Aapo Ritvanen

Ääniliitännäisen toteuttaminen Unreal Engine -pelimoottorilla

Insinööri (AMK)
Tieto- ja viestintätekniikka
Syksy 2021

KAMK • University of Applied Sciences

Tiivistelmä

Tekijä: Ritvanen Aapo

Työn nimi: Ääniliitännäisen toteuttaminen Unreal Engine -pelimoottorilla

Tutkintonimike: Insinööri (AMK), tieto- ja viestintätekniikka

Asiasanat: peliääni, äänitehoste, liitännäinen, ääniohjelmointi, äänisuunnittelu, äänten toistaminen

Työn toimeksiantajana toimi Kajaanin Ammattikorkeakoulun alla toimiva kehitysyksikkö Clever Simulation Entertainment (Clever). Clever on erikoistunut virtuaalitodellisuuden (VR) ja lisätyn todellisuuden (AR) teknologioihin, ja kehittää ratkaisuja pääasiassa teollisuus- ja opetuskäyttöön.

Työn tavoitteena oli toteuttaa liitännäinen, joka lisää Cleverin pelien fysiikkapohjaisiin esineisiin törmäys-, raahaus- ja vierimisäänet. Liitännäinen voidaan lisätä mihin tahansa Cleverin Unreal Engine -pelimoottorilla toimivaan peliprojektiin. Työssä oli tarkoituksena käyttää valmiiksi äänitettyjä eri materiaalien ääniä, ja toistaa niitä mahdollisimman luontevasti peleissä.

Peliääniä ilmestyi 1930-luvulla pelihallien ja kasinoiden myötä. Videopelien ja pelikonsoleiden yleistyessä ja kehittyessä nopeaa vauhtia olivat peliäänet myös osana kehitystä. Uusia äänipiirejä kehitettiin tiheään tahtiin ja teknologia tuli paremmaksi. Nykyään monet pelit havittelevat realistisuutta, jossa äänet ovat tärkeässä osassa. Peliäänet parantavat esimerkiksi immersiota ja äänipalautetta, joiden parantamista myös työn liitännäisellä tavoiteltiin. Peliääniin saadaan luonnollisuutta varioimalla niitä. Työn liitännäinen varioi ääniä esimerkiksi muuttamalla äänenvoimakuutta ja -korkeutta sekä valitsemalla satunnaisia ääniä.

Työssä kehitettiin toimiva liitännäinen, joka lisäsi pelien fysiikkapohjaisiin esineisiin törmäys- ja raahausäänet. Työssä todettiin, että äänet toivat immersiota pelin esineisiin. Vierimisäänet jäivät liitännäisestä pois aikarajoitteiden takia. Liitännäistä varten tehtiin myös oma peliprojekti, jossa törmäys- ja raahausääniä voitiin testata ja kehittää.

KAMK • University of Applied Sciences

Abstract

Author: Ritvanen Aapo

Title of the Publication: Developing Audio Plugin with Unreal Engine

Degree Title: Bachelor of Engineering, Information and Communication Technology

Keywords: game sound, sound effect, SFX, plug-in, audio programming, sound design, sound playback

This thesis was commissioned by Clever Simulation Entertainment (Clever), a development unit that works under Kajaani University of Applied Sciences. Clever is specialised in virtual reality (VR) and augmented reality (AR) technologies. They develop solutions primarily for manufacturing and educational purposes.

The goal of this thesis was to create a plugin that adds collision, dragging and rolling sounds to physical game objects in the games of Clever. The plugin can be added to any Clever's project that uses Unreal Engine game engine. The idea was to use different pre-recorded material sounds and play them as naturally as possible.

In the 1930s, game sounds first appeared along arcades and casinos. When video games and game consoles became more common and evolved rapidly, game sounds also followed the development. New sound chips were developed rapidly, and technology advanced. Nowadays many games attempt to be as realistic as possible in which sounds play a major part. Game sounds can improve for example immersion and audio feedback which were also the goals of the thesis. By adding variation to the sounds, they can be made sound more natural. The plugin creates variation by changing the volume and pitch in the sounds, as well as randomizing which sound to use.

In this thesis, a fully working plugin was developed that added collision and dragging sounds to physical game objects. It was concluded that sounds added immersion to the game objects. Rolling sounds were left out from the plugin due to the time constraints. A game project was also created for the development and testing of the plugin.

Sisällys

Symboliluettelo

Actor	Esine tai asia, joista Unreal Engine -pelikentät koostuvat
AR	Augmented Reality (suom. lisätty todellisuus)
Blueprint	Ohjelmointikieli Unreal Engine -pelimoottorissa
DAC	Digital to Analog Converter (suom. digitaali-analogimuunnin), äänipiiri
DSP	Digital Signal Processing (suom. digitaalinen signaalinkäsittely)
Event	Ohjelmallinen tapahtuma
FM	Frequency Modulation (suom. taajuusmodulaatio)
PCM	Pulse Code Modulation (suom. pulssimodulaatio)
PSG	Programmable Sound Generator, äänipiiri
Tietue	Ohjelmallinen tietokokonaisuus (engl. structure)
UE	Unreal Engine, pelimoottori
VR	Virtual Reality (suom. virtuaalitodellisuus)
Ääninäyte	Digitaalisia aallonmuodon arvoja tietyllä aikavälillä (engl. sample)
Äänisilmukka	Ääni, jota voidaan toistaa toistuvasti ja saumattomasti (engl. loop)

1 Johdanto

Peliääniä ei pidetä usein tärkeinä pelin kehityksessä. Niihin ei yleensä käytetä paljon aikaa, ja ne lisätään nopeasti vasta kehityksen loppupäässä. Vaikka videopelit ovat visuaalinen viihteen muoto ja niiden pääpainopiste on yleensä visuaalisessa puolessa, ovat myös äänet yhtä tärkeitä. Äänet ovat kuitenkin puolet pelikokemuksesta. Peliäänet voivat tuoda paljon elävyyttä, tunnetta ja uskottavuutta peleihin sekä antaa pelaajille tärkeää tietoa ympäröivästä pelimaailmasta.

Tämän työn aiheena on pelien äänitehosteet sekä niiden toistaminen. Työn tarkoitus on kerätä tietoa peliäänistä ja siitä, kuinka niitä lisätään peleihin niin, että ne kuulostavat luonnollisilta.

Työn toimeksiantajana toimii Kajaanin Ammattikorkeakoulun alla toimiva kehitysyksikkö Clever Simulation Entertainment (Clever). Clever on erikoistunut virtuaalitodellisuuden (VR) ja lisätyn todellisuuden (AR) teknologioihin ja kehittää ratkaisuja pääasiassa teollisuus- ja opetuskäyttöön [1].

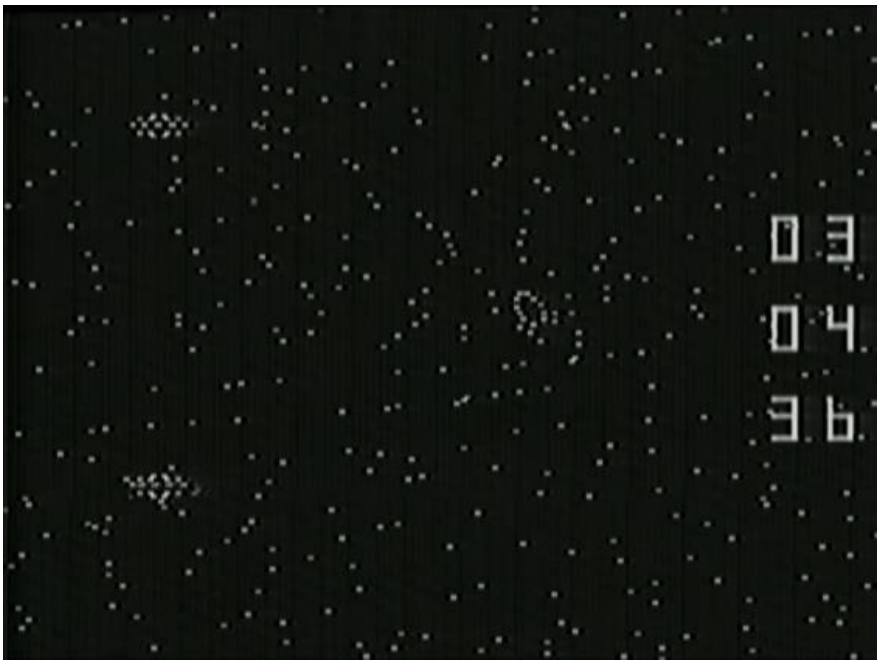
Työn tavoitteena on toteuttaa liitännäinen, joka lisää Cleverin pelien fysiikkapohjaisiin esineisiin törmäys-, raahaus- ja vierimisäänet. Työn tarkoitus on antaa Cleverille valmis kokonaisuus, jota he pystyvät käyttämään ja lisäämään projekteihinsa helposti. Liitännäisen tarkoituksena on lisätä immersiota, parantaa käyttäjäkokemusta, antaa yhtenäinen toimintapa fyysisten äänten toistamiselle sekä nopeuttaa äänten lisäämistä peliin. Työ oli myös hyvä tilaisuus tutkia ääniin liittyviä asioita sekä oppia tehdä käytännössä ääniohjelmointia, jotka ovat itselleni kiinnostavia asioita.

Työn alussa tarkastellaan peliäänten historiaa ja tarkoitusta sekä niiden jakamista eri kategorioihin. Tämän jälkeen käydään läpi eri tapoja, miten ääniä voidaan toistaa pelimaailmassa ja luoda näin ääniin luonnollista variaatiota. Käytännön osuudessa käydään läpi työn taustoitus ja syyt, miksi työ tehtiin sekä miten se toteutettiin. Lopussa pohditaan työn onnistumista ja työ päätetään yhteenvetoon.

2 Peliäänten historia ja kehitys

Varhaisimpia peliääniä löytyi 1930-luvulla flippereistä (engl. pinball), joita pelattiin sen ajan pelihalleissa (engl. arcade). Flippereiden äänet koostuivat erilaisista kellon ja summerin äänistä, joiden tarkoitus oli houkutella pelaajia ja luoda innostusta. Samoihin aikoihin peliääniä kuultiin myös kasinoiden peliautomaateissa, joita yleensä valmistivat samat yritykset, jotka tekivät flippereitä. Peliautomaateissa äänet koostuivat suurimmaksi osin kellon pirinästä ja kolikon kolahduksista, jotka toivat voiton tuntua, kun pelaaja sai voittoja tai pääsi lähelle voittoa. Voitaisiin sanoa, että peliäänten suurin tarkoitus oli houkutella pelaajia, luoda innostusta ja pitää pelaajat kiinnostuneina peleistä. [2, s. 7–8.]

Flippereiden ja varhaisten pelihallien jälkeen tulivat videopelihallit (engl. video game arcade). Ensimmäinen kaupallisesti sarjatuotettu arcade-videopeli oli Computer Space, joka tuli julkisuuteen vuonna 1971 [2, s. 8; 3 s. 7]. Computer Space on avaruustaistelupeli, jossa pelaajan liikuttama raketti ampuu ja väistää pelissä liikkuvia UFOja. UFOt yrittävät tuhota pelaajan törmäämällä siihen tai ampumalla sitä. Peli laskee pisteitä näytön sivulla, montako UFOa pelaaja on tuhonnut ja montako kertaa pelaaja on tuhoutunut. Peli loppuu, kun yhden kierroksen aika loppuu. Computer Spacessa on erilaisia avaruustaisteluääniä, esimerkiksi rakettimoottorin, ohjusten ja räjähdysten ääniä. Äänet koostuvat suurimmaksi osin erilaisista piippauksista ja suhinoista. [4.] Kuvassa 1 on esitetty kuvankaappaus Computer Space -pelistä.



Kuva 1. Kuvankaappaus Computer Space -videopelistä. [4.]

Vuonna 1972 markkinoille tuli Pong, josta tuli ensimmäinen menestys arcade-videopeli. Pong teki arcade-videopeleistä sensaation ja toi monia yrityksiä pelialalle. [2, s. 8; 3 s. 7; 5, s. 3.] Pong on kaksin pelattava pöytätennispeli, jossa pelaajat liikuttavat omia mailojaan pystysuunnassa ja yrittävät osua edestakaisin liikkuvaa palloa niin, että se menee vastustajan mailan taakse. Pelaaja saa jokaisesta vastustajan mailan taakse menneestä pallosta pisteen. Pelin äänet koostuvat lyhyistä yksinkertaisista piippauksista, jotka kuuluvat, kun pallo osuu mailaan tai pelikentän reunoihin. [5, s. 55; 6.] Pongin äänet olivat myös osittain syynä, miksi videopeliäänistä yleisesti tuli erittäin tunnettuja [2, s. 8]. Kuvassa 2 on esitetty kuvankaappaus Pong-pelistä.

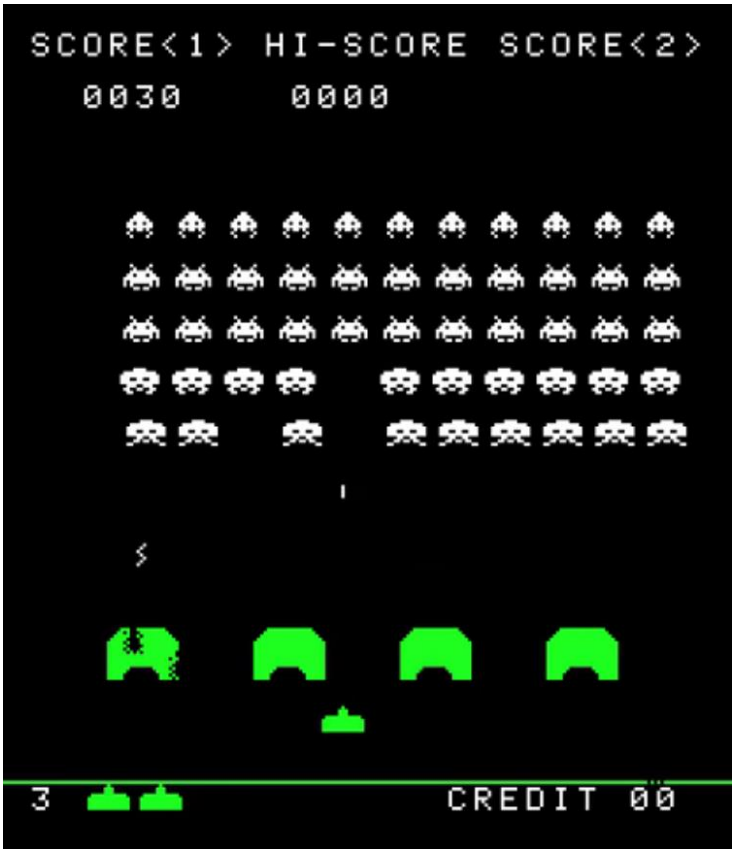
Kuva 2. Kuvankaappaus Pong-videopelistä. [6.]

Computer Space ja Pong olivat ensimmäisiä videopelejä, joissa oli ääniä. Vaikka varhaisten videopelien äänet eivät ole nykypäivän standardeissa niin ihmeellisiä, ne olivat vaikuttavia omana aikanaan ja antoivat paljon elävyyttä peleihin. Peleille ja peliäänille tämä oli iso askel oikeaan suuntaan. Kuvassa 3 näkyy, miltä Computer Spacen ja Pongin kabinetit näyttävät.

Kuva 3. Pong (vasemmalla) ja Computer Space (oikealla) -videopelien kabinetit. [7.]

Ensimmäisten videopelien äänet koostuivat yleensä hyvin yksinkertaisista ääniefekteistä, jotka olivat monotonisia piippauksia, surinoita ja suhinoita. Koska arcade-koneissa oli rajallisesti muistia, äänipiirejä sekä muita teknisiä komponentteja, äänien täytyi olla yksinkertaisia [2, s. 8–9; 3, s. 6]. Ääniin ei myöskään käytetty paljon aikaa, koska ohjelmointi vei kaiken ajan pelin kehityksestä. Äänet jäivät monesti toissijaiseksi asiaksi kehityksen aikana. Pelejä tehtiin myös hyvin pienillä tiimeillä, joista kaikki oli yleensä ohjelmoijia. [3, s. 6.]

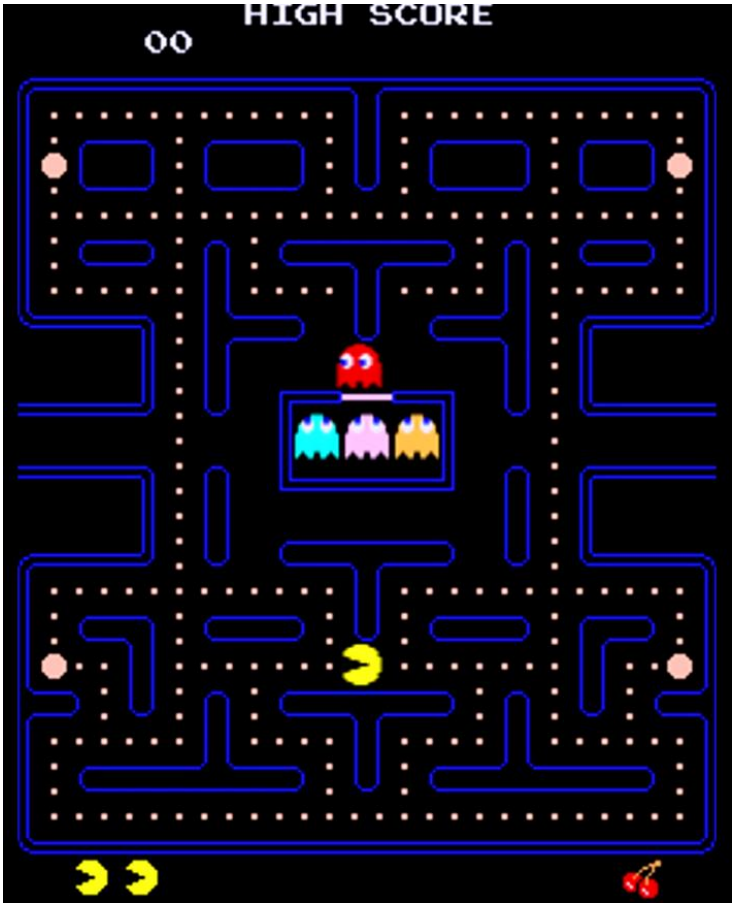
Varhaisissa arcade-videopeleissä ei ollut myöskään peliaikaista jatkuvaa musiikkia, vaan peleillä oli yleensä lyhyt intro-musiikki ja pelin loppumusiikki. Space Invaders vuodelta 1978 ja Asteroids vuodelta 1979 toivat ensimmäisinä jatkuvan musiikin videopeleihin, vaikka kyseisten pelien musiikki oli enemmänkin pelin tahdin pitämistä 2–4 nuotin avulla. Kuvassa 4 on esitetty Space Invaders -videopelin kuvankaappaus, jossa yläkulman viholliset lähestyvät alaspäin musiikin tahdissa.



Kuva 4. Kuvankaappaus Space Invaders -videopelistä. [8.]

Niin kuin ääniä, musiikkia oli hidasta ja haastavaa kehittää varhaisille arcade-laitteille teknisten rajoitteiden takia. Säveliä ja ääniä jouduttiin luomaan käsin yhdistämällä erilaisia elektronisia komponentteja tai vaihtamalla ykkösiä ja nollia suoraan elektronisista komponenteista. Koska musiikin tekeminen oli aikaa vievää ja hankalaa, ja sen hyötyä ei nähty arcade-ympäristöissä, olivat ääniefektit yleensä tärkeämmässä roolissa kuin musiikki. [2, s. 9–12.] Vasta vuodesta 1984 eteenpäin pelimusiikkien silmukat (engl. loop) eli toistuvat musiikkipätkät tulivat yleisiksi videopeleissä [2, s. 19].

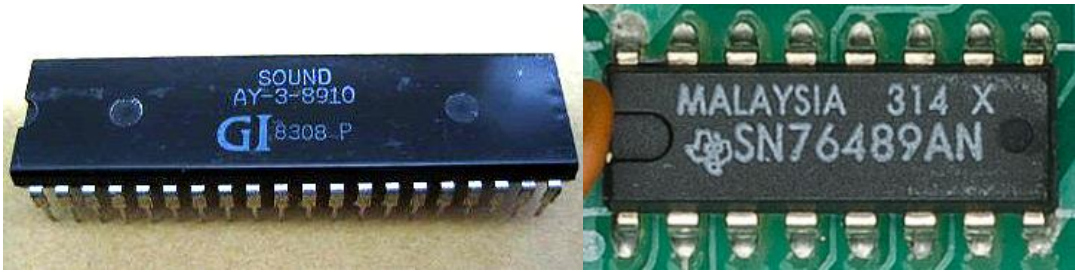
Arcade-pelit ovat olleet myös osaksi äänellistä ja musiikillista popkulttuuria, niin kuin Pac-Man vuodelta 1980. Pac-Man on keltainen pyöreä hahmo, joka syö pelikentässä löytyviä palloja ja väistää pelissä jahtaavia haamuja. Kun Pac-Man syö pelikentästä löytyviä keltaisia palloja, se päästää ”waka waka” -äänen, joka on ollut todella tunnettu ääni ja osa nykypäivänkin populaarimusiikin historiaa. Pac-Man oli myös todennäköisesti ensimmäinen videopeli, joka sisälsi kokopitkän välianimaatiomusiikin. [5, s. 131.] Kuvassa 5 on esitetty kuvankaappaus pelistä, jossa näkyvät pelin kenttä, Pac-Man ja haamut.



Kuva 5. Kuvankaappaus Pac-Man-videopelistä. [9.]

1980-luvun mennessä monet laitevalmistajat lisäsivät arcade-laitteisiin äänille omistettuja äänipiirejä nimeltään programmable sound generator (PSG). PSG:t sisälsivät erilaisia oskillaattoreita, jotka tuottivat tiettyjä äänen aaltomuotoja. Muuttamalla eri asetuksia äänipiiristä ajon aikana, pystyttiin saamaan äänipiiristä ulos eri sävyisiä, pituisia ja eri sävelkorkeudella olevia ääniä. Monet PSG-äänipiirit pystyivät toistamaan kolmea tai neljää eri ääntä yhtäaikaisesti. [2, s. 10–15.]

Vaikka äänet kuulostivat suurimmaksi osin erilaisilta piippauksilta ja suhinoilta, PSG:t lisäsivät pelimusiikkeihin sävelrikkautta ja äänensävyjä sekä paransivat ääniefektejä. Esimerkiksi General Instumentin valmistama AY-3-8910 ja Texas Instumentin SN76489 olivat kaksi varhaisinta ja myös paljon käytettyä PSG-äänipiiriä. Kyseisiä äänipiirejä käytettiin monissa arcade-laitteissa ja myöhemmin myös varhaisissa kotitietokoneissa ja pelikonsoleissa. Kyseisiä äänipiirejä käytettiin esimerkiksi Segan Carnival (1980), Atarin Centipede (1980) ja Nintendon Sheriff (1980) arcade-videopeleissä. Kyseisiä äänipiirejä käyttivät myös esimerkiksi Segan Master System - ja Genesis-pelikonsolit, sekä Sinclairin ZX Spectrum ja BBC:n Micro-kotitietokoneet. [2, s. 12–15.] Kuvassa 6 on esitetty AY-3-8910- ja SN76489-äänipiirit.



Kuva 6. AY-3-8910-äänipiiri (vasemmalla) ja SN76489-äänipiiri (oikealla). [10; 11.]

PSG-äänipiirien lisäksi arcade-laitteisiin alettiin lisäämään digitaali-analogimuuntimia eli digital to analog converter (DAC) -äänipiirejä. DAC itsessään ei luo ääntä, vaan kyseinen äänipiiri muuttaa digitaalista äänitietoa elektroniseksi eli analogiseksi äänisignaaliksi, jota voidaan toistaa kaiuttimista. Ääni on tallennettu muistiin digitaalisina aallonmuodon arvoina tietyllä aikavälillä eli ääninäytteinä (engl. sample) ja kun ääntä halutaan toistaa, muutettiin ääninäytetieto DAC:llä analogiseksi äänisignaaliksi. DAC:t erosivat PSG-äänipiireistä sillä, että DAC:llä pystyttiin toistamaan mitä vain ääniä ja ne eivät olleet rajoittuneita vain tiettyihin aallonmuotoihin. Isoin ongelma äänien toistamisessa DAC:llä oli ääninäytteiden tallentaminen muistiin, koska muistia oli hyvin rajoitetusti. Äänet täytyi olla todella lyhyitä ja monesti huonolaatuisia. Tämän takia DAC-äänipiirejä käytettiin yleensä vain lyhyisiin ääniefekteihin ja PSG-äänipiirejä musiikkeihin ja muihin ääniin. [2, s. 13–15.]

Arcade-videopelien suosion ansiosta mikroprosessorit halpenivat ja markkinoille alkoi tulemaan arcade-videopelien lisäksi kotona pelattavia pelikonsoleita (engl. home video game console). Magnavox Odyssey oli ensimmäinen kaupallinen pelikonsoli, joka tuli vuonna 1972 markkinoille, mutta se ei sisältänyt ääniä. [2, s. 20; 5, s. 3.] Vuodesta 1975 eteenpäin Atari sekä monet muut yritykset toivat kotona pelattavia Pong-pelejä markkinoille, joista monet käyttivät General Instrumentin AY-3-8500 äänipiirejä äänten toistamista varten. Atari toi vuonna 1977 Video Computer System -pelikonsolin, joka tunnetaan myös nimellä Atari 2600. Kuvassa 7 näkyy vaihdettava pelit eli pelimoduulit (engl. game cartridge), jonka Atari 2600 teki suosituksi. Atari 2600 sisälsi oman PSG-äänipiirin, mutta se oli rajoittuneempi ja musiikin teon kannalta epävireisempi kuin monet muut aikansa PSG-piirit. Muutama vuosi myöhemmin Mattelin Intellivision ja Colecon ColecoVision -pelikonsolit tulivat markkinoille. Intellivision ja ColecoVision sisälsivät aiemmin mainittuja arcade-laitteissa käytettyjä AY-3-8910 ja SN76489 äänipiirejä, jotka olivat huomattavasti parempia kuin Atari 2600:n äänipiiri. [2, s. 20–24.]

Kuva 7. Atari 2600 -pelikonsoli ja sen pelimoduuleja. [12.]

1980-luvun alkupuolella peliala kohtasi monien syiden seurauksesta myyntien voimakasta laskua. Tämän seurauksena Pohjois-Amerikassa tapahtui vuoden 1983 videopelilama, jonka aikana monet yritykset menettivät miljoonia dollareita ja yrityksiä meni konkurssiin tai lähti pelialalta. Vasta vuonna 1985 peliala elpyi laskusta, kun Nintendo julkaisi Nintendo Entertainment System (NES) -pelikonsolin. [2, s. 24; 5, s. 2–5.] Nintendo hallitsi jo Japanin pelikonsolimarkkinoita, mutta se ei ollut vielä päässyt kunnolla Amerikan markkinoille. Samaa pelikonsolia oli jo myyty miljoonia Japanissa nimellä Family Computer eli Famicom. Ongelmana oli, että Amerikan videopelilama oli saanut jälleenmyyjät menettämään kiinnostuksensa videopeleihin. Videopeleille oli tullut huono maine, minkä takia sanaa videopeli yritettiin välttää mainonnassa. Nintendo joutui markkinoimaan pelikonsolia eri tavoin ja luomaan kuvan, että NES on muutakin kuin vain pelikonsoli. Pelikonsolia markkinoitiin lelurobotin ja -pyssyn avustuksella, Famicom uudelleen nimettiin NES:ksi ja pelikonsolin ulkonäköä muutettiin. Kuvasta 8 näkee, miten Famicomin lelumainen ulkonäkö muutettiin näyttämään enemmän videonauhurilaiteen tyylistä laitetta. [2, s. 24; 13, s. 278–297; 14.] Näiden muutosten sekä esimerkiksi Super Mario Bros. (1985) ja The Legend of Zelda (1986) hittipelien avulla Nintendo voitti Amerikan puolelleen. NES sisälsi oman PSG-äänipiirin, joka pystyi toistamaan viittä eri ääntä samanaikaisesti ja toi monia kontrolloitavia asetuksia äänille. Äänipiirissä oli myös mahdollisuus toistaa lyhyitä ääninäytteitä kahdella eri tavalla. [2, s. 24–25; 3, s. 8.]

Kuva 8. Famicom (vasemmalla) ja Nintendo Entertainment System (oikealla) -pelikonsolit. [15; 16.]

Laman jälkeen peliala alkoi kasvamaan uudelleen ja uusia ja parempia pelikonsoleita tuli tiheään tahtiin. Pelikonsolit siirtyivät 8-bittisistä 16-bittisiin, jotka toivat parempia grafiikoita ja ääniä. Yksi äänellinen parannus oli taajuusmodulaatio syntetisaattorit eli frequency modulation (FM) syntetisaattorit. FM-syntetisaattoreista saatiin paljon enemmän irti erilaisia ääniä ja äänensävyjä kuin tavallisista PSG-äänipiireistä. Ne loivat myös realistisempia ääniefektejä. FM-syntetisaattorit olivat hyviä luomaan esimerkiksi urku, elektronisen pianon tai kielisoittimien tyylisiä ääniä, joita kuultiin paljon sen ajan peleissä. [2, s. 38–40; 3, s. 9.] Vuonna 1989 Sega julkaisi 16-bittisen Genesis -pelikonsolin, joka oli tunnetuin FM-syntetisaattoria käyttävä pelikonsoli. [2, s. 40; 13, s. 404.] Genesiksen äänet olivat huomattavasti paremmat NES:iin verrattuna kahden äänipiirinsä ansiosta. Genesis pystyi myös toistamaan kymmenen ääntä yhtäaikaisesti. [2, s. 39–40; 3, s. 8.] Vasta 1991 Nintendo julkaisi oman 16-bittisen Super Nintendo Entertainment (SNES) -pelikonsolin, joka oli taas huomattavasti parempi grafiikoiltaan ja ääniltään kuin Genesis. SNES:in instrumentit olivat ääninäytepohjaisia, minkä takia äänet olivat paljon realistisempia. SNES sisälsi myös monia reaaliaikaisia digitaalisen signaalinkäsittelyn efektejä eli DSP-efektejä (engl. digital signal prosessing). Näitä olivat esimerkiksi erilaiset kaikuefektit ja äänen suodatinefektit. [2, s. 45–46; 3, s. 9.]

Ääninäytteiden hyödyntäminen yleistyi DAC-piirien ja pulssikoodimodulaatio-menetelmän ansiosta. Pulssimodulaatio eli pulse code modulation (PCM) oli tapa tallentaa ja muuntaa nauhoitettua ääntä digitaaliseksi ääninäytetiedoksi, jota voitiin muuntaa taas DAC:illä takaisin toistettavaksi ääneksi [2, s. 14]. Muistin lisääntyessä ja CD-teknologian yleistyessä pystyttiin toistamaan parempilaatuisia ja pidempiä ääniä. Musiikkeja ei enää tarvinnut ohjelmoida tietyille äänipiireille, vaan CD:istä pystyttiin toistamaan suoraan digitaalista musiikkia pelin aikana. Tämä mahdollisti esimerkiksi äänitettyjen live musiikin, dialogin ja ääniefektien toistamisen pelin aikana. [2, s. 63.] Nykypäivänä äänet ovat suurimmaksi osin digitaalisina ääninäytteinä, joita toistetaan DAC:eillä, koska muisti ja teknologia eivät ole enää samalla tavalla rajana äänten toistolle.

Tietokoneet kävivät myös saman kehityskaaren pelikonsoleiden kanssa. Monet tietokoneet sisälsivät PSG-äänipiirejä 1980-luvulla, joiden avulla toistettiin tietokonepelien ääniä [2, s. 28–33]. Kun 1990-luvulla äänikortit (engl. sound card) tulivat, myös digitaalisesta musiikista ja äänistä tuli yleisiä. [3, s. 12.] Vaikka tietokoneilla pystyttiin pelaamaan pelejä, ne mahdollistivat myös pelien ja peliäänien tekemisen. Kun 1980–1990-luvun vaihteessa tietokoneista tuli yleisiä, myös äänten tuottamiselle tarkoitetut ohjelmistot yleistyivät ja halpenivat. Tämä ansiosta nykypäivänä kuka vain pystyy tekemään tietokoneellaan peliääniä- ja musiikkia. [3, s. 18.]

3 Peliäänten tarkoitus

Peliäänillä on kolmenlaista tarkoitusta: antaa pelaajalle palautetta pelissä tapahtuvista tilanteista, parantaa pelikokemusta tuomalla immersiota ja tuoda pelille lisää viihdearvoa [3, s. 4].

3.1 Äänipalaute

Kun pelaaja tekee jotain pelissä, esimerkiksi painaa näppäintä, pelkkä näytöltä tuleva visuaalinen palaute ei välttämättä riitä saamaan pelaajaa ”uskomaan” tai ymmärtämään pelissä tapahtunutta. Jos näytöltä tulevaan kuvaan yhdistetään toisenlainen aistiärsyke, tässä tapauksessa ääni, pelaajan mielessä kuva ja ääni yhdistyvät yhdeksi ja toiminto tuntuu myös pelaajan alitajunnassa aidommalta. Näin ääni vahvistaa näytöllä tapahtunutta ja toimii pelaajalle palautteena. [3, s. 4.] Kun ihminen yhdistää samaan aikaan tapahtuvan kuvan ja äänen mielessään yhdeksi, sitä kutsutaan termillä ”synchresis” [17, s. 20]. Pelaaja yhdistää myös tekemänsä toiminnon, tässä tapauksessa näppäimen painalluksen, ei ainoastaan kuvaan, mutta myös ääneen. Samaan aikaan tapahtuvan äänen ja toiminnon yhdistymistä mielessä yhdeksi kutsutaan termillä ”kinesonic synchresis”. [17, s. 32.]

Koska ääni ja toiminto yhdistyy melkein välittömästi ihmisen mielessä, äänen ja toiminnon yhteydestä tulee itsestään selvä ja ennalta arvattava. Tämän takia pelaaja myös odottaa, että ääni kuuluu uudestaan, jos hän tekee saman toiminnon uudestaan. Tämä äänen ja sitä vastaavan toiminnon toistettavuus on yksi tärkeimmistä syistä, miksi peliäänet toimivat hyvänä palautteen antajana pelaajille. Se parantaa niiden pelaajien tehokkuutta, jotka käyttävät äänipalautteita hyväksi helpottaakseen pelin pelaamista. [17, s. 32–33.]

3.2 Immersio

Pelaaja voi kokea pelatessaan, että hän ”uppoutuu” tai syventyy pelimaailman sisään ja tuntee vahvaa yhteyttä siihen. Pelaajasta voi tuntua, että hän on osa peliä. Tällöin yleensä ulkopuolinen maailma unohtuu osittain tai jopa kokonaan ja ulkopuolisen maailman ärsykkeet eivät haittaa pelaamista. Tätä ilmiötä kutsutaan yleisesti immersioksi, vaikka immersio terminä on monesti kiistelty ja heikosti määritelty. [3, s. 5; 18, s. 36–39.]

Immersiota pidetään yhtenä tärkeimmistä asioista, joka tekee pelistä pelaamisen arvoisen. Immersiosta puhutaan paljon eri osa-alueissa, koska se on tärkeä pelaajille, pelinkehitykselle ja pelitutkijoille. Pelaajat, pelisuunnittelijat ja pelitutkijat ovat samaa mieltä siitä, että immersio kuuluu osaksi pelikokemusta. Koska suurin osa pelaajista nauttii peliin syventymisestä, immersio otetaan huomioon pelin kehityksen aikana. Pelisuunnittelijoille on tärkeää ymmärtää, mikä aiheuttaa immersiota pelaajissa. Immersion luominen on tärkeää, koska suunnittelijat haluavat tehdä pelaajien arvostamia pelejä. Jos pelaaja on syventynyt peliin tarpeeksi, se voi myös auttaa peittämään muita puutteita esimerkiksi pelisuunnittelussa. [18, s. 36–37.]

Vaikka immersion tiedetään olevan tärkeä asia peleissä, ei tiedetä syytä, mistä se syntyy. Myöskään äänten osallisuudesta immersion syntymiseen tiedetään vähän. Syynä tähän voi olla, että syventyminen tapahtuu melko tiedostamattomasti, minkä takia äänen ja immersion yhteyttä on hankala määrittää. Peliäänien tiedetään kuitenkin parantavan immersiota, mutta tarkkaa teoriaa siitä, miten peliäänet vaikuttavat suoraan immersioon, ei vielä ole. [18, s. 8; 18, s. 37.] Näistä voidaan myös päätellä, että immersion ja äänten yhteys voi olla opittu ja käytännönläheinen. Se mikä on immersiivistä on löydetty testaamalla, tutkimalla käyttäjäpalautetta ja käyttäen eri osapuolien ammattitaitoja.

On kuitenkin olemassa joitain teorioita siitä, miten peliäänet vaikuttavat immersioon. Yksi teoria on, että äänet muistuttavat oikeasta maailmasta, jonka takia ne tuovat immersiota. Kaikki fysiikkaan pohjautuvat tilanteet ja esineet luovat yleensä jotain ääniä meidän maailmassamme. Ääni yhdistyy ihmisen mielessä muiden aistien kanssa, ja vahvistaa ja voimistaa niitä tuoden todenmukaisuutta. Niinpä äänet ovat tärkeässä osassa jokapäiväistä elämäämme ja todiste siitä, että olemme oikeassa maailmassa. Niiden puuttuminen voi tehdä maailmasta epätodellisen. Tämän takia äänet ovat tärkeä keino, minkä avulla virtuaalinen maailma voidaan saada todentuntuisemmaksi. [19, s. 23; 19, s. 32.] Ihmiset havittelevat nykyään enemmän ja enemmän realistisempia pelejä. Vaikka pelikehityksen aikana keskitytään monesti visuaaliseen puoleen, myös realistisempia peliääniä tarvitaan, jotta pelit olisivat immersiivisempiä. [10, s. 22–23.]

Toinen teoria on ulkopuolisten äänten peittäminen. Jotta pelaaja voisi syventyä pelimaailmaan helpommin, ulkopuolisen maailman ärsykkeet täytyy saada pois. Peliäänillä voidaan elävöittää maailmaa, mutta myös peittää ulkopuolisia ääniä häiritsemästä pelikokemusta. [3, s. 5.]

3.3 Viihdearvo

Pelien tarkoitus on olla viihteellisiä. Tämän takia peliäänien tarkoitus on myös tuoda pelaajille viihdearvoa. Peliäänet voivat koostua oikean maailman äänistä, mutta niiden eivät tarvitse olla välttämättä realistisia. Peliäänien täytyy olla miellyttäviä ja maailmaan sopivia. Äänisuunnittelijan ideana on luoda äänellinen maailma, joka kuulostaa pelaajista miellyttävältä, on tarpeeksi uskottava ja luo tunteita. Fantasia-maailmoissa on paljon olioita ja ilmiöitä, jotka tekevät ääniä, joita ei ole oikeassa maailmassa. Tämän takia äänisuunnittelijan täytyy käyttää paljon luovuutta ja rakentaa äänet itse yhdistelemällä ja muokkaamalla erilaisia ääniä. Äänisuunnittelijan täytyy miettiä, minkälaisia ääniä pelimaailmassa pitäisi olla ja mitä ääniä pelaaja odottaa kuulevansa pelimaailmassa. [3, s. 6; 3, s. 78; 3, s. 101–103.]

Äänillä ja musiikilla on kyky myös tuoda ihmisille tunteita. Yksi peliäänien ja varsinkin pelimusiikin tarkoitus on luoda tunnelmaa ja parantaa tämän kautta pelin kokemusta, viihdearvoa ja immersiota. [3, s. 71; 3, s. 130–131; 19, s. 33.]

4 Peliäänityypit

Peleissä on paljon erilaisia ääniä ja niitä voidaan kategorisoida monin eri tavoin. Peliäänien kategorisointi auttaa ymmärtämään, mistä kaikista äänistä peliäänet koostuvat. Tässä kappaleessa käydään läpi kaksi eri tapaa kategorisoida ääniä, jotka ovat oleellisimpia työn kannalta.

4.1 Kategoriat tuotannon näkökulmasta

Yleisin tapa kategorisoida peliääniä on tuotannon näkökulmasta. Tähän kuuluu yleensä kolme kategoriaa: puhe tai dialogi, ääniefektit ja musiikki. Tätä kategorisointia on perinteisesti käytetty elokuvien ja pelien tekemisessä ja se jakaa äänet selkeisiin äänentuotannon osa-alueisiin. Puhetta äänitetään ääninäyttelijöiden kanssa, ääniefektejä tekevät äänisuunnittelijat ja musiikkia tekevät artistit tai orkesterit. Jokainen osa-alue tuottaa jotain äänimateriaalia ja lopulta tiedostoja, joita käytetään pelissä. [18, s. 15; 19, s. 31.]

Näitä kolmea kategoriaa voidaan myös jakaa pienempiin alakategorioihin tarpeen mukaan. Esimerkiksi yksi tapa on erottaa ääniefekteistä taustaäänet (engl. ambient sounds) erilliseksi kategoriaksi. Tämä mukailee enemmän sitä, miten ääniä toistetaan pelissä. Taustaäänet toistetaan omina ääninä pelin taustalla, kun taas ääniefektit ovat useasti dynaamisia/reaktiivisia ääniä eli reagoivat ympäröivään maailmaan ja esimerkiksi pelaajan toimintoihin. [18, s. 15–16; 19, s. 31.]

Vaikka tätä kategorisointia käytetään laajasti, se ei välttämättä toimi kaikkiin tilanteisiin. Kun ääniä kategorisoidaan tuotannon näkökulmasta, kategorioilla on hyvin pieni yhteys siihen, miten ääniä käytetään loppujen lopuksi pelissä. Kategoriat eivät kuvaa sitä, miten äänet toimivat pelissä, vaan keskittyvät enemmän äänien alkuperään. Esimerkiksi pelin puheosuudet voisivat olla myös osa ääniefektejä tai musiikkia. Tämä voi haitata joissain tilanteissa peliäänten rakennetta ja toimivuutta sekä hankaloittaa johdonmukaista äänten kategorisointia. [18, s. 15.]

4.2 Kategoriat interaktiivisuuden näkökulmasta

Ääniä voidaan myös kategorisoida interaktiivisuuden mukaan, eli kuinka paljon pelaajan vuorovaikutus pelin kanssa vaikuttaa niihin. Tällöin ääniä voidaan jakaa lineaarisiin, adaptiivisiin ja interaktiivisiin ääniin.

Peliäänet ovat yleensä dynaamisia eli ne reagoivat muutoksiin pelissä. Se, mihin äänet reagoivat, määrittää, ovatko ne adaptiivisia vai interaktiivisia ääniä. Interaktiiviset äänet reagoivat pelaajan tekemiin toimintoihin. Esimerkiksi jos pelaaja painaa ohjaimen painiketta, pelin hahmo reagoi pelaajan toimintoon hyppäämällä ja toistamalla hyppyäänen. Hyppyääni olisi tässä tapauksessa interaktiivinen ääni. Adaptiiviset äänet taas reagoivat pelissä tapahtuviin tilanteisiin, jotka eivät ole välttämättä pelaajan käynnistämiä. Esimerkiksi jos pelissä alkaa loppumaan aika, pelin musiikki nopeutuu, jotta pelaaja tajuaa kiirehtiä. Tässä tapauksessa musiikki olisi adaptiivinen ääni. [20, s. 265–266.]

Lineaariset äänet, eli äänet, jotka eivät ole dynaamisia, eivät reagoi pelissä tapahtuviin muutoksiin tai pelaajan toimintoihin. Ne ovat samanlaisia jokaisella kerralla, eivätkä muutu pelin tilanteen tai hetken mukaan. Esimerkiksi pelin välianimaatioissa (engl. cutscene) voi olla taustalla musiikkia ja äänitehosteita, jotka toistetaan aina samalla tavalla pelikerrasta tai hetkestä riippumatta. Välianimaatio ei myöskään yleensä reagoi pelaajan toimintoihin. [20, s. 267.]

5 Peliäänten toistaminen

Peliäänten tarkoitus -kappaleessa käytiin läpi, kuinka äänen ja sitä vastaavan pelaajan tekemän toiminnon toistettavuus on yksi tärkeimmistä syistä, miksi peliäänet toimivat hyvänä äänipalautteena. Vaikka toistettavuus on tärkeä ominaisuus, sillä on myös haittapuoli: jos samaa ääntä toistetaan useasti, äänestä voi tulla ajan mittaan rasittava ja uuvuttava pelaajalle sekä se voi olla liian ennalta arvattava. Jotta tätä voitaisiin välttää, ääniin tarvitaan variaatiota. [17, s. 33.]

Ääniin voidaan lisätä variaatiota monilla eri tavoilla. Yksi keino on tehdä monia samankaltaisia ääniä ja toistaa niitä pelissä satunnaisessa järjestyksessä. Variaation luominen tapahtuu tällöin ennen äänien tuomista peliin, kun äänisuunnittelija luo monia variaatioita jostain tietystä äänestä tai nauhoittaa monta samankaltaista ääntä. Äänistä tehdään tiedostoja ja samankaltaiset äänitiedostot jaetaan omiin kansioihinsa. Sen sijaan, että peli toistaisi vain jonkun tietyn äänen joka kerta, peli valitseekin satunnaisesti yhden äänen kansiosta ja toistaa sen. [17, s. 33–34.]

Ääniefekteille, jotka ovat kolmiulotteisessa maailmassa, äänenvoimakkuuden ja panoroinnin (engl. panning) muuttaminen ovat tärkeitä tapoja kertoa äänen etäisyydestä ja suunnasta. Panorointi tarkoittaa sitä, kuinka voimakkaasti jokin ääni kuuluu kahden (stereo) tai useamman (surround) kaiuttimen välillä. Esimerkiksi jos stereojärjestelmässä yhdestä kaiuttimesta toistetaan voimakkaammin ääntä kuin toisesta, ääni kuulostaa siltä kuin se tulisi enemmän siitä suunnasta. Näin voidaan esittää äänen fyysistä paikkaa. Varsinkin ensimmäisestä persoonasta pelattavat pelit käyttävät paljon hyödyksi äänenvoimakkuutta ja panorointia. [3, s. 250–252.]

Pelimusiikissa variaatiota voidaan saada aikaiseksi muuttamalla ja vaihtelemalla pelin aikana esimerkiksi tempoa, sävelkorkeutta, rytmiä, äänenvoimakkuutta, melodiaa, harmoniaa ja musiikillista rakennetta [2, s. 147]. Variaatiota ja dynaamisuutta saadaan myös muuttamalla kappaleen miksausta eli jokaisen instrumentin tai äänen äänenvoimakkuutta pelin aikana eri tilanteissa [2, s. 152]. Musiikkeihin voidaan lisätä myös DSP-efektejä, joita ovat esimerkiksi erilaiset kaikuefektit. [2, s. 149.]

Pelimusiikin variaatiotavat ovat myös suurimmaksi osin suoraan johdettavissa peliääniefekteihin. Ääniefekteissä voidaan muokata nopeutta (tempo), äänenkorkeutta (sävelkorkeus) ja ajoitusta (rytmi). Ääniefekteihin voidaan myös lisätä samalla tavalla DSP-efektejä. Kun tähän lisätään äänenvoimakkuuden, panoroinnin sekä saman kaltaisten äänien satunnaisessa järjestyksessä toistaminen, saadaan ääniefekteihin paljon variaatiota ja luonnollisuutta.

6 Ääniliitännäisen toteutus

Työn tavoitteena on toteuttaa liitännäinen, joka lisää Cleverin pelien fysiikkapohjaisiin esineisiin törmäys-, raahaus- ja vierimisäänet. Ajatuksena on antaa Cleverille valmis ja toimiva kokonaisuus, jota he pystyvät käyttämään ja lisäämään mihin tahansa peliprojektiin helposti. Työtä ei jatketa mistään olemassa olevasta työstä, vaan se aloitetaan alusta asti.

Työn aihe valittiin, koska liitännäinen lisäisi Cleverin VR/AR-peleihin ja -sovelluksiin enemmän immersiota, ja parantaisi niiden käyttäjäkokemusta antamalla käyttäjille välitöntä äänipalautetta esineiden osumista. Työ helpottaa myös käyttäjiä esineen materiaalin tunnistamisessa. Kehittäjille työ antaa yhtenäisen toimintavan fyysisten äänten toistamiselle sekä nopeuttaa äänien lisäämistä peliin.

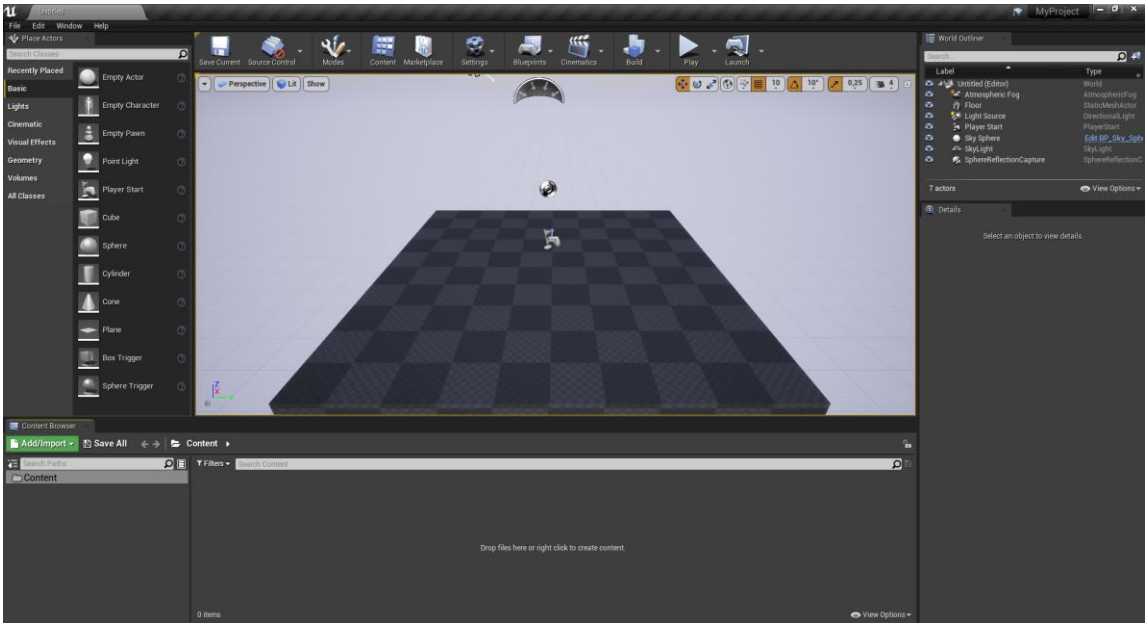
Työssä on tarkoituksena käyttää valmiiksi äänitettyjä eri materiaalien ääniä ja toistaa niitä mahdollisimman luontevasti ja fysiikoiden mukaisesti pelissä. Liitännäiseen lisätään valmiiksi muutamalle perusmateriaalille (esim. puu, metalli ja muovi) äänet, joita Clever voi käyttää projekteissaan. Liitännäinen on tehty myös niin, että Clever voi lisätä myöhemmin ääniä muille materiaaleille.

Työn lopuksi liitännäisestä palautetaan myös Cleverille dokumentaatio, joka pohjautuu suurimmaksi osin tähän käytännönosuuteen.

6.1 Unreal Engine

Työ toteutettiin Unreal Engine -pelimoottorilla, jota Clever käyttää monissa projekteissaan. Unreal Engine (UE) on Epic Games -yrityksen kehittämä kehitysympäristö ja pelimoottori, jota monet tunnetut pelit sekä myös elokuva-, televisio-, arkkitehtuuri-, simulointi- ja tuoteteollisuudenalat käyttävät [21; 22]. UE:sta on monta eri versiota, mutta tässä projektissa käytettiin versiota 4.26. Koska liitännäisten kehitys oli uutta, työn aikana tutkittiin ja opeteltiin myös, miten UE-liitännäisiä kehitetään. Tätä varten työn kehityksessä hyödynnettiin UE:n dokumentaatiota sekä verkosta löytyviä esimerkkejä ja oppaita.

Pelilogiikkaa ja toiminnallisuuksia voidaan kehittää UE:ssa C++- ja Blueprint-ohjelmointikielillä. Blueprint on UE:n sisällä toimiva visuaalinen ohjelmointikieli, jossa ohjelmointi tapahtuu yhdistelemällä erilaisia toiminnallisia palikoita, jotka suoritetaan sarjassa. Blueprintin ajatuksena on olla yksinkertaisempi ohjelmointikieli kuin C++, millä voidaan tehdä helposti ja nopeasti pelilogiikkaa. Vaikka Blueprint on suorituskyvyltään hieman hitaampi ja ohjelmoinnin kannalta rajatumpi kuin C++, sitä voidaan silti käyttää kokonaisten UE-peliprojektien kehittämiseen. Blueprintin helppous mahdollistaa esimerkiksi sen, että kenttäsuunnittelijat ja artistit voivat ohjelmoida toiminnallisuuksia peliin ilman, että he tarvitsevat isompaa ohjelmointiosaamista. Näin muutkin kuin ohjelmoijat voivat lisätä peliin helposti toiminnallisuuksia. Tämä mahdollistaa myös sen, että ohjelmoijat voivat tehdä toiminnallisia elementtejä Blueprintteihin käyttäen tehokkaampaa ja monimutkaisempaa C++-kieltä, joita muut voivat käyttää helposti Blueprintin kautta. Näin voidaan tasoittaa projektin ohjelmallista työtä tasaisemmin eri osapuolille ja parantaa työnkulkua. [23; 24; 25.] Kuvassa 9 on esitetty UE:n käyttöliittymä, jossa voidaan muokata esimerkiksi peliprojektin pelikenttiä, tiedostoja ja asetuksia sekä ohjelmoida Blueprint-kielellä [26].



Kuva 9. Kuvankaappaus UE:n käyttöliittymästä.

6.2 Liitännäisen rakenne

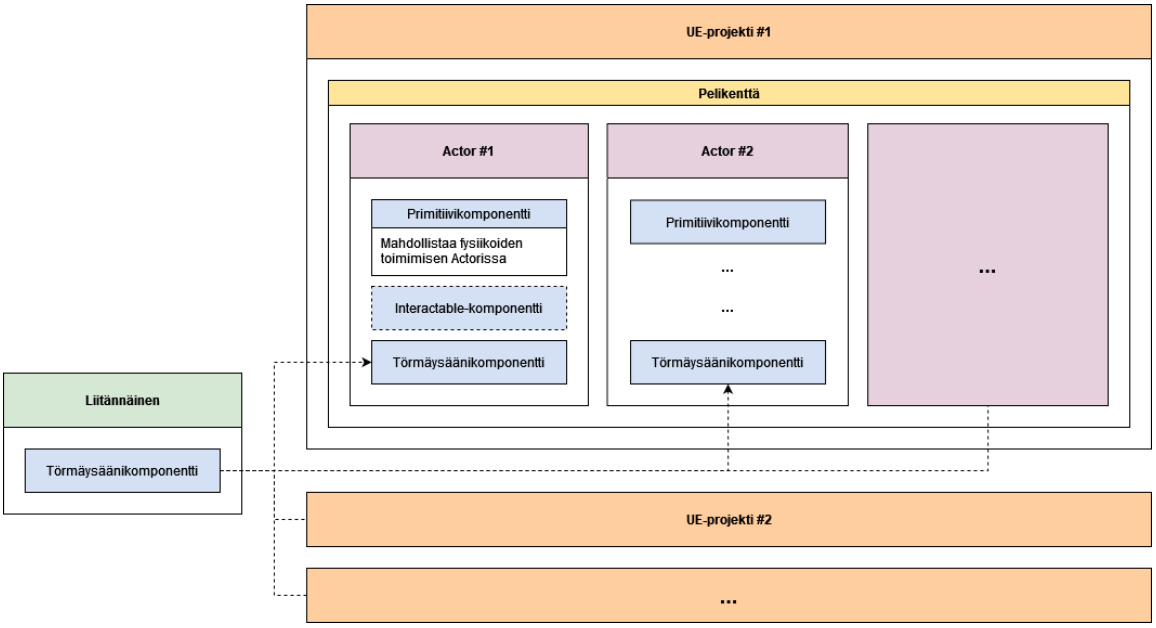
Liitännäinen (engl. plug-in) on tietokoneohjelma tai moduuli, jolla voidaan lisätä johonkin isompaan ohjelmaan nopeutta tai toimintoja [27; 28]. UE:ssa liitännäiset ovat koodia ja dataa, joilla

voidaan laajentaa pelimoottorin ominaisuuksia tai lisätä peleihin toiminnallisuuksia. UE:n liitännäisiä voidaan lisätä tai poistaa milloin vain projektikohtaisesti. [29.] Koska samaa liitännäistä voidaan lisätä helposti moneen eri projektiin, liitännäiset mahdollistavat järkevän yhteisen toiminnallisuuskokonaisuuksien jakamisen projektien välillä. Myös valmiiksi tehdyillä liitännäisillä voidaan säästää aikaa pelin kehityksessä. Koska työn äänitoiminnallisuutta tullaan käyttämään monessa projektissa, työn tekeminen liitännäisenä vaikutti parhaalta ratkaisulta.

Jokainen UE-projekti koostuu pelimaailman palasista eli pelikentistä (engl. level), joissa pelaaja voi liikkua ja olla vuorovaikutuksessa. Jokainen pelikenttä koostuu esimerkiksi erilaisista esineistä, muodoista ja asetuksista eli asioista, joita kenttään voi lisätä. [30.] Näitä asioita, joita voi lisätä pelikenttään kutsutaan UE:ssa Actoreiksi. Actorit koostuvat erilaisista komponenteista (engl. component), jotka lisäävät Actoriin toiminnallisuuksia. Esimerkiksi yksi komponentti voi lisätä Actorille visuaalisen ulkomuoto, kun taas toinen komponentti määrää, miten Actor liikkuu pelimaailmassa. [31; 32.]

Blueprintillä ohjelmoituja toiminnallisuuksia suoritetaan yleensä tapahtumien eli eventtien kautta. Eventit voivat olla esimerkiksi, että peli alkaa, pelaaja saa osuman tai esine törmää johonkin. Pelimoottori kutsuu näitä eventtejä aina, kun kyseinen tapahtuma tapahtuu ja suorittaa toiminnallisuudet, jotka käyttäjä on ohjelmoinut eventin suoritettavaksi. [33.] Käyttäjä voi tehdä myös omia eventtejä eli Custom Eventtejä. Kun Custom Eventtejä halutaan suorittaa, niitä täytyy joko kutsua erikseen jossain pelin aikana tai sitoa (engl. bind) johonkin toiseen eventtiin. Eventteihin voidaan sitoa monia Custom Eventtejä. Kun eventtiä kutsutaan, se suorittaa kaikki siihen sidotut Custom Eventit. [34.]

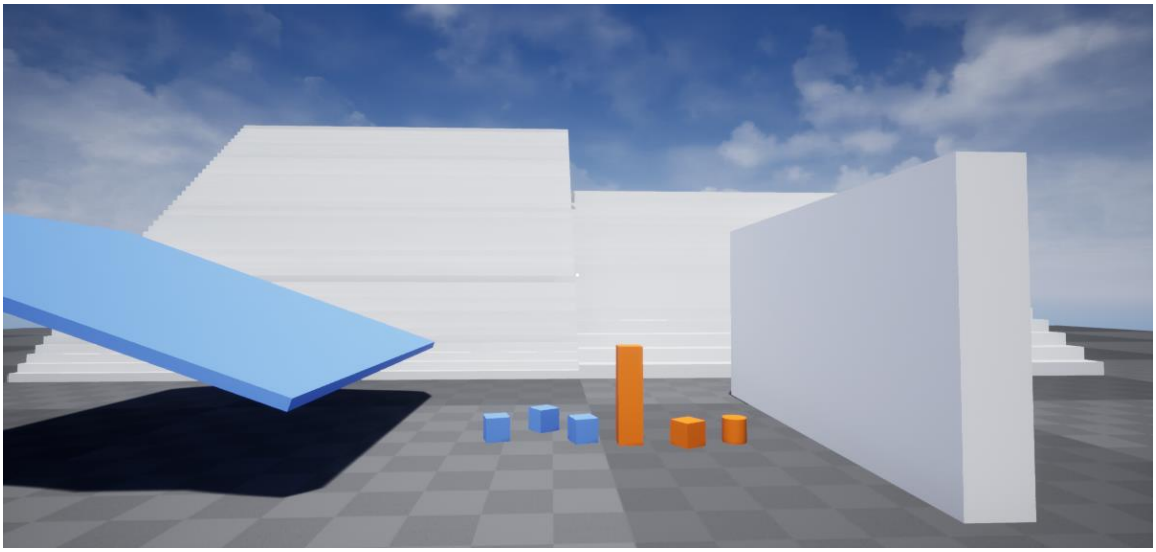
Työn liitännäinen lisää UE-peliprojekteille uuden törmäysäänikomponentin, jonka voi lisätä mihin tahansa fysiikoilla toimivaan Actoriin. Kun komponentti lisätään Actoriin, siihen tulee törmäysäänet. Jotta törmäysäänikomponentti voisi toimia, Actor tarvitsee fysiikat, jotka UE:n primitiivikomponentti tarjoaa. Primitiivikomponentti sisältää tietoja Actorin visuaalisesta ulkomuodosta ja fyysisistä ominaisuuksista sekä tarjoaa näihin liittyviä eventtejä [32; 35]. Kuvassa 10 näkyy liitännäisen toiminnallinen rakenne.



Kuva 10. Liitännäisen toiminnallinen rakenne.

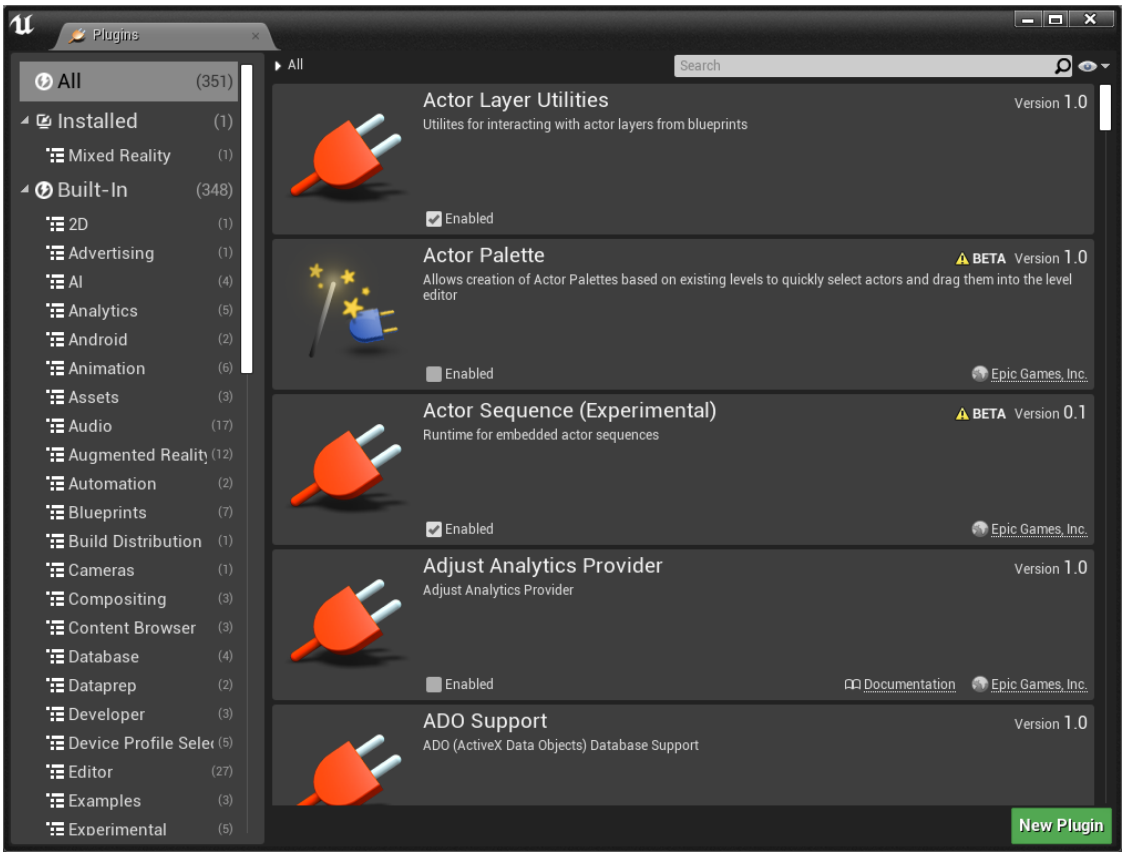
6.3 Liitännäisen lisääminen peliprojektiin

Työssä tehtiin liitännäistä varten uusi UE-peliprojekti, jossa voitiin testata liitännäisen toimivuutta kehityksen aikana. Peliprojekti on myös hyödyllinen tulevaa varten, jos liitännäistä halutaan jatkokehittää. Kuvassa 11 näkyy kuvankaappaus peliprojektin testikentästä, jossa on pelaaja, lattia, kaksi portaikkoa, vino taso ja muutamia erilaisia fysiikkapohjaisia esimerkkiesineitä, joita pelaaja voi liikuttaa, siirtää ja tiputtaa. Peliprojektin ja liitännäisen koodit ovat erillään toisistaan. Liitännäisen koodi on modulaarinen eli se ei ole riippuvainen mistään peliprojektista ja sen voi helposti lisätä tai poistaa projekteista.



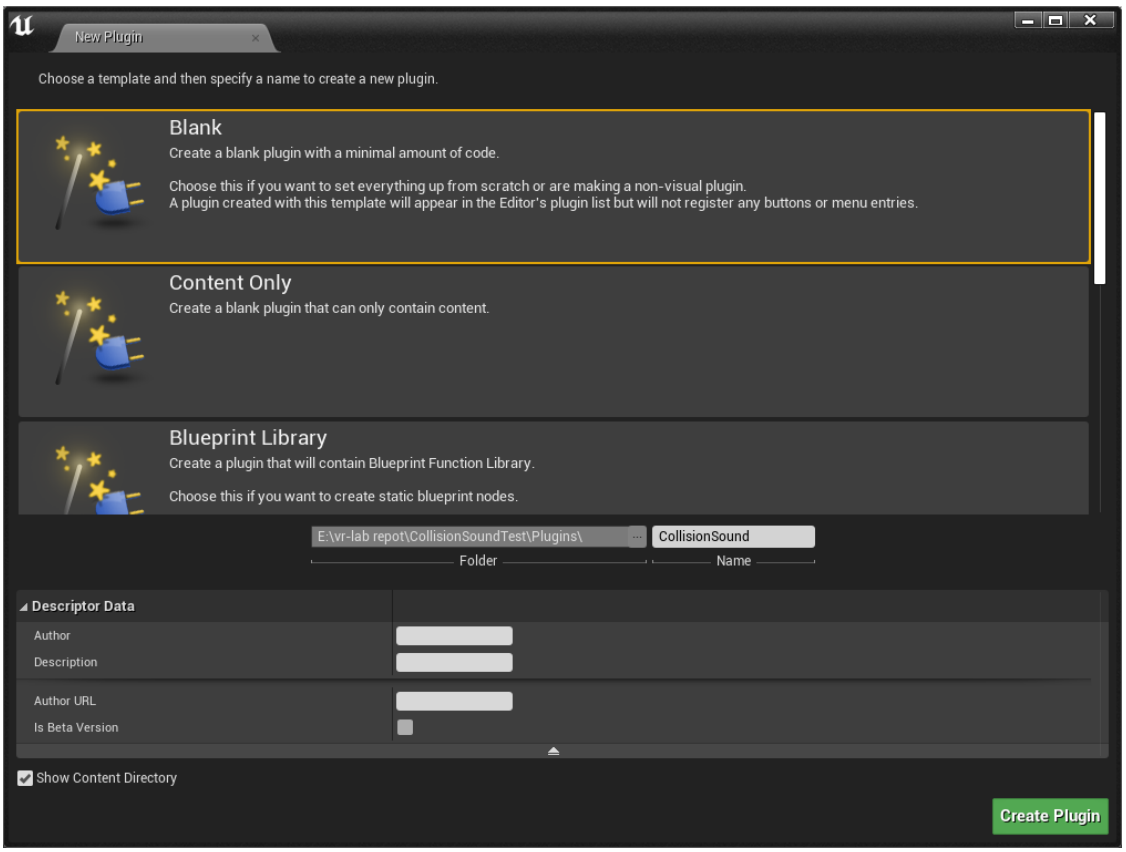
Kuva 11. Kuvankaappaus peliprojektin testikentästä.

Uusi liitännäinen voidaan luoda UE:n Plugin Browser -ikkunasta, joka on liitännäisille tarkoitettu selain. Selaimesta löytyy kaikki liitännäiset, joita voidaan lisätä tai poistaa projektista. Kyseisessä ikkunassa on myös New Plugin -nappi oikeassa alakulmassa, josta voidaan luoda uusi liitännäinen avattuun projektiin. [29.] Kuvassa 12 on UE:n Plugin Browser.



Kuva 12. Unreal Enginen Plugin Browser.

Painikkeen painaminen avaa uuden ikkunan, jossa määritetään uuden liitännäisen nimi ja tiedostosijainti projektissa sekä muita liitännäisen kuvaustietoja. Tästä ikkunasta voidaan myös määrittää, minkä mallin (engl. template) mukaan liitännäinen luodaan. [29.] Työn liitännäinen aloitettiin tyhjästä mallista. Kun tiedot on määritetty ja Create Plugin -nappia on painettu, UE lisää uuden liitännäisen projektiin. Kuvassa 13 on ikkuna, jossa voidaan määrittää uuden liitännäisen tiedot.



Kuva 13. Ikkuna liitännäisen määrittelemiselle.

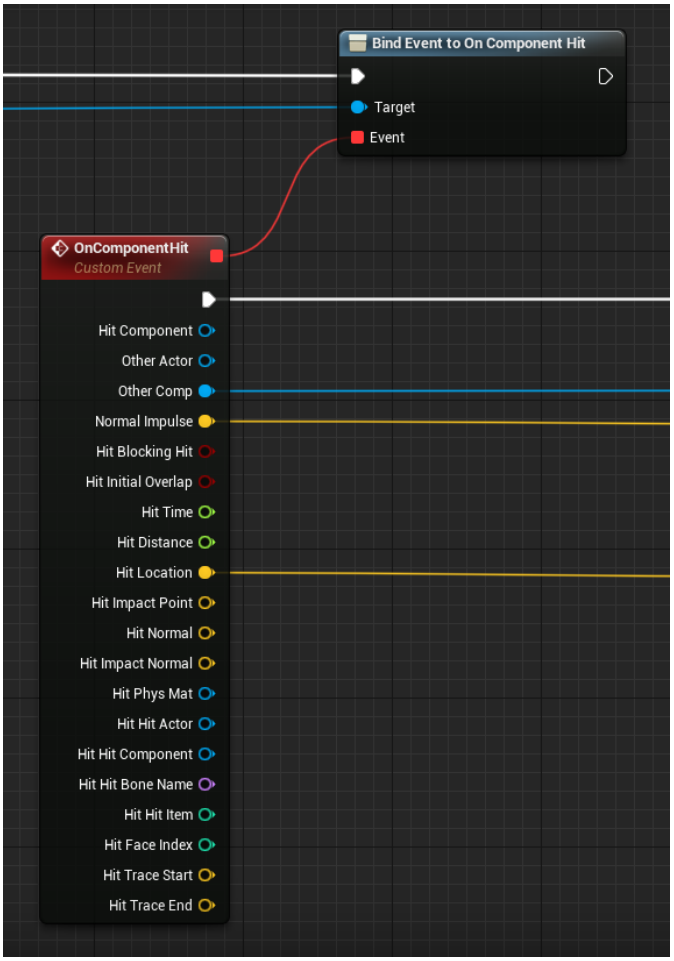
Liitännäiseen voidaan lisätä nyt uusi komponentti, joka käyttää joko Blueprint- tai C++-kieltä. Blueprint-komponentti luodaan käyttämällä UE:n Content Browser -aluetta, jossa on kaikki projektiin liittyvät tiedostot. Content Browserissa on ”Add/Import” -painike, jonka kautta voidaan lisätä Blueprint-tiedosto ja valita Blueprintin tyypiksi komponentti. C++-komponentti luodaan UE:n tiedostovalikon kautta, jossa on New C++ Class -valinta. Tämä avaa uuden ikkunan, jossa voidaan valita C++-tiedoston tyypiksi komponentti.

Kun liitännäinen ja sen komponentti on lisätty ja valmisteltu peliprojektia varten, liitännäisen kehitys voidaan aloittaa. Työn liitännäistä aloitettiin aluksi kehittämään C++:lla, mutta se muutettiin loppujen lopuksi täysin Blueprint-pohjaiseksi. Syynä tähän oli, että liitännäisen kehittäminen ja

ohjelmallinen kääntäminen C++:lla vei paljon enemmän aikaa kuin Blueprinteillä. Blueprint kielenä oli myös nopeampi ja helpompi käyttää. Liitännäinen on myös vaivattomampi lisätä projekteihin, koska Clever käyttää jo monissa projekteissaan pelkästään Blueprint-kieltä.

6.4 Tömäysäänien kehitys

Jotta törmäysääniä voidaan toistaa, tarvitaan tieto, missä pisteessä ja millä voimalla esimerkiksi pelaajan heittämä laatikko törmää pelikentässä olevaan seinään. Liitännäisen rakenne -kappaleessa käytiin läpi, kuinka kaikki kentässä olevat asiat ovat Actoreita, ja kaikki fysiikoilla toimivat Actorit tarvitsevat primitiivikomponentin. UE tarjoaa primitiivikomponentin kautta On Component Hit -fysiikka-eventin, joka tapahtuu aina kun primitiivikomponentin sisältämä Actor törmää toiseen Actoriin [36]. UE tarkistaa fysiikoita eli tekee fysiikkamoottorin päivityksiä (engl. tick/update) monia kertoja sekunnissa ja kutsuu On Component Hit -eventtiä sillä päivityksellä, kun Actorit törmäävät toisiinsa [36; 37; 38]. Kun On Component Hit -event tapahtuu, se antaa törmäyksestä tietoja, joista yhdet ovat törmäyspiste ja -voima. Törmäyspiste on kohta pelimaailmassa, jossa törmäysääni kuuluu, ja törmäysvoima on kuinka isolla voimalla Actor puskeutuu toiseen Actoriin. Törmäysvoimasta voidaan laskea, kuinka voimakkaasti äänen täytyy kuulua. Kuvassa 14 näkyy, kun törmäävän Actorin On Component Hit -eventtiin on sidottu samanniminen Custom Event, josta saadaan törmäyspiste eli Hit Location ja törmäysvoima eli Normal Impulse.



Kuva 14. On Component Hit -event ja siitä saatavat törmäyksen tiedot.

UE:ssa äänenvoimakkuus on esitetty desimaaliluvulla, jonka pienin arvo eli minimiarvo on 0,0 ja suurin arvo eli maksimiarvo on 1,0. Arvoa voidaan ajatella prosenttina, kuinka voimakkaasti ääni kuuluu, jossa 0,0 on 0 % ja 1,0 on 100 %. Jotta törmäysvoima voidaan muuttaa äänenvoimakkuudeksi, tulee määrittää ensin äänenvoimakkuuden minimi- ja maksimiarvoja vastaavat törmäysvoimat. Toisin sanoen: täytyy tietää pienin törmäysvoima, jolloin ääni toistetaan pienimmällä äänenvoimakuudella sekä suurin törmäysvoima, jolloin ääni toistetaan suurimmalla äänenvoimakkuudella. Nämä minimi- ja maksimitörmäysvoimat saadaan testaamalla ja kuuntelemalla, mikä tuntuu ja kuulostaa oikealta. Minimi- ja maksimitörmäysvoimille on Min Hit Impulse - ja Max Hit Impulse -muuttujat, joita liitännäisen käyttäjä pystyy muokkaamaan mieleiseksi jokaista törmäyskomponenttia kohden. Muuttujille on asetettu myös yleisesti toimivat oletusarvot.

Törmäysvoiman ja äänenvoimakkuuden minimi- ja maksimiarvot luovat rajat kahdelle numerovälille. Ajatuksena on muuttaa On Component Hit -eventin antama törmäysvoima näiden numerovälien välillä lineaarisesti eli samassa suhteessa. Jos törmäysvoima suurenee, myös äänenvoimakkuus suurenee samassa suhteessa. Jotta äänenvoimakkuus saadaan, täytyy laskea numerovälien

välinen suhde ja kertoa se törmäysvoimalla. Suhde saadaan erottamalla kummakin numerovälin suurin ja pienin arvo ja jakamalla äänenvoimakkuuden erotus törmäysvoiman erotuksella.

Jos esimerkiksi minimitörmäysvoima on 20 ja maksimitörmäysvoima 2000, törmäysvoiman ja äänenvoimakkuuden suhde on tällöin

$$\frac{1{,}0-0{,}0}{2000-20} = \frac{1}{1980}.$$

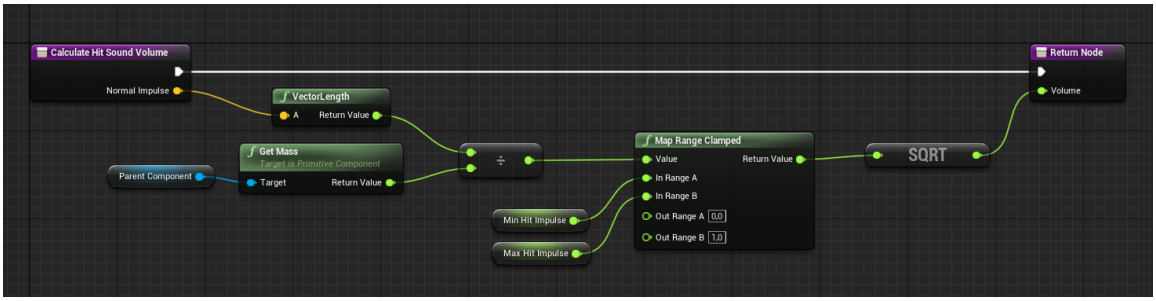
Jos törmäysvoima on esimerkiksi 900, kertomalla suhde törmäysvoimalla saadaan

$$\frac{1}{1980} \cdot 900 = 0{,}4545 \ldots \approx 0{,}45.$$

Ääni toistuisi tällöin noin 45 % voimakkuudella.

UE:ssa arvon muuntaminen lineaarisesti kahden numerovälin välillä tapahtuu Map Range Clamped -funktiolla. Funktio ottaa vastaan parametreina kummankin numerovälin minimi- ja maksimiarvot sekä muutettavan arvon ja palauttaa muutetun arvon funktiosta. Funktio myös varmistaa, ettei arvo mene alle minimiarvon tai yli maksimiarvon. Tätä kutsutaan clamppaamiseksi, joka on myös funktion nimessä. [39.]

Kun törmäysvoima on muutettu Map Range Clamped -funktiolla äänenvoimakkuudeksi hyödyntäen minimi- ja maksimitörmäysvoiman arvoja, lasketaan lopuksi äänenvoimakkuuden neliöjuuri. Kun törmäysääniä testattiin laatikoilla, pehmeämmät törmäykset olivat todella hiljaisia ja äänet voimistuivat kiihtyvästi, mitä kovemmin laatikko törmäsi seinään. Äänet vaikuttivat kovenevan eksponentiaalisesti tai logaritmisesti. Kun äänenvoimakkuudesta otettiin neliöjuuri, ongelma saatiin kumottua ja äänet kovenivat tasaisemmin. Tarkkaa syytä ilmiölle ei löydetty työn aikana. Yksi mahdollinen syy voi olla, että UE käyttää luonnollisempaa logaritmista äänenvoimakkuuden säätämistä, joka voi luoda käänteisen ilmiön törmäysvoiman ja äänenvoimakkuuden muuntamisessa. Kuvassa 15 on Calculate Hit Sound Volume -funktio, joka muuttaa Normal Impulse -parametrista saadun törmäysvoiman äänenvoimakkuudeksi ja palauttaa äänenvoimakkuuden funktiosta. Funktion alussa törmäysvoimasta jaetaan törmäävän Actorin massa pois, jotta massa ei vaikuttaisi törmäysääniin.



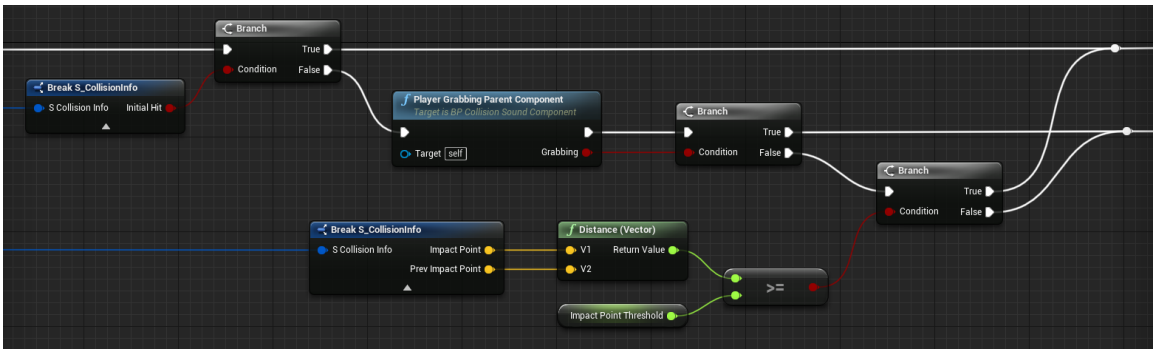
Kuva 15. Calculate Hit Sound Volume -funktion toiminta.

Aina kun Actor törmää toiseen Actoriin, törmäysvoima muutetaan äänenvoimakkuudeksi ja ääni toistetaan törmäyspisteessä kyseisellä äänenvoimakkuudella. Ongelmaksi tulee kuitenkin, että törmäysääni kuuluu jokaisella fysiikkamoottorin päivityksellä, kun Actorit ovat kosketuksessa toisiinsa. Tässä työssä kosketuksella tarkoitetaan ajanjaksoa, jolloin Actorit ovat yhtäjaksoisesti monen fysiikkamoottorin päivityksen ajan törmäyksessä toisiinsa. Koska päivityksiä tapahtuu yleensä monia kymmeniä sekunnissa, myös ääniä kuuluu yhtä monta yhtäaikaisesti, joka kuulostaa rätinältä. Jotta tätä ei tapahtuisi, täytyy löytää ehtoja, joilla voidaan rajata ja toistaa vain oleellisimmilta ja luonnollisimmilta vaikuttavat törmäysäänet.

Aiemmin mainittu Map Range Clamped -funktio rajaa jo törmäysääniä minimitörmäysvoiman avulla. Määrittämällä minimitörmäysvoima sopivaksi voidaan rajata pois kaikki törmäysäänet kosketuksen aikana, jotka ovat alle minimitörmäysvoiman. Tämä vähentää jo paljon turhia törmäysääniä kuulumasta, joissa on pieniä törmäysvoimia. On kuitenkin tilanteita, jolloin törmäysvoimat ovat liian suuria koko kosketuksen ajan. Tätä voi tapahtua, jos Actor on vähän kuin paineen alla eli esimerkiksi, jos pelaaja työntää Actoria seinään tai se on puristuksissa kahden Actorin välissä.

Yksi tapa, miten törmäysääniä voidaan rajata näissä tilanteissa, on toistaa törmäysääni vain Actorin kosketuksen aloitushetkellä. Tämä tapa toimii hyvin, jos pelaaja pitää kädessä Actoria, koska Actorin liikkeet ja törmäykset ovat kädessä pitäessä kontrolloidummat. Myös kosketuksen aikana kuuluvat törmäysäänet eivät ole yleensä näissä tilanteessa yhtä tärkeitä. Kyseinen tapa ei kuitenkaan toimi yhtä hyvin, jos Actor liikkuu vapaasti fysiikoiden mukaan pelikentässä. Esimerkiksi, jos Actor tippuu maahan ja pyörii sitä pitkin irtautumatta siitä, monet törmäysäänet, jotka kuuluisivat kosketuksen aikana jäävät toistumatta tällä rajauksella. Tämän takia vapaasti liikkuvat Actorit tarvitsevat myös toisen ehdon: kosketuksen aikana olevia törmäysääniä toistetaan vain, jos törmäyspisteen paikka muuttuu tietyn etäisyyden verran. Toisin sanoen: kosketuksessa oleva Actor tarkistaa jokaisella päivityksellä, onko se tarpeeksi kaukana siitä, missä kuului viimeisin törmäysääni,

ja jos on, se toistaa seuraavan törmäysäänen. Tämä auttaa lisäämään kosketuksen aikana olevia törmäysääniä, vaikka joissain tilanteissa törmäysääniä voi kuulua liikaa ja törmäyksiin tulee taas rätinää. Tämän takia etäisyyden kynnysarvolle (engl. threshold), eli arvolle, jonka ylittäessä kuuluu törmäysääni, on tehty Impact Point Threshold -muuttuja, jolla voidaan tasapainottaa, kuinka herkästi törmäysääniä halutaan kuuluvan kosketuksen aikana. Liitännäisen käyttäjä voi säätää muuttujaa haluamansa mukaan. Tähänkin muuttujaan on asetettu yleisesti toimiva oletusarvo. Kuvassa 16 on ehtojen toiminnallisuus, jossa jokainen yllä mainittu ehto tapahtuu yhdessä Blueprintin Branchissä. Ensimmäinen ehto tarkistaa, jos on kosketuksen aloitushetki, toinen ehto tarkistaa, jos pelaaja pitää Actoria kädessä ja viimeinen ehto tarkistaa törmäyspisteen etäisyyden.



Kuva 16. Törmäysäänien rajausten ehdot.

Peliäänten toistaminen -kappaleessa käytiin läpi eri tapoja varioida ääniä, joista yksi oli satunnaisten äänten valitseminen. Jotta törmäysääniin saatiin lisää luonnollisuutta, tässä työssä käytettiin muutamaa törmäysäänitiedostoa, joita toistettiin satunnaisessa järjestyksessä. Liitännäisessä on kaksi eri tapaa toistaa ääniä: täysin satunnaisessa tai sekoitetussa (engl. shuffle) järjestyksessä. Ero näissä on, että sekoitettu järjestys toistaa kaikkia ääniä tasapuolisesti yhden kerran, eikä toista samaa ääntä kahta kertaa. Täysin satunnainen järjestys valitsee aina ensimmäisen satunnaisen äänen ja toistaa sen, jonka takia joitain törmäysääniä toistetaan enemmän kuin toisia ja sama ääni voi toistua peräkkäin useasti. Kumpikin tapa voi toimia hyvin eri tilanteissa, mutta yleisesti sekoitettu järjestys on parempi, koska se luo enemmän variaatiota ääniin. Liitännäisessä on muuttuja, joka säätää kumpaa tapaa käytetään ja jota käyttäjä voi muokata haluamansa mukaan. Vaikka tässä työssä on käytetty viittä törmäysääntä, liitännäiseen voi lisätä niin monta törmäysääntä kuin haluaa.

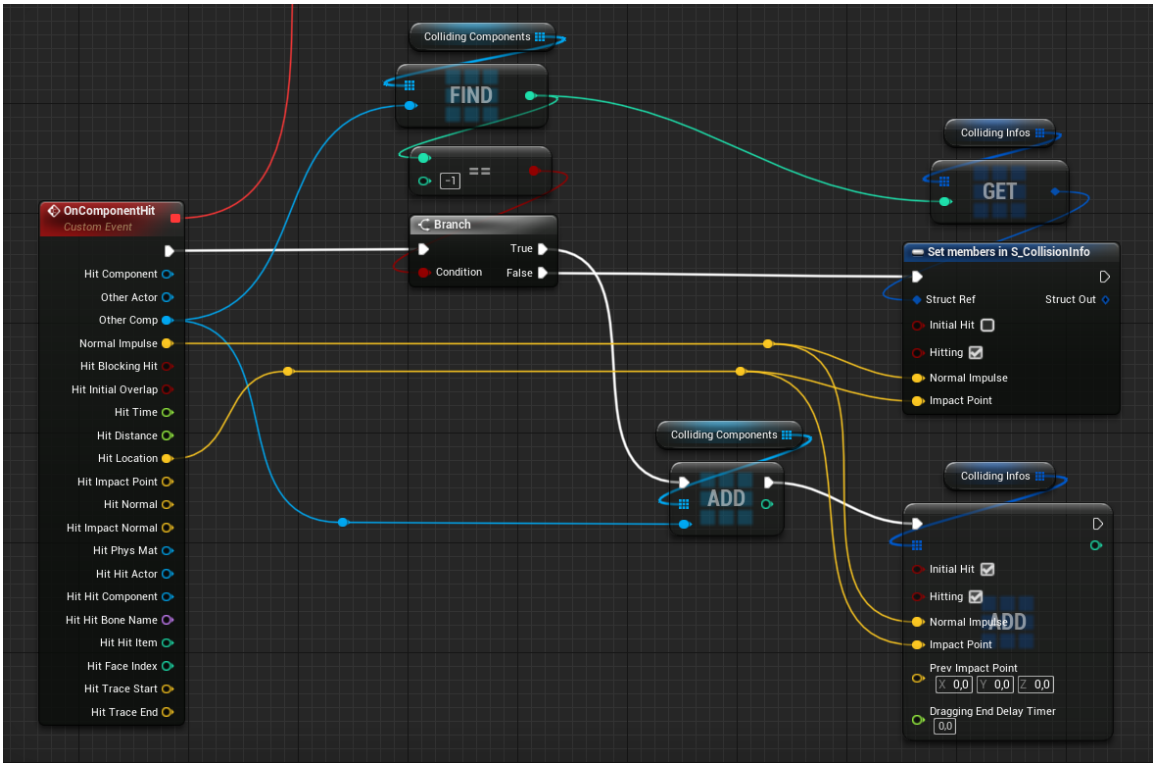
Törmäysääniin luodaan myös variaatiota äänenkorkeuden vaihteluilla. UE:ssa äänenkorkeutta voidaan säätää desimaalikertoimen avulla. Kun kerroin on 1,0, se vastaa alkuperäistä äänenkorkeutta ja mitä isompi tai pienempi kerroin on sitä korkeammalla tai matalammalla äänenkorkeus

on. Aina kun törmäysääntä toistetaan, äänenkorkeuden kerroin valitaan satunnaisesti joltain lukuväliltä, jossa 1,0 on keskimmäinen arvo. Liitännäisessä on muuttuja, joka vastaa kumpaakin lukuvälin minimi- ja maksimiarvoa, jota liitännäisen käyttäjä voi säätää haluamansa mukaan. Tähänkin muuttujaan on asetettu yleisesti toimiva oletusarvo.

6.5 Raahausäänien kehitys

Jotta raahausääniä voidaan toistaa, tarvitaan tieto, milloin Actor aloittaa ja lopettaa kosketuksen toisen Actorin kanssa. Ongelmana on, että On Component Hit -eventistä ei saa tietoa siitä, milloin Actorit aloittavat tai lopettavat kosketuksen. UE:ssa on törmäysten lisäksi overlap-eventtejä. Overlap on hetki, jolloin kaksi Actoria ovat osittain tai kokonaan toistensa sisällä. UE tarjoaa overlapeille eventit, kun ne alkavat ja loppuvat. Overlap-eventit eivät kuitenkaan anna törmäyksestä tietoja, ja eivät välttämättä toimi, jos Actoreista on laitettu törmäykset päälle, koska ne eivät voi mennä törmätessään missään vaiheessa toistensa sisälle. Ne eivät myöskään päivity yhtä nopeasti ja tarkasti kuin On Component Hit -eventit. Näiden takia niitä ei myöskään käytetä liitännäisessä raahausäänten toistamiseen.

Tämän takia tarkistus siitä, milloin Actor aloittaa ja lopettaa kosketuksen, jouduttiin tekemään itse. Koska On Component Hit -event tapahtuu aina jokaisella päivityksellä, kun Actorit törmäävät toisiinsa, kaikista Actorien välisistä kosketuksista voidaan pitää listaa, jota päivitetään jokaisella päivityksellä. Jokaiselle Actorien väliselle kosketukselle luodaan tietue (engl. structure) eli ohjelmallinen tietokokonaisuus, joka sisältää tietoa kosketuksesta. Tietuessa on tietoja esimerkiksi siitä, onko kosketus alkanut kyseisellä päivityksellä, onko Actorit vielä kosketuksessa toisiinsa sekä missä kohdassa ja millä voimalla viimeisin törmäys tapahtui. Listaan lisätään ja siitä poistetaan aina tietoja sen mukaan, kun Actorit aloittavat tai lopettavat kosketuksen. Kuvassa 17 näkyy, kun aiemmin esitetty Custom Event, joka on sidottuna On Component Hit -eventtiin, lisää Collision Infos -listaan aina uuden tietuen kosketuksesta. Jos kosketus löytyy jo listasta, Custom Event päivittää tiedot viimeisimpiin tietoihin.

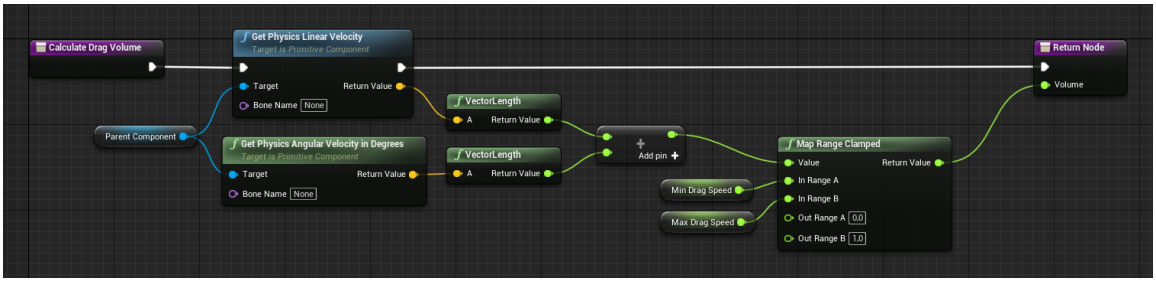


Kuva 17. Kosketuksiin liittyvien tietojen lisäys ja päivitys.

Kosketuksia käydään läpi UE:n Tick-eventissä, joka tapahtuu jokaisella pelimoottorin päivityksellä kaikkien On Component Hit -eventtien jälkeen. Tick-eventissä kosketusten tiedot käydään läpi, aiemmin mainitut törmäysäänten rajaukset ja laskut suoritetaan ja mahdolliset törmäysäänet toistetaan. Jotta voidaan tietää, koskevatko Actorit toisiaan vielä seuraavalla pelimoottorin päivityksellä, Tick-eventin lopussa jokainen kosketus merkataan loppuneeksi laittamalla tietuesta Hitting-muuttuja epätodeksi. Jos Actorit koskevat seuraavassa päivityksessä toisiaan, On Component Hit Custom Event laittaa Hitting-muuttujan takaisin todeksi ja ne pysyvät listassa normaalisti. Jos Actorit eivät koske enää toisiaan, Custom Event pitää Hitting-muuttujan epätotena ja tietue poistetaan listasta Tick-eventissä.

Raahausääniä kuuluu vain, jos esineet liikkuvat ja ovat kiinni toisissaan. Tämän takia raahausäänten äänenvoimakkuus määräytyy sen mukaan, millä nopeudella esine liikkuu ja pyörii kosketuksen aikana. Äänenvoimakkuus lasketaan samalla tavalla kuin törmäysäänissä käyttäen Map Range Clamped -funktiota. Erona on, että törmäysvoiman sijasta Map Range Clamped -funktiolle annetaan Actorin liikkeen ja pyörimisen nopeuden summa. Raahaukselle on minimi- ja maksimiraahausnopeudet, joidenka mukaan äänenvoimakkuutta nostetaan suhteessa. Minimi- ja maksimiraahausnopeuksille on Min Drag Speed ja Max Drag Speed -muuttujat, joita liitännäisen käyttäjä

voi säätää haluamansa mukaan. Muuttujille on asetettu myös yleisesti toimivat oletusarvot. Kuvassa 18 näkyy Calculate Drag Volume -funktio, joka muuttaa liikkeen ja pyörimisen nopeuden summan äänenvoimakkuudeksi.

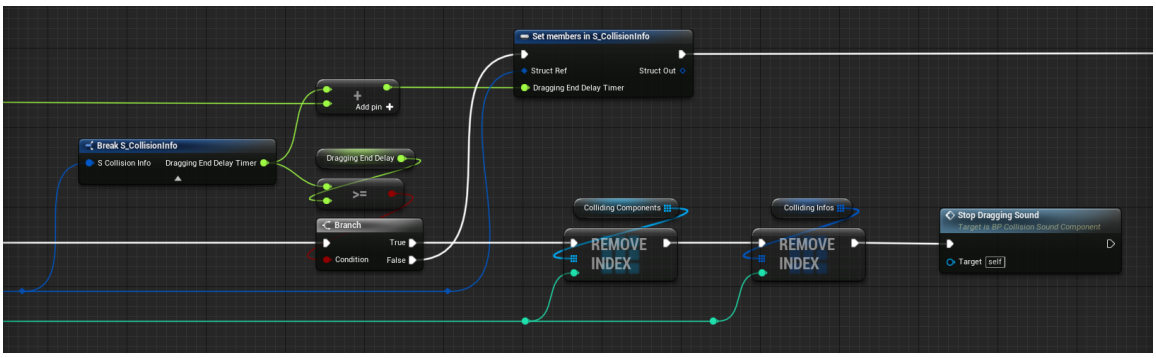


Kuva 18. Calculate Drag Volume -funktion toiminta.

Raahausääni alkaa, kun äänenvoimakkuus on isompi kuin nolla ja se pysäytetään, jos äänenvoimakkuus menee takaisin nollaan. Kun Actor liikkuu kosketuksen aikana, äänenvoimakkuus nousee ja raahausääntä toistetaan. Raahausääni on äänisilmukka (engl. loop) eli ääni, jota voidaan toistaa toistuvasti ja saumattomasti. Raahausääntä toistetaan aina jostain satunnaisesta kohdasta äänisilmukkaa, jotta ääneen saadaan enemmän variaatioita. Raahausääniin lisätään variaatiota myös äänenkorkeuden satunnaisilla vaihteluilla. Samalla tavalla kuin törmäysäänissä äänenkorkeuden kerroin valitaan satunnaisesti joltain lukuväliltä, jossa 1,0 on keskimmäinen arvo. Liitännäisessä on muuttuja, joka vastaa kumpaakin lukuvälin minimi- ja maksimiarvoa, jota liitännäisen käyttäjä voi säätää haluamansa mukaan. Tähänkin muuttujaan on asetettu yleisesti toimiva oletusarvo. Raahausäänen paikkaa tarvitaan myös päivittää jokaisella pelimoottorin päivityksellä, että raahausääni kuuluu koko raahauksen ajan Actorista.

Vaikka logiikkaa vierimisäänille ei ehditty tehdä työn aikana, vierimisääniä pystytään toistamaan vaihtamalla raahausäänten äänisilmukat vierimisääniksi. Työssä testattiin ääniä sylinterin muotoisilla Actoreilla, joita vieritettiin maata pitkin. Kun Actorit vierivät, huomattiin, että äänet alkoivat alusta satunnaisesti vierimisen aikana ja ne kuulostivat katkonaisilta. Syyksi löydettiin se että, vierivät esineet voivat joissain tilanteissa irtautua maasta muutaman fysiikkapäivityksen ajan. Kun tämä tapahtuu, myös äänet loppuvat. Tämä ratkaistiin lisäämällä raahausäänten ehtoihin pieni viive loppuun. Kun Actor irtautuu maasta, odotetaan jokin tietty määritetty aika. Jos Actor aloittaa kosketuksen uudestaan kyseisen ajan sisällä, äänen toistamista jatketaan normaalisti. Jos Actor ei aloita kosketusta uudestaan kyseisen ajan sisällä, äänen toistaminen lopetetaan. Tämä estää, että Actorin hetkelliset irtoamiset eivät aloita vierimisääntä alusta. Kuvassa 19 näkyy raahausäänten ehto, joka odottaa Dragging End Delay -muuttujan ajan ennen kuin ääni pysäytetään ja kosketuksessa oleva Actor poistetaan listasta. Ehto toimii ikään kuin ajastimena, joka lisää kuluneen

ajan Dragging End Delay Timer -muuttujaan ja tarkistaa, onko se mennyt Dragging End Delay -muuttujan yli.



Kuva 19. Ehto katkonaisten raahausäänten estämiseksi.

6.6 Äänimateriaalien tuottaminen

Työtä varten tehtiin viisi erilaista pahvilaatikon törmäysääntä ja yksi raahausääni. Törmäysäänet ladattiin netistä freesound.org-sivustolta, jossa ihmiset voivat jakaa ja ladata äänitiedostoja Creative Commons -lisenssien alla ilmaiseksi [40]. Creative Commons on maailmanlaajuinen voittoa tavoittelematon organisaatio, joka tarjoaa kaikille ilmaisia laillisia työkaluja. Creative Commons tunnetaan lisenssijärjestelmästään, joka antaa helpon ja standardisoidun tavan hallita teosten tekijänoikeuksia. Lisenssijärjestelmä koostuu kuudesta eri lisenssivaihtoehdoista, joita teosten tekijä voi käyttää. Lisenssien ideana on sallia erilaisia luovia oikeuksia teoksen käytöstä, kunhan teoksen tekijä on mainittu. Teosten tekijä voi myös luopua tekijänoikeuksistaan ja luovuttaa teoksensa vapaaseen yleiseen käyttöön Creative Commonsin CC0 -lausunnolla, joka vastaa miten public domain -teokset toimivat. [41; 42; 43.] Törmäysäänet olivat CC0 ääniä eli täysin vapaasti yleiseen käyttöön tarkoitettuja ääniä. Raahausääni äänitettiin omalla Zoom H5 -ääninauhurilla. Kaikki äänet leikattiin, muokattiin paremmaksi ja viimeisteltiin liitännäistä varten käyttäen FL Studio -ohjelmistoa. Yksittäisistä itse äänitetyistä raahausäänistä muodostettiin pitkä äänisilmukka.

Koska liitännäistä oli kehitetty suurimmaksi osin tietokoneella, varmistettiin vielä, että äänet toimivat myös VR-ympäristössä. Koska törmäysäänten toimivuudessa ei ole eroavaisuuksia, jos peliä pelataan tietokoneella tai VR-ympäristössä, liitännäiseen ja peliprojektiin ei ollut tarvetta tehdä paljon muutoksia. Kun liitännäinen oli testattu toimivaksi ja ohjelman lähdekoodi oli kommentoitu, oli liitännäinen valmis palautettavaksi.

7 Pohdinta

Peliääniin saadaan luonnollisuutta monella eri tavoilla. Realistisia ääniä varten ei välttämättä tarvitse laskea monimutkaisia fysiikkalaskuja, vaan yksinkertaisilla äänten variaatioilla ja efekteillä saadaan jo paljon ääniin luonnollisuutta. Tämän työn liitännäistä kehitettiin hakemalla erilaisia ehtoja, joilla pystyttiin rajaamaan törmäykset, jolloin haluttiin äänien kuuluvan. Työn alussa luultiin, että törmäysääniä pitää lähestyä fysiikkaan pohjautuvien laskujen näkökulmasta, vaikka tärkein oli vain luoda illuusiota ja löytää ratkaisuja, millä törmäysäänet kuulostavat luonnollisilta ja loogisilta.

Liitännäistä voitaisiin parantaa tulevaisuudessa monella tapaa. Törmäysäänissä on vieläkin hetkellistä rätinää tietyissä tilanteissa monista ehdoista ja rajauksista huolimatta. Törmäysäänten rajaaminen oli haastavaa työn aikana ja äänten tasapainottamiseen oli hankalaa löytää kaikissa tilanteissa toimivia ratkaisuja. Mahdollisia tulevia parannuksia tähän voisi olla löytää hienovaraisempaa tapaa laskea törmäyspisteiden muutoksia tai laajentaa törmäysääniä toimimaan enemmän fysiikkaan pohjautuvilla laskuilla.

Työssä oli myös ongelmana törmäysten aikana kuuluvat raahausäänet. Kun Actor liikkui tarpeeksi nopealla vauhdilla samalla, kun se törmäsi ja kimposi pois toisesta Actorista, siitä kuului lyhyt hetki raahausääntä. Yksi mahdollinen ratkaisu tähän olisi lisätä lyhyt viive ennen kuin raahausääni toistetaan. Toinen vaihtoehto olisi hiljalleen alussa nostaa raahausäänen voimakkuutta ja lopussa laskea sitä. Raahausäänet eivät myöskään ota huomioon monia kosketuksia, vaan ne toistavat vain ensimmäisen kosketuksen listalta.

Törmäys- ja raahausäänet eivät ota myöskään huomioon, jos Actorit ovat raskaampia tai kevyempiä. Tämä päätettiin, jotta työn minimi- ja maksimiarvoja olisi helpompi määrittää eri painoisille ja kokoisille Actoreille. Actorien paino tai koko ei yleensä muutu pelin aikana, minkä takia painon lisäämä kerroin törmäysvoimiin ei näissä tapauksissa auta minimi- ja maksimiarvojen säätämisessä.

Vierimisääniä liitännäiseen ei ehditty lisätä liitännäiseen työn aikana. Vaikka näitä ei keretty tehdä, vierimisääniä pystyy toistamaan vaihtamalla raahausäänten äänisilmukat vierimisääniksi. Tämä toimii yllättävän hyvin joissakin tilanteissa, esimerkiksi pallon muotoisissa esineissä. Tarve vierimisäänille on yleensä myös harvinaisempaa, jonka takia ne jätettiin suosiolla pois aikarajoitteiden takia.

Työ oli erittäin hyvä oppimistilaisuus tehdä projektia äänten parissa. Työ oli antoisa ja opetti paljon Unreal Engine -pelimoottorista ja yleisesti peliäänistä sekä niiden toistamisesta. Koska olen kiinnostunut äänistä ja niiden ohjelmoimisesta ja haluan erikoistua näihin liittyviin asioihin, työstä saadusta kokemuksesta on varmasti hyötyä myös tulevaisuudessa. Työn aikana saatiin aikaseksi toimiva törmäysääniliitännäinen, jonka pystyy lisäämään helposti Cleverin projekteihin mukaan. Vaikka törmäys- ja raahausäänet eivät ole täydellisiä, ne toimivat suurimmassa osassa tilanteista ja tuovat yllättävän paljon äänellistä immersiota fysiikkapohjaisiin esineisiin. Liitännäinen toteutettiin myös siinä ajatuksessa, että Clever voi halutessaan lisätä helposti lisää toiminnallisuuksia ja ääniä liitännäiseen tulevaisuudessa. Näiden kannalta työ vaikutti onnistuneelta.

8 Yhteenveto

Työn tavoitteena oli toteuttaa liitännäinen, joka lisää Cleverin UE-pelien fysiikkapohjaisiin esineisiin törmäys-, raahaus- ja vierimisäänet. Työn ideana oli lisätä immersiota ja käyttäjäkokemusta Cleverin peliprojekteissa, antaa yhtenäinen toimintapa fyysisten äänten toistamiselle sekä nopeuttaa äänten lisäämistä peliin. Työn tarkoitus oli antaa Cleverille valmis kokonaisuus, jota he pystyvät käyttämään ja lisäämään projekteihinsa helposti.

Peliääniä ilmestyi 1930-luvulla pelihallien ja kasinoiden myötä. Kun videopelit ja pelikonsolit kehittyivät ja yleistyivät nopeaa vauhtia, olivat peliäänet myös osana kehitystä. Ensimmäiset videopelit koostuivat usein yksinkertaisista piippauksista ja suhinoista. Uusia äänipiiriä kuitenkin kehitettiin tiheään tahtiin ja teknologia tuli paremmaksi. Monet arcade-laitteet, pelikonsolit ja kotitietokoneet sisälsivät aluksi PSG-äänipiirejä, mutta digitaalisten ääninäytteiden yleistyessä ja muistin lisääntyessä DAC-äänipiireistä tuli yleisiä.

Työssä käytiin läpi, kuinka peliäänet parantavat pelien palautetta ja immersiota sekä lisäävät viihdearvoa. Äänipalaute antaa pelaajalle välitöntä tietoa pelin tapahtumista, vahvistaa näytöltä nähtyä kuvaa ja yhdistää pelaajan tekemiä toimintoja. Peliäänet parantavat myös immersiota, koska äänet muistuttavat oikeasta elämästä ja peittävät ulkopuolisia ääniä häiritsemästä pelikokemusta.

Työssä käytiin läpi myös, kuinka peliääniefekteihin ja -musiikkeihin saadaan paljon luonnollisuutta varioimalla äänten äänenvoimakkuutta, panorointia, nopeutta, äänenkorkeutta ja ajoitusta sekä lisäämällä ääniin erilaisia DSP-efektejä. Ääniefektejä voidaan myös varioida toistamalla samankaltaisia ääniä satunnaisessa järjestyksessä. Variaation luominen tapahtuu tällöin ennen äänien tuomista peliin, kun äänisuunnittelija luo monia variaatioita jostain tietystä äänestä.

Työn aikana saatiin tehtyä toimiva liitännäinen, joka lisäsi UE-pelien fysiikkapohjaisiin Actoreihin törmäys- ja raahausääniä. Liitännäinen koostui törmäysäänikomponentista, jonka lisääminen mihin tahansa fysiikoilla toimivaan Actoriin antoi sille äänet. Liitännäinen voidaan lisätä mihin tahansa Cleverin UE-peliprojektiin. Liitännäistä varten tehtiin myös oma peliprojekti, jossa törmäys- ja raahausääniä voitiin testata ja kehittää. Törmäys- ja raahausääniä kehitettiin hakemalla erilaisia luonnollisilta ja loogisilta tuntuvia ehtoja, joilla pystyttiin rajaamaan hetket, jolloin haluttiin äänien kuuluvan. Työn liitännäinen varioi myös ääniä muuttamalla äänenvoimakuutta ja -korkeutta

sekä valitsemalla satunnaisia ääniä. Työssä käytettiin netistä ladattuja ja itse äänitettyjä materiaalien ääniä, jotka leikattiin ja muokattiin liitännäistä varten. Vierimisääniä ei ehditty tehdä liitännäiseen työn aikana, mutta niitä voidaan luoda vaihtamalla raahausäänet vierimisääniksi. Vaikka törmäys- ja raahausäänet eivät olleet täydellisiä ja niissä olisi vielä parannettavaa, ne toimivat suurimmassa osassa tilanteista. Työssä todettiin, että äänet toivat immersiota fysiikkapohjaisiin esineisiin.

Lähteet

1 Kajaanin Ammattikorkeakoulu. Pelien hyötysovellukset (VR/AR). Saatavilla: https://www.kamk.fi/fi/Tutkimus-ja-kehitys/Pelien-hyotysovellukset. Viitattu 20.4.2021.

2 Collins K. Game Sound: An Introduction to the History, Theory, and Practice of Video Game Music and Sound Design. Cambridge, MA, USA: MIT Press; 2008.

3 Marks A, Novak J. Game Development Essentials: Game Audio Development. Clifton Park, NY, USA: Delmar Cengage Learning; 2009.

4 Andys-arcade. Original Nutting Associates Computer Space (1971) gameplay. 2017. Saatavilla: https://www.youtube.com/watch?v=YR7gmVpw6Io. Viitattu 26.5.2021.

5 Wolf MJ. Before the Crash: Early Video Game History. Detroid, MI, USA: Wayne State University Press; 2012.

6 Andys-arcade. Original Atari PONG (1972) arcade machine gameplay video. 2014. Saatavilla: https://www.youtube.com/watch?v=fiShX2pTz9A. Viitattu 27.5.2021.

7 Digital Game Museum. Arcade machines. 2012. Saatavilla: https://www.flickr.com/photos/digitalgamemuseum/7695478798/. Viitattu 25.10.2021.

8 Old Classic Retro Gaming. Arcade Game: Space Invaders (1978 Midway/Taito). 2014. Saatavilla: https://www.youtube.com/watch?v=Sx0ZcT2fY0U. Viitattu: 27.10.2021.

9 Hirudov2d. Arcade Longplay - Pac-Man (1980) Midway. 2019. Saatavilla: https://www.youtube.com/watch?v=7O1OYQRqUag. Viitattu 27.10.2021.

10 Joho345. General Instrument AY-3-8910 Sound Chip. 2006. Saatavilla: https://commons.wikimedia.org/wiki/File:AY-3-8910.jpg. Viitattu: 27.10.2021.

11 Evan-Amos. ColecoVision Motherboard Top. 2014. Saatavilla: https://commons.wikimedia.org/wiki/File:ColecoVision-Motherboard-Top.jpg. Viitattu 27.10.2021.

12 The Associated Press. Sales of unearthed Atari games total more than $100,000 – ABC13 Houston. 2015. Saatavilla: https://abc13.com/atari-games-100-000-unearthed/962812/. Viitattu: 27.10.2021.

13 Kent SL. The Ultimate History of Video Games: From Pong to Pokémon and Beyond: the Story Behind the Craze that Touched Our Lives and Changed the World. Roseville, CA, USA: Prima Publishing; 2001.

14 Sean O'Kane. 7 things I learned from the designer of the NES – The Verge. 2015. Saatavilla: https://www.theverge.com/2015/10/18/9554885/nintendo-entertainment-system-famicom-history-masayuki-uemura. Viitattu: 28.10.2021.

15 Evan-Amos. Nintendo Famicom Console Set FL. 2016. Saatavilla: https://commons.wikimedia.org/wiki/File:Nintendo-Famicom-Console-Set-FL.png. Viitattu: 28.10.2021.

16 Evan-Amos. NES Console Set. 2013. Saatavilla: https://commons.wikimedia.org/wiki/File:NES-Console-Set.png. Viitattu: 28.10.2021.

17 Collins K. Playing with Sound: A Theory of Interacting with Sound and Music in Video Games. Cambridge, MA, USA: MIT Press; 2013.

18 Sander Huiberts. Captivating Sound: The Role of Audio for Immersion in Computer Games. Portsmouth, UK: University of Portsmouth; 2010.

19 Grimshaw M. Game Sound Technology and Player Interaction: Concepts and Developments. Hershey, PA, USA: Information Science Reference; 2011.

20 Richardson J, Hawkins S. Essays on Sound and Vision. Helsinki: Helsinki University Press; 2007.

21 Epic Games. About Epic Games. Saatavilla: https://www.epicgames.com/site/en-US/about. Viitattu 7.6.2021.

22 Epic Games. Frequently Asked Questions. Saatavilla: https://www.unrealengine.com/en-US/faq. Viitattu: 28.10.2021.

23 Epic Games. Introduction to C++ Programming in UE4. Saatavilla: https://docs.unrealengine.com/4.26/en-US/ProgrammingAndScripting/ProgrammingWithCPP/IntroductionToCPP/. Viitattu 1.10.2021.

24 Epic Games. Blueprint Overview. https://docs.unrealengine.com/4.26/en-US/ProgrammingAndScripting/Blueprints/Overview/. Viitattu 1.10.2021.

25 Epic Games. Blueprint Best Practices. https://docs.unrealengine.com/4.26/en-US/ProgrammingAndScripting/Blueprints/BestPractices/. Viitattu 1.10.2021.

26 Epic Games. Tools and Editors. Saatavilla: https://docs.unrealengine.com/4.26/en-US/Basics/ToolsAndEditors/. Viitattu: 28.10.2021.

27 Cambridge University Press. Plug-in. Saatavilla: https://dictionary.cambridge.org/dictionary/english/plug-in. Viitattu 24.9.2021.

28 Oxford University Press. Plug-in. Saatavilla: https://www.lexico.com/en/definition/plug-in. Viitattu 24.9.2021.

29 Epic Games. Plugins. Saatavilla: https://docs.unrealengine.com/4.26/en-US/Production-Pipelines/Plugins/. Viitattu 24.9.2021.

30 Epic Games. Levels. Saatavilla: https://docs.unrealengine.com/4.26/en-US/Basics/Levels/. Viitattu 29.9.2021.

31 Epic Games. Actors. Saatavilla: https://docs.unrealengine.com/4.26/en-US/ProgrammingAndScripting/ProgrammingWithCPP/UnrealArchitecture/Actors/. Viitattu 29.9.2021.

32 Epic Games. Components. Saatavilla: https://docs.unrealengine.com/4.26/en-US/ProgrammingAndScripting/ProgrammingWithCPP/UnrealArchitecture/Actors/Components/. Viitattu 24.10.2021.

33 Epic Games. Events. Saatavilla: https://docs.unrealengine.com/4.26/en-US/ProgrammingAndScripting/Blueprints/UserGuide/Events/. Viitattu 7.10.2021.

34 Epic Games. Custom Events. Saatavilla: https://docs.unrealengine.com/4.26/en-US/ProgrammingAndScripting/Blueprints/UserGuide/Events/Custom/. Viitattu 5.11.2021.

35 Epic Games. UPrimitiveComponent. https://docs.unrealengine.com/4.26/en-US/API/Runtime/Engine/Components/UPrimitiveComponent/. Viitattu 24.10.2021.

36 Epic Games. OnComponentHit. Saatavilla: https://docs.unrealengine.com/4.26/en-US/API/Runtime/Engine/Components/UPrimitiveComponent/OnComponentHit/. Viitattu 11.11.2021.

37 Epic Games. Physics Sub-Stepping. Saatavilla: https://docs.unrealengine.com/4.26/en-US/InteractiveExperiences/Physics/Substepping/. Viitattu 11.11.2021.

38 Epic Games. Collision Overview. Saatavilla: https://docs.unrealengine.com/4.26/en-US/InteractiveExperiences/Physics/Collision/Overview/. Viitattu 11.11.2021.

39 Epic Games. Map Range Clamped. Saatavilla: https://docs.unrealengine.com/4.27/en-US/BlueprintAPI/Math/Float/MapRangeClamped/. Viitattu 11.11.2021.

40 The Music Technology Group (MTG). Freesound – help – Frequently Asked Questions. Saatavilla: https://freesound.org/help/faq/. Viitattu 3.11.2021.

41 Creative Commons. Frequently Asked Questions. Saatavilla: https://creativecommons.org/faq/. Viitattu 4.11.2021.

42 Creative Commons. About CC Licenses. Saatavilla: https://creativecommons.org/about/cclicenses/. Viitattu 4.11.2021.

43 Creative Commons. CC0. Saatavilla: https://creativecommons.org/share-your-work/public-domain/cc0. Viitattu 4.11.2021.