi ja deterministlikke algoritme ning põhjendati, miks antud rakenduse loomisel on kasutatud just stohhastilisi meetodeid. Kirjeldati veel ka programmi loomisel kasutatud tehnoloogiaid ja põhjendati nende valikut.

Kokkuvõttes võib öelda, et töö eesmärgid said täidetud. Bakalaurusetöö võimalikeks edasiarendusteks oleks näiteks paremate algoritmide implementeerimine, reaalsetes mängudes kasutamine ning ka Unity lisade poes müümine.

Kasutatud kirjandus

[1] *Colliders.* [WWW] https://docs.unity3d.com/Manual/CollidersOverview.html (10.05.2017)

[2] Crossley-Holland, P. *Rhythm.* [WWW] https://www.britannica.com/art/rhythm-music (12.05.2017)

[3] Frishert, S. *Why adaptive audio systems?* [WWW] http://stijnfrishert.com/why-adaptive-audio-systems/ (15.05.2017)

[4] *GameObjects.* [WWW] https://docs.unity3d.com/Manual/GameObjects.html (10.05.2017)

[5] *Introduction to the C# Language and the .NET Framework.* [WWW] https://docs.microsoft.com/en-us/dotnet/articles/csharp/getting-started/introduction-to-the-csharp-language-and-the-net-framework (17.05.2017)

[6] Kaljuste, H., Loitme, T. Laulik XI kl. Tallinn: Valgus, 1978.

[7] Krigul, E. Solfedžo II klassile. Haabneeme: Eesti Muusikakoolide Liit, 2010.

[8] *License comparisons.* [WWW] http://download.unity3d.com/unity/licenses (15.05.2017)

[9] *Microsoft Visual Studio.* [WWW] https://en.wikipedia.org/wiki/Microsoft_Visual_Studio (9.05.2017)

[10] Paulus, E. – *The use of generative music systems for interactive media.* [WWW] http://www.eude.nl/projects/paper-gmsim/paper-GMSIM.pdf (8.05.2017)

[11] Raik, P. Solfedžo I. Tallinn: P. Raik, 2003.

[12] *Scenes.* [WWW] https://docs.unity3d.com/Manual/CreatingScenes.html (10.05.2017)

[13] *Scripting.* [WWW] https://docs.unity3d.com/Manual/ScriptingSection.html (10.05.2017)

[14] Skuin, A., Sepp, K. Muusikaõpik 9.klass. Tallinn: Avita, 2009.

[15] *Sound samples* [WWW] http://www.philharmonia.co.uk/explore/sound_samples (23.03.2017)

[16] Unity. [WWW] https://et.wikipedia.org/wiki/Unity (9.05.2017)

[17] Young, D.M. – *Adaptive game music: The evolution and future of dynamic mysic systems in video games.*[WWW] https://etd.ohiolink.edu/!etd.send_file?accession=ouhonors1340112710&disposition=inline (7.05.2017)