Ville Laitala

IMMERSIIVINEN 3D-ÄÄNIPELI

Opinnäytetyö
CENTRIA-AMMATTIKORKEAKOULU
Tieto- ja Viestintätekniikan koulutusohjelma
Lokakuu 2018

TIIVISTELMÄ OPINNÄYTETYÖSTÄ

Centria-ammattikorkeakoulu	**Aika** Lokakuu 2018	**Tekijä/tekijät** Ville Laitala
Koulutusohjelma Tieto- ja viestintätekniikan koulutusohjelma		
Työn nimi IMMERSIIVINEN 3D-ÄÄNIPELI		
Työn ohjaaja Kauko Kolehmainen		**Sivumäärä** 66
Työelämäohjaaja David Oliva		

Tämän työn tavoitteena oli luoda immersiivinen 3D-äänipeli. MyTrueSound Oy -yritys oli saanut toimeksiannon Näkövammaistenliitto ry:ltä, jota kautta se tuli tietooni. Tavoitteena oli luoda demo yrityksen käyttöön sekä kasvattaa kirjoittajan ohjelmointitaitoja ja oppia Unity käyttö syvemmin. Opinnäytetyössä käytettiin Unity3D-pelimoottoria sekä Google Resonancea äänentoisto väliohjelmistona.

Opinnäytetyön teoriaosa alkoi immersiivisen pelin perusteista ja käsitteistä, minkä jälkeen kerrotaan pelimoottoreista, Unitystä sekä Google Resonancesta, 3D-grafiikasta ja äänistä videopeleissä syvemmin. Lopuksi käytännön osiossa kerrotaan, kuinka demo luotiin.

Opinnäytetyön tuloksena syntyi yksinkertainen, toimiva Unity-3D:llä pelattava äänipelidemo, joka käyttää Google Resonancea äänentoisto väliohjelmistona.

Asiasanat
C#, Google Resonance, pelinkehitys, Unity, äänet, äänipeli

ABSTRACT

Centria University of Applied Sciences	**Date** October 2018	**Author** Ville Laitala
Degree programme Information and Communication Technologies		
Name of thesis IMMERSIVE 3D AUDIO GAME		
Instructor Kauko Kolehmainen		**Pages** 66
Supervisor David Oliva		

The aim of this thesis was to create an immersive 3D-Audio game. The commision originated from myTrueSound Oy. The aim was to create a demo for the company and also to increase the author's programming skills and learn Unity more deeply. Unity3D game engine was used in the thesis and Google Resonance Audio as audio middleware.

The theoretical part of the thesis started with the basics and concepts of an immersive game, followed by more about game engines, Unity, Google Resonance, 3D graphics and sounds in video games. Finally the practical section explains how the demo was created.

As a result of the thesis, a simple, functional Unityn 3D game demo was created that uses Google Resonance as audio middleware.

Key words
C#, Google Resonance, game development, Unity, audio, audiogame

KÄSITTEIDEN MÄÄRITTELY

2D- ja 3D- grafiikka	Kaksi- tai kolmiulotteista digitaalista kuvaa tai tekstiä.
3D-ääni	Ääni, jolla on sijainti ja nopeus suhteessa kuuntelijaan.
Ambisonic	Täyden ympyrän muotoinen äänen lähde, joka vaakasuoran tason lisäksi kattaa äänilähteet kuuntelijan yläpuolella ja alapuolella.
Asset	Assetteja ovat 3D-mallit, tekstuurit, äänitiedostot ja skriptit.
Audio väliohjelmisto	Kolmannen osapuolen äänimoottori, joka antaa enemmän työkaluja pelien äänien kontrolloimiseen.
Boolian tila	Totuusarvo tosi/epätosi (true/false).
CPU	Tietokoneen prosessori, joka suorittaa konekielellä käännettyjä käskyjä.
Fysiikkamoottori	Osa pelimoottorissa, joka simuloi likimääräisesti jonkin fyysisen esineen vuorovaikutusta ympäristöön.
HRTF	Head Related Transfer Function, eli kuinka korvat vastaanottavat ääniä eri suunnista.
Lokalisointi	Toimia, joilla sovitetaan tuotteita, kuten julkaisuja tai ohjelmistoja vieraaseen ympäristöön.
Occlusion	Imitoi, kuinka äänet havainnoidaan suljetussa tilassa.
Pelihahmo	Pelin sisäinen objekti, jota pelaaja liikuttaa ja on sen kautta vuorovaikutuksessa peliympäristön kanssa.
Renderointi	Tietokoneohjelma piirtää kuvaa.
Ruudunpäivitys	Kuinka monta kertaa sekunnissa kuvaa näytetään - mitä pienempi arvo sitä vähemmän näytetään kuvia, jolloin liikkuva kuva näyttää hitaalta tai nykivältä.
Scene	Sisältää pelin peliympäristön ja käyttöliittymät.
Skripti	Komentosarja, joka ohjaa peliobjektien toiminallisuuden. Sillä muokataan, käsitellään ja automatisoidaan järjestelmiä.
Spatial audio	Simuloi ääniä niin, että ne kuulostavat samalta kuin todellisessa elämässä.
Tekoäly	Tietokoneohjelma, joka kykenee tekemän älykkäinä pidettäviä toimintoja.
Trigger	Peliobjekti, joka tekee jotain, kun siihen kosketaan.
Äänipeli	Peli, jonka toiminta perustuu ääneen.

TIIVISTELMÄ
ABSTRACT
KÄSITTEIDEN MÄÄRITTELY
SISÄLLYS

1 JOHDANTO

Tämän opinnäytetyön tarkoitus on toteuttaa immersiivinen 3D-äänipeli, joka on suunnattu henkilöille, joilla on näkövaikeuksia. Äänipelin tekemiseen käytetään Unity-3D-pelimoottoria sekä äänentoisto väliohjelmistona Google Resonance -lisäosaa. Toimeksiantaja on vuonna 2017 perustettu turkulainen MyTrueSound -yritys, joka tuottaa konsultointipalveluita 3D-äänien suhteen muille yrityksille, ja itse toimeksianto tuli Näkövammaistenliitto ry:ltä. Tarkoituksena on toteuttaa mahdollisemman immersiivinen 3D-ääniympäristö.

Opinnäytetyön aikana tutkitaan, kuinka äänillä voidaan ohjata pelaajaa demossa, ja tärkeintä on, että pelaaja ymmärtää helposti, miten ääni auttaa häntä kulkemaan peliympäristössä ja tajuamaan ympäristön rajoituksia. Erityisesti pitäisi välttää tilanteita, joissa pelaaja ei tiedä, minne pitäisi suunnata. Äänet ovat erittäin tärkeä osa peliympäristöä: yhtä lailla kuin pelingrafiikka, se auttaa pelejä immersion sekä oman tyylin luonnissa.

Opinnäytetyössä kerrotaan tietoa pelimoottoreista ja Unitystä ja sen ominaisuuksista ja siitä, kuinka sillä luodaan pelejä. Myös immersiivisen pelin ominaisuudet sekä vaiheet käydään läpi, minkä lisäksi käydään läpi 3D-grafiikka ja pelien äänien ominaisuudet. Lopuksi käytännön osuudessa käydään läpi, kuinka peli luotiin.

2 IMMERSIIVINEN PELI NÄKÖVAMMAISELLE JA PELIMOOTTORI

Immersiolla tarkoitetaan pelaajan uppoutumista peliin ja läsnäolon tuntemista. Immersiivisessä pelissä pelaaja on kuin osa sen maailmaa. Immersio on tärkeää monissa peleissä ja sitä tavoitellaan tässäkin. Abstraktimmissa peleissä, kuten pulmapeleissä, se ei ehkä ole tavoiteltua, kuten muissa tavallisissa tarinavetoisissa peleissä.

Pelaajien uppoutuminen peleihin ei ole aina välitöntä, vaan se tapahtuu vähitellen ja vaiheittain. Vaiheita kutsutaan nimillä sitoutuminen, syventyminen ja täydellinen immersoituminen. Pelaajat siirtyvät vaiheesta toiseen huomaamattomasti, kuitenkin aina tässä järjestyksessä. Näitä vaiheita on hankala käsitellä, koska ne ovat henkilö- ja pelikohtaisia. (Brown & Cairns 2004, 1297-1300)

Syventymisvaiheessa pelaaja kokee mielenkiintoa pelin eri aspekteja kohtaan. Pelaajan syventyessä peliin pelaaja huomaa nauttivansa haasteista, visuaalisesta puolesta ja juonesta. Pelaaja tuntee pelin olevan laadukas. Täydellisessä immersoitumisessa pelaaja tuntee olevansa läsnä pelissä. Tässä vaiheessa pelaaja on uppoutunut pelimaailmaan ja yleensä menettää ajantajunsa. Peli, joka onnistuu tässä, on hyvin tunnelmallinen ja on kyennyt luomaan empatiayhteyden pelaajaan. Immersiotasojen saavuttaminen vaatii paljon tarkkaa ja hyvää suunnittelua pelintekijöiltä ja intohimoa luoda maailma, josta pelaaja voi nauttia. Immersion luomiseen on tehty monia keinoja ja niitä keksitään lisää. (Brown & Cairns 2004, 1297-1300)

Immersion rikkominen on erittäin helppoa ja tavanomaiset elementit, kuten pelikenttiä rajaavat abstraktit rajat tai päättömästi käyttäytyvä tekoäly, voivat murtaa pelaajan tunnelman. Yksi yleisimpiä immersion rikkojia ovat peliympäristön osat, joihin pääsy rajataan selittämättömin keinoin. Tällaisia ovat esimerkiksi pelimaailmassa olevat esineet, joiden ohi voisi helposti mennä oikeassa elämässä mutta estävät pelihahmon etenemisen. Tekoälyn puute, dialogi tai ääninäyttely ovat yleisiä immersion rikkovia tekijöitä. Samaa dialogia toistavat hahmot tai useampi hahmo, jotka ovat saman henkilön ääninäyttelemiä, rikkovat monesti immersiota.

2.1.1 Tunnelma

Tunnelma tarkoittaa pelimaailman rikkautta, syvyyttä, ilmapiiriä ja sen välittämiä tuntemuksia. Niillä pyritään välittämään pelaajalle pelin maailmaa ja saada pelaaja vakuuttamaan pelimaailman todellisuudesta. Tunnelmaelementeillä luodaan yhteen pelimaailman eri osat ja sidotaan ne tarinaan. Myös elävällä ympäristöllä ja rikkaalla äänimaailmalla voidaan luoda tunnelmaa. (Game Architecture and Design 2018.)

Pelaajat uppoutuvat, kun kokevat pelimaailman olevan elossa. Pelimaailma, jossa on historiaa, draamaa, syvyyttä ja yksityiskohtia, on pelaajalle mielenkiintoinen ja siihen on helppo osallistua. Pienetkin lisäykset voivat luoda sen eloon. Urheilupeleissä harvoin on tarinaa, mutta hurraava yleisö tai kommentaattorit luovat tunnelmaa. Kenttäsuunnittelu ja pelaajalle näkyvä ympäristö ovat suurimmat tekijät elävän maailman laatimiseksi. Pienet yksityiskohdat, kuten pieneläimet metsässä ja huonekalut pelihahmojen kodeissa, korostavat pelimaailman uskottavuutta. Nämä antavat todellisen, elävän maailmankuvan pelaajalle. Näitä voidaan käyttää myös tarinan luomisessa, pelaajan johdattelemisessa ja tunteiden herättämisessä. Yksityiskohtia voi myös käyttää pelimekaanikoissa, kuten eläinten metsästystä avoimen maailman peleissä. (Game Architecture and Design 2018.)

2.1.2 Tarina

Tarinat antavat pelaajille syyn uskoa pelimaailmaan ja välittää siitä. Tunnelma ja ilmapiiri auttavat tässä. Tarinalla voidaan konkretisoida pelien abstraktiutta. Tarinalla ja kerronalla saadaan myös pelaaja uppoutumaan peliin tunnetasolla usein luomalla yhteys pelaajan ja pelattavan pelihahmon välille. Pelaaja, joka tuntee yhtenäisyyttä pelihahmoonsa, kokee pelin tarinan ja maailman syvällisemmin ja henkilökohtaisemmin kuin pelaaja, joka tätä yhtenäisyyttä ei tunne. Pelkän pisteiden keräämisen tai persoonattomien hahmojen surmaamisen sijaan pelaaja voisi kerätä esineitä, joilla on jotain historiallista merkitystä, tai pelastaa pelihahmoja, joihin henkilö tuntee jonkinlaista vetoomusta.

Pelit alkavat muiden medioiden tavalla premissistä. Premissi on tarinan lähtökohta, jonka tarkoitus on herättää pelaajan mielenkiinto tarinaa ja maailmaa kohtaan. Se asettelee maailman tilannetta ennen pelin alkua ja alustaa konfliktin. Konfliktit ovat pelaajan ja hänen tavoitteidensa väliin asetettuja esteitä ja haasteita. Ne luovat jännitystä ja epävarmuutta tarinan lopputuloksesta ja kärjistyvät pelin edetessä, kunnes peli loppuu kliimaksiin, huipentumaan, ja tarina ratkeaa. Konfliktit toimivat yleensä parhaiten, kun

ne kohdistuvat pelaajan hahmoon. Tekemällä konfliktista henkilökohtaisen pelaaja yleensä välittää siitä enemmän. (Game Design Workshop 2018.)

2.1.3 Flow

Flow eli virtauskokemus tai optimaalinen tila on yksi pelinkehittäjien omaksumista psykologisista konsepteista, ja sitä pidetään yhtenä pelaamisen syistä. (Gigi 2011.)

Flown määritellään olevan tunnetila, jossa ihminen sitoutuu aktiviteettiin niin voimakkaasti, että millään muulla ei tunnu olevan merkitystä. Sen osatekijöitä ovat keskittyminen, toiminnan ja tietoisuuden sekoittuminen sekä hallinnan ja muuntuneen ajan tunne. Pelinkehityksen näkökulmasta Flow kuvastaa tilaa, johon pelaaja uppoutuu pelaamisen yhteydessä.

Tietyt edellytykset, kuten pelaaja ymmärtää tavoitteet, peli antaa selkeää palautetta pelaajalle etenemisestä pelissä, peli on tasapainossa tavoitteiden ja haasteiden kanssa eikä pelaajan keskittyminen häiriinny esimerkiksi huonon käyttöliittymän takia auttaa pelaajaa uppoutumaan Flow-tilaan. Tasapaino haastavuuden ja pelaajan kykyjen välillä on merkittävin vaatimus. (Rutledge 2014.)

Pelien tehtävien tulisi olla sopivan haastavia, jotta pelaaja ei tunne turhautumista tai ahdistusta. Pelaajan oppiminen ja kehittyminen pelissä pitää myös ottaa huomiin. Pelin haasteiden on kasvettava pelaajien kykyjen kanssa. (Gigi 2011.)

2.2 Pelien mahdollistaminen näkövammaiselle

Käytämme aistejamme kerätäksemme tietoa ympäröivästä maailmasta. Jos kuulemme yllättävän törmäyksen, saatamme hätkähtää, tai kun haistamme jotain hyvää keittiöstä, meidän tekee mieli syödä. Jos menetämme yhden aisteistamme, kuten näön, miten voimme pelata videopelejä? Henkilöt, joilla on näkövaikeuksia, tarvitsevat äänitehosteita onnistuakseen pelien läpäisyssä. Esimerkiksi lähestyvät hyökkääjän askeltenäänet ja karkuun juoksevan hyökkääjän askeleet kuulostavat erilaiselta. Monissa peleissä äänet parantavat vain näkevän pelaajan kokemusta. Joissakin ne sallivat pelaajien, joilla on näkövaikeuksia nauttia ja pelata pelejä. Näitä pelejä kutsutaan helppopääsyisiksi peleiksi. Useat yritykset sekä akateemiset laitokset luovat tarjontaa tälle. (Science Buddies 2011.)

Tärkein asia on korvata visuaaliset vihjeet äänillä. Äänipeleissä pelaajat suuntavat äänillä. Käyttämällä kuulokkeita pelaaja onnistuu erottamaan äänet oikealta ja vasemmalta tarkasti: pelaajat kuulevat, mistä äänet tulevat, ja pystyvät luomaan kuvan ympäristöstä. Esimerkiksi jos kävelee käytävää pitkin ja oikealla puolella aukeaa toinen käytävä, saa pelaaja tämän tiedon kuulemalla askeltenäänien kaikuvan oikealta puolelta, ja jos joku hyökkää pelaajan kimppuun, käännetään pelihahmoa, kunnes hyökkääjän äänet kuuluvat edestäpäin, tai piippaava ääni voimistuu, kunnes pelaaja on äänilähteen edessä. (Taylor 2013.)

Jos jotain edustetaan grafiikoilla, siihen voidaan myös lisätä äänitehosteet tunnistamaan se. Esimerkiksi luola voidaan tunnistaa, kun askelten äänet vaihtuvat kaikuviksi. Eri alustoja voidaan edustaa eri askeläänillä. Pelaaja voi kuulla ankan ääniä lasten peleissä tai zombien ääniä ammuskelupeleissä. Murina voi edustaa koiraa tai zombia, ja myös sanat toimivat. Ruutua lukevat ohjelmat, jotka automaattisesti lukevat kaiken tekstin pelaajalle ääneen, auttavat, kun pelissä on paljon tekstiä. Esimerkiksi esineiden kuvailu auttaa näkövaikeuksista kärsiviä: he eivät esimerkiksi halua, että selitetään mitä sininen tarkoittaa vaan mikä tavara on sininen. (Taylor 2013.)

Äänien pitää olla kiinnostavia. Ihmiset haluavat visuaalisen hienouden lisäksi myös hyviä ja kiinnostavia ääniä. Taustaäänet, kuten lokkien, kissojen ja kellojen äänet sekä ovien aukeamiset, sulkeutumiset ja ihmisten kävelyäänet, auttavat luomaan kiinnostavan ympäristön. (Taylor 2013.)

2.3 Pelimoottorit

Pelimoottori on ohjelmisto, joka toimii pelin pohjana ja on taustalla toimiva ohjelmisto, joka antaa valmiita osia sekä järjestelmiä pelin ohjelmiston rakenneosiin. Kehittäjät voivat pelimoottorin päälle kehittää pelejä konsoleille, mobiililaitteille ja tietokoneille. Pelimoottori tarjoaa grafiikkamoottorin 2D- ja 3D-grafiikalle, joka piirtää pelin grafiikan käyttäjälle ohjelmiston määrittelemällä tavalla sekä huomioi käyttäjän syötteen. Moottori myös sisältää fysiikkamoottorin, törmäyksen tunnistuksen, äänet, animoinnin, tekoälyn ja lokalisointituen. (Ward 2008.)

Pelimoottorin tarjoamat osat auttavat pelinkehittäjiä keskittymään pelin tekemiseen ja kehittämiseen. Pelimoottori myös mahdollistaa pelinkehittämisen monelle pelialustalle samanaikaisesti ilman suurten muutosten tarvetta. (Ward 2008.)

Pelimoottoreiden kehitys on tiimityötä, ja osa-alueille on omat kehittäjäryhmät, kuten fysiikkamoottori ja äänimoottori. Pelimoottoreita on kolmea erilaista: lähes valmiita pelimoottoreita, jotka antavat kattavan paketin eri ominaisuuksia pelikehittäjille, jotka eivät vaadi suurta työtä kehittäjiltä. Lisäksi on täysin alkeellisia moottoreita, joiden päälle kehittäjän pitää itse rakentaa, ja valmiita pelimoottoreita, jotka ovat täysin valmiita, mutta joita ei voi muokata tarpeiden mukaan, ja jotka eivät tarvitse ohjelmointia suuresti. (Ward 2008.)

2.3.1 Unity-3D-pelimoottori

Unity on Unity Technologiesin kehittämä 3D- ja 2D-pelien kehitysympäristö. Unity 2018 on yhtiön uusin versio tällä hetkellä Unity-pelimoottoreista. Unity-pelimoottorit voivat käyttää laajaa valikoimaa eri alustoja, mikä on sen yksi etu. Alustoihin kuuluvat Applen iOS, Googlen Android, PC-käyttöjärjestelmät sekä pelikonsolit. (Platform 2018.) Unity-termillä voidaan tarkoittaa eri asioita, kuten Unity-editoria tai Unity-pelimoottoria.

Unity-pelimoottori on yksi suurimmista ja suosituimmista pelimoottoreista markkinoilla. Unity tavoittaa 3 miljardia laitetta, ja se on asennettu 24 miljardia kertaa 12 kuukauden aikana. (Public Relations 2018.) Unitystä on saatavilla kaksi eri versiota: Personal Edition ja Professional Edition. Professional on ammattilaisten käyttöön tarkoitettu versio, jonka hinta räätälöidään käyttäjätarpeiden mukaan. Se sisältää Personal Edition -ominaisuuksien lisäksi Unity Analytic Pro, sekä mahdollisuuden käyttää pelimoottorin tulevia beta-versioita. Personal Editionista nämä puuttuvat. (Software 2018.)

2.3.2 Ohjelmointi

Unity käyttää C#:a ohjelmointikielenään. C# on alan standardikieli, joka on hyvin samanlainen Javan ja C++:n kanssa. (Unity scripting languages 2018.) Unityn rakennuspalikat eli sisäänrakennetut komponentit (Components) ja peliobjektit (GameObjects) auttavat rakentamaan pelin Unityllä. Kaikki alkaa Unityllä peliobjekteista, ja kaikki objektit ovat peliobjekteja: hahmot, valot, erikoistehosteet, lavasteet. Peliobjektit eivät tee mitään itsessään. Jotta ne tekisivät jotain, niille pitää lisätä tarpeisto eli properties, joka lisätään lisäämällä komponentti. Komponentit määrittelevät ja kontrolloivat niitä peliobjekteja, joihin ne on lisätty. Esimerkiksi valoa tehtäessä, peliobjektiin lisätään valokomponentti. (GameObjects 2018.)

Muuttujat (variables) tarkoittavat sitä, että komponenteilla on useita muuttuvia ominaisuuksia, joita voi muokata joko inspektorin kautta ja/tai skriptin kautta. Näitä ovat esimerkiksi valon väri, intensiteetti ja pituus. (Variables 2018.)

Ohjelmoinnin avulla voi muuttaa jokaista objektia ja sen ominaisuuksia. Unityn sisäänrakennetut komponentit ovat erittäin monipuolisia, mutta niiden lisäksi tarvitsee skriptejä. Skriptien avulla voi tehdä oman pelilogiikan ja pelin käyttäytymisen. Skriptikomponenttien avulla voi laukaista pelin tapahtumia, tarkistaa törmäykset, lisätä fysikaalisia ominaisuuksia, ohjata peliä pelaajan komennoilla ja paljon muuta. (Components 2018.)

2.4 Google Resonance

Google Resonance on tehokas spatiaalinen ääniä teknologia, joka on optimoitu suorituskykyä ja monien alustojen tukea silmällä pitäen. Edistyksellisien ääniteknisten ominaisuuksien ansiosta Resonance Audio menee perustavanlaatuisen 3D-tilaratkaisun edelle tarjoamalla tehokkaita työkaluja monimutkaisten ääniympäristöjen mallintamiseen. SDK mahdollistaa äänilähteiden ohjauksen mukauttamisen, lähellä olevien kenttien vaikutuksen, geometriapohjaisen kaikuokkluusio ja ambisoni-äänitiedostojen tallennuksen. Monien alustojen tuen ansiosta Resonance Audio SDK yhdistyy saumattomasti suosituimpiin peli- ja audiomoottoreihin sekä digitaalisiin äänityöasemiin (DAW), joiden ansiosta kehittäjä voi keskittyä immersiivisen äänen luomiseen. Resonance Audio lisää immersiota VR- ja AR-mediassa ja -peleissä. Se antaa kehittäjille mahdollisuuden renderöidä ääniä joka suunnasta simuloiden oikean maailman ääniä. (Overview 2018.)

Resonance Audio tarjoaa korkealaatuisen spatiaalisen äänen mobiili- ja tietokonesovelluksille. Resonance muuttaa sisäisesti kaikki äänilähteet globaaliksi korkealaatuiseksi Ambisonic-äänikentäksi. Tämä mahdollistaa sen, että HRTF:ää eli Head Related Transfer Functionia voidaan soveltaa vain kerran äänikentille sen sijaan, että ne sisältäisivät yksittäisiä äänilähteitä. Tämä optimointi pitää CPU-kustannukset yhtä äänilähdettä kohti minimissä, mikä mahdollistaa useampien samanaikaisten lähteiden toiston kuin useimmissa perinteisissä äänilähdealustustekniikoissa. (Overview 2018.)

Resonance Audion ambisonicjärjestys on säädettävissä, joten alueellista tarkkuutta voi hallita. Korkeamman asteen ambisonic-äänikenttien avulla parannetaan tarkkuutta ja lähdelokalisaatiota. Resonance

audion digitaaliset signaalikäsittelyalgoritmit on optimoitu spatialisoimaan satoja samanaikaisia 3D-äänilähteitä ilman, että se haittaa äänenlaatua jopa mobiililaitteilla. (Overview 2018.)

2.4.1 Äänilähteet

Resonance Audio sallii virtuaalisten äänilähteiden ja Ambisonic-äänikenttien spatialisoinnin. Äänilähteet, joita kutsutaan usein lähdepisteiksi, määrittelevät äänilähteen tietyssä pisteessä virtuaalisessa ympäristössä. Jokainen äänilähde on renderoitu monoäänivirrasta ja äänilähde voidaan konfiguroida suuntauskuvion ja etäisyyden vaimennuskäyrien avulla. (Design tips 2018.)

Jos esimerkiksi kiinnittää äänilähteen lintuun, käyttäjä kuulee linnun siipien äänen ja laulun luonnollisesti, kun se menee kauemmaksi tai lähemmäksi. Äänilähteitä voi myös jakaa koko ympäristöön luomaan yleistä tunnelmaa. Äänilähteitä tehtäessä käytetään yksisuuntaisia äänitiedostoja, joihin ei kuulu mitään kaikua. (Design tips 2018.)

2.4.2 Ambisonic-äänikentät

Kuten valonsäteet visuaaliselle renderoinnille, Ambisonic edustaa koko 360 asteen äänimaailmaa ohjelmoimalla ääniaaltoja kuuntelijan ympärille pallon muotoisena. Resonance Audio tukee jopa kolmannen kertaluokan Ambisonic-dekoodausta. Jokainen Ambisonic-äänikenttä tuotetaan monikanavaisesta äänivirrasta. Virta aina renderoidaan kuuntelijan sijaintipaikassa. Koska Ambisonic-tiedostot vastaavat vain pään kiertoon, äänikentät toimivat parhaiten edustamaan ääntä etäisyyksien päässä. (Design tips 2018.)

Riippuen ympäristöstä, jota luodaan, on mahdollista käyttää äänihuoneita. Äänihuoneet tarjoavat varhaisia heijastuksia ja kaikuja. Tämä tekee äänistä realistisempia, kun seinät tai rakenteet ovat lähellä käyttäjän asemaa skenessä. Kaiku ja huoneen materiaalipinnat voidaan määritellä vastaamaan rakennusympäristön äänitehosteita. Äänihuoneet ovat hyödyllisiä, kun skene tapahtuu huoneen sisällä. Skenen tapahtuessa ulkona äänihuone saattaa kuulostaa vähemmän luonnolliselta. Tämä johtuu siitä, että ulkona maa voi olla ainoa ääniä heijastava pinta. (Design tips 2018.)

3 3D-GRAFIIKKA

3D-grafiikan maailma on monimutkainen ja peli-, filmi, arkkitehtuuri ja insinöörialat käyttävät sitä hyödykseen. 3D-taitelijat ja -suunnittelijat käyttävät tiettyjä tekniikoita ja prosesseja, kuten 3D-grafiikkasuunnittelua, visualisointia ja animointia tuodakseen visionsa eloon näytölle.

3D-tietokonegrafiikka on kolmiulotteista grafiikkaa, joka käyttää X-, Y- ja Z-koordinaattiakseleita kun taas 2D-grafiikka käyttää vain X- ja Y-koordinaattiakseleita. Tietokoneen näytölle luotu kuva on kuitenkin kaksiulotteista, mikä näyttää kolmiulotteiselta. X-koordinaattiakseli on horisontaalinen, y-koordinaattiakseli vertikaalinen ja z-koordinaatti kuvaa syvyyttä, mikä tarkoittaa sitä, kuinka kaukana 3D-grafiikka piirretään 3D-ohjelmassa. (Slick 2014a.)

3D-grafiikan piirtoon tietokoneen näytölle on kolme vaihetta. Sovellusvaihe eli application stage luo kolmiulotteisen mallin, joka piirretään ruudulle. Geometrinen vaihe on, joka vastaanottaa tietoa sovellusvaiheesta ja luo mallille geometrisia muunnoksia ja leikkauksia mallin kolmannelle vaiheelle. Rasterointivaihe luo 3D-mallista ruudulle kaksiulotteisen kuvan, joka koostuu pikseleistä, ja kuva näyttää katsojalle kolmiulotteiselta. (Puhakka 2008.)

3D-grafiikka koostuu monikulmioista eli polygoneista, monikulmiot koostuvat vertexeistä eli reunapisteistä, jotka ovat pienempiä elementtejä. Reuna on suora, joka kulkee edgestä eli reunapisteestä toiseen. Reunojen välillä olevaa aluetta kutsutaan faceksi eli tasoksi, joka on itse monikulmio eli polygon. Kun monikulmiot ovat vierekkäin yhtenäisesti, sitä kutsutaan meshiksi eli monikulmioverkoksi, joka antaa mallin 3D-kappaleelle. Mitä tiheämpi verkko on, sitä tarkemmalta se näyttää. Tarkemmat polygoniverkot vaativat aina enemmän tehoja työasemalta renderöintiin. (Slick 2014b.)

3.1 Piirtorajapinnat

OpenGL eli Open Graphics Library ja Microsoftin DirectX ovat yleisimmät ohjelmointirajapinnat. Ne käyttävät työaseman GPU:ta eli grafiikkasuoritinta ja määrittelevät, miten ohjelmisto tarjoaa tietoa tai palveluita sovelluksille tai muille tietojärjestelmille. Ohjelmointirajapinnat toimivat laitteiston ja sovel-

luksen välillä. Ne helpottavat ohjelmoijien työtä, sillä ohjelmoijat voivat ohjelmoida sovelluksen tukemaan ohjelmointirajapintaa, jolloin ei tarvitse ohjelmoida sovellusta erikseen laitteistoa varten. (Sharma 2013.)

Microsoft DirectX on suljetun lähdekoodin grafiikkarajapinta, mikä tarkoittaa sitä, että vain Microsoft voi muokata sitä. Ensimmäinen versio julkaistiin vuonna 1995 ja viimeisin versio on DirectX12, joka julkaistiin 2015 ja joka vaatii Windows 10:n toimiakseen. (Langley 2015; Sharma 2013.)

OpenGL on avoimen lähdekoodin grafiikkarajapinta, joka on laitteistoriippumaton ohjelmointirajapinta interaktiivisen tietokonegrafiikan tuottamiseen. OpenGL kehitettiin vuonna 1992 Silicon Graphicsin IRIS-piirtorajapinnan pohjalta sen korvaajaksi. OpenGL on tarkoitettu piirtämään perusprimitiivejä ja sitä käytetään opetuksessa, ohjelmistoissa, laitteissa ja peleissä. OpenGL ei ole sidoksissa yhteen käyttöjärjestelmään, vaan sitä käytetään macOS-, Linux- ja Windows-käyttäjäjärjestelmissä. (Singer 2013.)

3.2 Valo ja perspektiivi

3D-grafiikan suunnitellussa otetaan huomioon valo ja perspektiivi, kun taas normaalissa elämässä ei paljon mietitä, miten valo heijastuu tai täyttää huoneen. Jokainen osa 3D-grafiikasta pitää valaista jotenkin. Yksi tekniikkaa on käyttää ray-tracingia, joka laskee reitin valolähteen kohteesta kimmoten seinistä, peileistä ja muista heijastavista kohteista. Tämä on erittäin monimutkaista jo yhden valolähteen kanssa. Valo on tärkeää, sillä se antaa painoa ja kiinteyttää objektin ja antaa varjostuksen ja varjon. Varjostus on voimassa, kun valo loistaa objektiin voimakkaammin toisella puolella. Tämä antaa esimerkiksi pallolle pallon muodon ja peiton taitteet saavat peiton näyttämään syvältä ja pehmeältä. Nämä eroavaisuudet valossa mahdollistavat objektille syvyyden vaikutuksen sekä korkeuden ja leveyden. Painon illuusio tulee varjosta. Kiinteät objektit luovat varjon, kun valo loistaa niihin. Koska olemme tottuneet näkemään, että oikeat objektit ja ihmiset luovat varjon, se vahvistaa illuusiota siitä, että katsomme oikeaa objektia eikä tietokonegeneroitujakuvioita. (Franklin 2008a.)

Perspektiivi tarkoittaa sitä, kun suoralla tiellä seistessä ja katsoessa horisonttiin näyttää siltä, että tien molemmat puolet yhdistyvät ja puut kauempana näyttävät pienemmältä kuin lähellä olevat. Tämä on “yhden pisteen perspektiivi”. Suurin osa 3D-grafiikasta käyttää “yhden pisteen perspektiiviä”. Yleisin tekniikka perspektiivin luomiseen on Z-buffer. Z-buffer saa nimensä yleisestä akselin nimestä. Z-buffer

antaa jokaiselle polygonille numeron sen perusteella, kuinka lähellä ne ovat ruutua. Mitä suurempi sitä lähempänä se on horisonttia. (Franklin 2008a.)

3.3 Syväterävyys ja antialiasointi

Yksi keino saada 3D näyttämään oikealta on syväterävyys. Esimerkiksi katsottaessa puita läheltä kauemmat puut eivät näytä tarkalta. Tietokone pystyy näyttämään kuvaa niin, että kaikki näkyy tarkasti, mutta tämä ei näytä oikealta ihmissilmään. Toinen keino on antialiasointi. Tietokone on hyvä luomaan suoria linjoja, mutta kun pitää luoda kaartavia viivoja, ne näyttävät “portailta” eivätkä sulavalta viivalta. Jotta kaarteet näyttävät sulavilta, tietokone lisää silmää huijatakseen viivaan asteittain erivärisiä pikseleitä riippuen taustasta, jotta viiva näyttäisi kaartuvalta. (Franklin 2008b.)

4 PELIEN ÄÄNET

Pelien äänistä on tullut erittäin tärkeä osa immersiota. Immersio on yksi tärkeimmistä pelien ominaisuuksista nykypäivänä erityisesti yksinpeleissä, sillä immersion puute voi pilata muuten hyvän pelin tai tehdä keskinkertaisesta pelistä hyvän. Henkilö, joka luo äänitehosteita työkseen, tunnetaan nimellä äänisuunnittelija eli sound designer. Hänen päätyönsä on tehdä hyvästä pelistä vielä parempi. Pelien äänistä on tullut suosittu ja erittäin tärkeä osa pelejä, ja monet pelialan ammattilaisista ovat yrittäneet selvittää peliäänien eri elementit. Ensimmäiset pelit olivat ehkä alkukantaisia, mutta kehittäjät ymmärsivät äänitehosteiden tärkeyden. Peliäänien monimutkaisuus on synnyttänyt täysin oman työkuvauksen ja uran. Peliäänet kattavat kaikki mahdolliset pelimaailman tunnelman luomiseen tarkoitetut äänet aina pelaajan nappien painallusten palauteääniin. (Isaza 2010.)

4.1 Peliäänien tarkoitus

Peliäänien tarkoitus on antaa palautetta pelaajalle, luoda immersiota ja viihdyttää, ja jokainen näistä on tärkeä osa hyvän pelin luomiseksi. Monet pelit käyttävät 3D:tä ja jopa hyperrealistista grafiikkaa, mutta pelaaja silti vain katsoo pikseleitä. Äänet ovat ainoa “oikea” tunne, jonka pelaaja voi kokea, vaikka ääni ei ole oikeaa linnunlaulua tai aseen laukauksia, mutta ääni on oikean äänen nauhoitusta tai todella oikeanmukaista jäljennystä. Oikea tai ei, kokemus on sama ja äänillä on tärkeä rooli. (Isaza 2010.)

Äänitehosteiden tärkeyden huomaa, kun pelaa ilman niitä. Esimerkiksi käyttäjäliittymässä pelaajalla ei ole auditiivista palautetta napin painalluksesta, jolloin pelaaja ei saa tärkeää vihjettä tai tunteiden vahvistusta. Hyvä pelikokemus immersioi pelaajan täysin, ja esimerkiksi pelkkä napinpainalluksesta lähtevä ääni vahvistaa pelaajan tunnetta siitä, että hän tekee jotain. (Isaza 2010.)

Äänitehosteiden hyvä ymmärtäminen auttaa pelin tavoitteiden käsittämisessä. Laaja tietämys äänitehosteista ohjaa ja auttaa tuotantoryhmää ja äänisuunnittelijaa luovassa prosessissa hyödyntämään täysin äänimoottorin ominaisuuksia ja käyttämään tehokkaasti ääntä. On tärkeää, että kehittäjä ymmärtää, mitkä

alueet pelimaailmassa edellyttävät ääntä sekä niiden käyttötarkoitusta. Ne auttavat äänituotannon menestykselliseen lopputulokseen. (Isaza 2010.)

4.1.1 Äänitehosteiden käyttö

Videopelin tärkein tehtävä on viihdyttää, ja äänitehosteet ovat tärkeitä siinä. Räjähteiden, laukausten ja autojen törmäysten äänet lisäävät pelin viihdearvoa, ja hyvät äänet saavat pelaajan odottamaan, milloin he pääsevät uudestaan kuulemaan niitä. (Isaza 2010.)

Äänitehosteet auttamat luomaan oikean tunnelman tavallisesta napin painalluksesta aina pelimaailman äänimaailmaan. Esimerkiksi lapsille suunnatuissa peleissä piirretyistä tutut äänitehosteet toimivat paremmin kuin esimerkiksi kauhupeleissä, jotka käyttävät hyödykseen tunnelmaltaan pimeitä ja aavemaisia ääniä. (Isaza 2010.)

Realismin luomiseen esimerkiksi Battlefield käyttää pelimaailmansa aikaan sopivia ääniä, jotka lisäävät aidontuntuisuutta ja auttavat pelaajia tuntemaan, että he osallistuvat pelin aikakauden konflikteihin. Aseiden ja ajoneuvojen äänet on suunniteltu olemaan mahdollisimman autenttiset oikeiden aseiden ja ajoneuvojen kanssa. Taustaäänet on tehty niin, että pelaajalle koko ajan kuulostaa siltä kuin olisi taistelu käynnissä. (Isaza 2010.)

Monet ensimmäisen persoonan ammuskelu käyttävät hyödykseen ääniä antaakseen pelaajalle vihjeen ympäristöstään ja muista tapahtumista lähiympäristössä. Tämä näkyy esimerkiksi siten, että vesiputous kuuluu vaimeasti ensin, mutta kun pelaaja pääsee lähemmäksi, se alkaa kuulua kovemmin selkeästi havaittavasta kohdasta. (Isaza 2010.)

4.1.2 Äänitehosteet käyttöliittymissä ja brändinä

Kosketusnäytön ja liitäntäpalautteen luominen on vaikeaa, mutta niillä on tärkeä tehtävä virtuaalisessa maailmassa. Pelkkä valojen päälle laittaminen luo pienen äänen oikeassa elämässä, mikä antaa tärkeän palautteen. Nämä äänet ovat tärkeitä pelimaailmassa myös, sillä ne antavat pelaajalle palautteen siitä, että hänen tekonsa teki jotain, mitä ei voi aina visualisoida. Konsolit antavat myös äänipalautteen napin painalluksissa ja ruutujen vaihtuessa. (Isaza 2010.)

Äänitehosteita voi myös käyttää brändin luomiseen, ja jokainen peli pyrkii tuoreeseen ja innovatiiviseen identiteettiin. Kehittäessään pelin alkuperäistä ulkoasua, tuntumaa ja ääntä kehittäjät yrittävät luoda tunnistettavan brändi-identiteetin, joka määrittelee pelin identiteetin. Näin kuka tahansa, joka näkee taidetta tai kuulee äänen, voi heti tunnistaa kyseisen pelin. Monet pelit, kuten Halo, World of Warcraft ja Mario, tunnistetaan helposti niiden äänien kautta. Esimerkiksi Halon ja World of Warcraftin tunnusmusiikki on hyvin ikoninen, kun taas Mario-pelit tunnistetaan tunnusmusiikin sekä Marion liikkumisen aikana kuuluvien äänitehosteiden avulla, sillä samoja äänitehosteita on käytetty ensimmäisestä pelistä lähtien. (Isaza 2010.)

4.2 Äänitehosteiden lisääminen

Jokaisessa videopelissä on tietty alue, jossa äänitehosteet ovat pakollisia, ja näitä esiintyy aloitusruuduissa, käyttöliittymissä ja pelattavuudessa. Jokainen videopeli yksinkertaisista palapeleistä massiivisiin moninpeleihin käyttää ääniä näissä paikoissa. (Isaza 2010.)

Käynnistäessään pelejä pelaajat kohtaavat erilaisia animoituja logoja julkaisijoilta, kehittäjiltä ja muilta luovilta tahoilta. Näiden tarkoitus on markkinoida yhtiöitä, jotka ovat olleet pelin kehityksessä mukana, mutta näillä on myös toinen tarkoitus. Pelaajan tietämättä kehittäjä luo näyttämön sekä jännitystä, asettaa tunnelman ja mikä tärkeintä, varmistaa, että äänet ovat päällä. (Isaza 2010.)

Näiden äänien ei tarvitse olla mitenkään erityisiä, vaan pelkät yksinkertaiset äänet, kuten hitaat “whoosh”- ja yksinkertaiset “klik”-äänet riittävät. Tämä ei tarkoita, että pelaajaa ei voi houkutella avausnäytöllä. (Isaza 2010.).

4.2.1 Äänet välielokuvissa ja käyttöliittymissä

Avaus-, loppu- ja siirtymäelokuvat tunnetaan nimellä cinematic tai välikohtaus. Ne luovat tarinantaustat, tunnelman, liikuttavat tarinaa eteenpäin, antavat tarvittavia vihjeitä sekä ylistävät pelaajaa tason läpipääsystä. Yleensä niitä löytyy pelin alusta, tasojen välistä ja pelin lopusta. Välikohtauksia esiintyy myös eri “pomokohtauksien” aikana, jolloin ne korostavat tarinan muutosta sekä lisäävät jännitystä. Nämä pienoiselokuvat eivät yleensä ole interaktiivisia, jolloin ne antavat pelaajalle hyvän hengähdystauon. (Isaza 2010.)

Jokainen iso pelijulkaisu käyttää voimakkaita pienoiselokuvia luodakseen pelin näyttämön. Näiden elokuvien äänet ovat yleensä aina kaikista parhaimpia, ja ne ovat yleensä tiimissä kaikista kokeneimpien äänisuunnittelijoiden kädenjälkeä. (Isaza 2010.)

Pelissä olevat välielokuvat liikuttavat tarinaa eteenpäin tiivistämällä, mitä pelaaja on saavuttanut, vihjaamalla, mitä on tulossa. Nämä elokuvat ovat yleensä dramaattisia ja ovat riippuvaisia äänistä luodakseen tunnelman. Pelaaja investoi paljon aikaa peleihin, joissa täysi läpäisy on iso saavutus, joten tästä hyvästä palkinto on yleensä loppuelokuva. Taidokkaasti tehty elokuva jättää pelaajalle hyvän olon voitosta yleensä samalla saattaen pelin juonen loppuun. (Isaza 2010.)

Luovasta näkökulmasta nähtynä äänitehosteet tuotetaan ja integroidaan tähän esikirjoitettuun välineeseen samalla tavalla kuin perinteisissä elokuvissa. Taustaympäristön tunnelman luovat äänet ja muut tarvittavat äänet on luotu käyttäen erilaisia tekniikoita, joista tulee lopuksi muokattu äänitiedosto. Äänitehosteet sekoitetaan taustamusiikkiin ja vuoropuheluun niin, että äänenvoimakkuutta, panorointia ja tasausta säädetään oikein, jolloin ne on tarkasti synkronoitu animaatioiden kanssa, jotka katsoja näkee.

Äänet priorisoidaan kunkin kohtauksen aikomusten säilyttämiseksi, jolloin yleinen äänimaailma on hallittavissa ja ymmärrettävissä. Lisäksi äänet palvelevat erityistä tarkoitusta lisäämällä uskottavuutta ja realismin tuntemusta tai vain viihdettä. (Isaza 2010.)

Valikkonäytöt voivat olla yksinkertaisia tai monimutkaisia riippuen pelin tarpeista; näille alueille luotavat äänitehosteet ovat yleensä melko hienovaraisia ja vastaavat aina pelin teemaa. Näille alueille tyypillisiä ovat näppäinäänet, tunnelma- ja ympäristöäänet, hälytykset, huomiosignaalit ja muut äänimerkit. Musiikki ja satunnaiset vuoropuhelut kuuluvat myös äänirakenteeseen, ja näiden elementtien on toimittava yhdessä. (Isaza 2010.)

4.2.2 Maailmaa ja tunnelmaa luovat äänitehosteet

Taustaympäristö ja ympäristön äänitehosteet antavat elämän peliin. Kokonaisvaltainen immersio on tavoitteena, nämä äänet eivät ainoastaan lisää realismin tuntemusta visuaaleille vaan myös peittävät äänet pelaajan olohuoneessa tai makuuhuoneessa. Olipa kyse kohtauksesta hektisellä kaupunkialueella tai hiljaisella maaseudulla, äänet, jotka osoittavat toiminta-alueelle, on välttämättömiä. (Isaza 2010.)

Yleistä tunnelmaa täydentävät usein erityiset ympäristöäänet merkittävistä esineistä ja asioista, jotka ovat alueella. Kaupunkikuvaan kuuluu tunnelma, johon sisältyy kaupungin yleiset taustaäänet, kuten liikennemelu, rakentamisen äänet ja lentokoneiden äänet. Ympäristöäänet kuvaavat kohteita, joita pelaaja kohtaa alueiden tutkimisen aikana. Roskasäiliön sisällä oleva tuli, katulamppu tai suihkulähde ovat mahdollisia melua aiheuttavia esineitä, joita pelaajat odottavat kuulevansa. (Isaza 2010.)

Tiettyjen esineiden äänet nauhoitetaan usein kentällä, mutta ne voidaan myös tallentaa studiossa, jos on mahdollista. Niiden kohteiden osalta, jotka ovat hankalia tai epäkäytännöllisiä tallentaa, voidaan käyttää muita kohteita niiden sijaan tai luoda omia ääniä sekä ottaa ääniä ilmaisista tai maksullisista äänikirjastoista. (Isaza 2010.)

4.2.3 Tärkeimmät vuorovaikutusäänitehosteet ja pelaajien palauteäänitehosteet

Keskeiset vuorovaikutukset ja pelaajan vaikutukset ovat minkä tahansa pelikokemuksen sisältö, jolloin niiden aiheuttamat äänet ovat keskipisteenä. Näitä ovat aseiden laukaukset ja räjähdykset, joita pelaaja kuulee ammuskelupeleissä. Autopeleissä näitä ovat moottorien, renkaiden ja muiden autojen äänet. Fantasiapeleissä ja historiallisissa peleissä näitä voivat olla miekan voimakas kalina tai maaginen tulipallo. Nämä äänet pelaaja "tuntee", kun hän ovat vuorovaikutuksessa virtuaalisen kokemuksen kanssa. Vaikka musiikki, tunnelmaa luovat äänet ja vuoropuhelu ovat tärkeitä, peli ei voi eikä saa koskaan olla ilman näitä edempänä mainittuja elintärkeitä äänielementtejä. (Isaza 2010.)

Nämä äänet tarjoavat ensisijaisesti viihdettä. Ne ovat epäilemättä iso osa hauskoja pelejä. Joskus ääni on niin täydellinen, että se yksinään antaa suurta tyydytystä pelaajalle. Ihmiset pelaavat pelejä huvikseen, ja äänisuunnittelijalla on erinomainen tilaisuus toimittaa mahtavia ääniä. (Isaza 2010.)

Toiseksi, nämä keskeiset äänet antavat pelaajalle palautetta niiden käytönaikaisista toiminnoista. Äänet eivät ainoastaan vahvista, että jotain tehdään, mutta tuottavat myös muita hienoja vihjeitä. Esimerkkinä pelaaja siirtyy avaamaan suljettua ovea. Odotuksen mukaan ovenkahva tai toimilaitteen mekanismi tekee äänen – mutta onko ovi auki vai lukittu selviää kuuntelemalla, mitä kuuluu. Äänet täydentävät oviaukon ulkoasua avautumalla tai jäämällä kiinni, jolloin pelaajalla on tarvittavat tiedot mitä tehdä seuraavaksi. (Isaza 2010.)

4.2.4 Äänien luominen keksityille esineille

Äänien luominen kohteille, jotka eivät todellisuudessa ole olemassa, voi olla haaste, jos haluaa pitää äänet sopivana ja silti herättää uskottavuuden tunteen. Tyypillisesti äänet perustuvat todellisiin esineisiin, kuten miekkoihin tai pistooleihin, mutta niitä manipuloidaan antamaan niille tuntematonta laatua, joka saattaa kuulijan mielikuvitusympäristöön. Jokainen miekkaisku tai pistooli on tuttu pelaajalle samalla, kun äänet kuulostavat erilaiselta. (Isaza 2010.)

Kun pelissä on esineitä, jotka eivät perustu todellisuuteen, äänisuunnittelijan avain on saada ne kuulostamaan kuin ne olisivat peräisin kyseisestä objektista. Pulmanselvityspelit, kuten Bejeweled, käyttävät arkade-tyyliääntä tehokkaasti viihdyttämällä pelaajaa ja tarjoamalla samalla tuntuvaa palautetta ja vihjeitä siitä, mitä pelissä tapahtuu. Kun pelikappaleet on sovitettu tai bonusesitteet näytetään, ääni antaa tärkeitä vihjeitä. Tällaiset pelit ovat erittäin riippuvuutta aiheuttavia johtuen taidokkaasti sovelletuista äänitehosteista. (Isaza 2010.)

4.2.5 Äänten kategorointi

Dynaaminen ääni on sitä, kun ääni reagoi pelin muutoksiin, esimerkiksi pelimaailman tai pelaajan syötteeseen. Dynaamiseen ääneen kuuluu myös mukautuva ääni, kun pelimaailma reagoi pelattavuuteen. Vuorovaikutteinen ääni on taas sitä, kun ääni reagoi pelattavuuteen pelaajan vuorovaikutuksesta. (Quinn 2008.)

Diegetic-ääni on ääni, joka näkyy ruudulla tai ääni äänenlähteestä, jonka on selvästi kerrottu olevan lähistöllä, esimerkiksi hahmojen äänet, objektien tekemät äänet tai musiikki, joka tulee itse maailmasta. Ei-dynaaminen diegetic-ääni on ääni, joka kuuluu pelaajan lähistöllä, mutta pelaaja ei voi vaikuttaa siihen. Mukautuva diegetic-ääni on se, kun ääni reagoi pelimaailman muutoksiin mutta ei pelaajan muutoksiin. Vuorovaikutteinen-diegetic ääni, joka tapahtuu pelaajan lähellä ja johon pelaaja voi vaikuttaa itse, esimerkiksi askelten äänet tai aseiden vaikutus. (Quinn 2008.)

Ei-diegetic-ääni viittaa taustamusiikkiin ja äänitehosteisiin. Mukautuva ei-diegetic-ääni tarkoittaa ääniä, jotka vaikuttavat pelattavuuteen mutta eivät ole pelaajan vaikuttamia, eivätkä ole osa pelimaailmaa, eivätkä ole peli hahmon kuultavia, joita ovat esimerkiksi pelimusiikki tai äänitehoste. Ei-dynaaminen ääni ovat ääniä, joihin pelaaja ei voi itse vaikuttaa mitenkään. Tällaisia ovat esimerkiksi äänet, ei-interaktiivisen välikohtauksen aikana. (Quinn 2008.)

Kineettinen gestural-vuorovaikutus viittaa Diegetic- tai ei-diegetic-äänen, joihin pelaaja ja pelihahmo voi osallistua fyysisesti ja joiden äänenlähde näkyy ruudulla. Tästä esimerkkinä on ohjaimen käyttäminen pelimaailmassa olevan instrumentin soittamiseen. (Quinn 2008.)

Äänet voidaan myös jakaa yksinkertaisemmin. Alue viittaa erinäisiin ympäristön aiheuttamiin ääniin, jotka ovat diegetic ja luovat tunnelman. Tehosteet viittaavat diegetic-ääniin, jotka pelattavuus aiheuttaa. Esimerkiksi askelten, räjähdysten ja aseiden äänet, äänet ovat tällaisia ja ne voivat olla ruudulla tai ruudun ulkopuolella. Vaikutus viittaa ei-diegetic-ääniin, jotka luovat pelin tunnelman, ja nämä voivat olla orkesterimusiikkia tai matalaäänisiä sävyjä. Käyttäjäliittymä on ei-diegetic-ääniä, esimerkiksi valikon äänet, ja näiden äänien tarkoitus ei ole luoda tunnelmaa, vaan antaa informaatiota. (Quinn 2008.)

Ensimmäisessä esimerkissä peliäänien kuvaamisessa on ehkä liikaa eri kategorioita ja on hankala laittaa tiettyjä ääniä yhteen kategoriaan. Toinen esimerkki äänien jakamisesta on yksinkertaisempi ja siinä on helpompi jakaa komplekseja ääniä eri kategorioihin. (Quinn 2008.)

5 PELIN LUOMINEN

Projektin aluksi on hyvä miettiä tavoitteet ja suunnitella, kuinka toteuttaa ne. Selvä konsepti tavoitteista helpottaa toteutusta. Projektin saavuttamiseksi on tietyt tavoitteet, jotta pelistä tulee halutunlainen ja hyvä. Hyvä peli on kiinnostava sekä siinä on hyvä pelattavuus ja pelisuunnittelu. Pelimoottorin ja kunnollisen suunnitelman kanssa on helppo luoda peli.

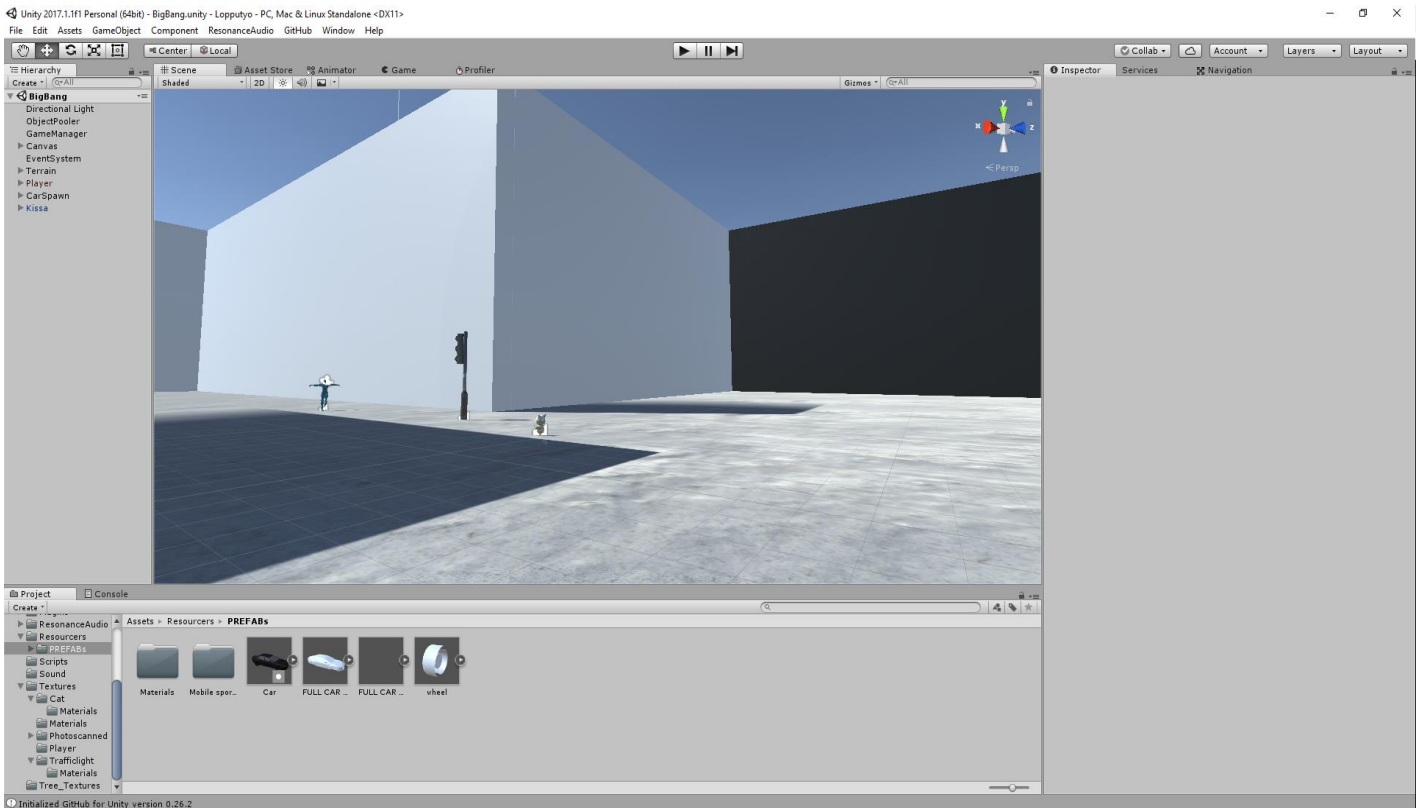
Projektin luomisen helpottamiseksi ensimmäinen asia oli pelisuunnitelma. Pelisuunnitelman tarkoitus on nopeuttaa pelin toteutusta ja välttää hidasteita, jotka saattaisivat hidastaa pelin luontia. Pelisuunnitelma on dokumentti, joista on useita valmiita pohjia, jotka ovat pelikohtaisia ja eivät sovi jokaiseen projektiin.

Pelinkehityksessä tarinan ja grafiikan lisäksi on tärkeää pelin koodi. Pelin koodi vaikuttaa pelattavuuteen: huono koodi pilaa pelin täysin, eikä mikään pysty pelastamaan sitä. Pelin ohjelmointiin käytettiin Microsoftin Visual Studiota, joka on suositeltu editori Unityssä, joka käyttää C#-ohjelmointikieltä.

5.1 Uusi projekti ja kehitysympäristö

Uuden projektin luomiseksi valitaan “New” oikeasta yläkulmasta, kun Unity avataan. Seuraavaksi valitaan kiintolevyltä tai pilvestä projektille sijainti, johon se halutaan tallentaa ja se nimetään. Koska projekti on 3D, se valitaan valinnan alapuolelta.

Kuva 1 näyttää Unityn käyttäjäliittymän perusnäkymän, joka on vapaasti muokattavissa.



KUVA 1. Unityn käyttäjäliittymän perusnäkymä

Tärkeimmät osat ovat skene-näkymä, hierarkiaikkuna, projekti-ikkuna ja inspector-ikkuna. Skene-näkymä on editointi-ikkuna, jossa voi muokata 3D- tai 2D-maailmaa ja peliobjekteja. Pelinäkymässä maisema eli scene näkyy kuin se näkyisi pelimaailman kameran kautta. Hierarkiaikkunassa näkyy lista kaikista pelimaailman objekteista. Projekti-ikkunassa on kaikki projektiin kuuluvat assetit ja asseteille voi tehdä omat kansiot. Inspector-ikkunassa säädetään peliobjektien ominaisuuksia ja komponentteja. Lisäksi työkalupalkissa ylärivissä on näkymän liikutteluun työkaluja. Ylärivistä löytyy myös Play-painike, jonka kautta projektia voidaan testata pelinäkymäikkunassa.

5.1.1 Assetit

Assetit ovat kaikki peliobjektit, skriptit tai muut resurssit, joita käytetään Unityssä. Assetteja voidaan tehdä itse tai hakea esimerkiksi Unityn Asset Storesta, joka löytyy Unityn omilta sivuilta tai Unity-ohjelman kautta, josta valitaan Window ja sieltä Asset Store. Tämän projektin 3D-maailmassa käytettiin Unityn Asset Storesta sekä muualta hankittuja 3D-malleja.

5.1.2 Objektit ja komponentit

Skenet sisältävät kaksi objektia alustavasti, Main Cameran ja Directional Lightin. Main Camera on skenen pääkamera, jonka kautta pelinäkymä renderöidään. Directional Light -objekti tuottaa skenen valaistuksen.

Unityssä on valmiina mahdollisuus lisätä valmiita 3D-objekti muotoja, kuten kuutioita, sylintereitä, tasoja ja palloja, mutta varsinaista mallintamista sillä ei voi tehdä. Objektit voidaan lisätä skeneen valikosta valitsemalla GameObject, sieltä 3D Object ja haluttu objekti. Objekteihin voi lisätä komponentteja ja se sisältää jo valmiiksi seuraavat komponentit: Transform, Mesh, Collider ja Renderer. Transform määrittelee objektin koon, asennon sekä sijainnin 3D-avaruudessa XYZ-koordinaatteina. Mesh on 3D-objektin polygonimalli. Collider määrittelee fyysiset törmäysrajat, collider mahdollistaa objektin törmäämisen toiseen colliderilla varustettuun objektiin. Collider lisätään objektin ympärille käyttäen 3D-perusmuotoja, kuten kuutioita, palloja ja sylintereitä. Colliderin voi tehdä myös tarkasti 3D-objektin ulkopintoja mukailevaksi Mesh Collidereiksi, mutta tämä on raskasta ja vaikuttaa suorituskykyyn alentavasti. Render piirtää objektin ruudulle, ja ilman sitä objekti on näkymätön tarvittaessa. Komponetteja voidaan lisätä Add Component –painikkeella.

5.2 Tavoite ja suunnittelu

Pelisuunnitelmani pääidea oli luoda immersiivinen äänipeli näkörajoitteiselle henkilölle. Tärkein oli, että pelaaja ymmärtää helposti, miten ääni auttaa häntä kulkemaan peliympäristössä ja tajuamaan ympäristön rajoituksia. Erityisesti vältetään tilanteita, joissa pelaaja ei tiedä, minne pitäisi suunnata. Pelin ei tarvinnut olla pitkä, mutta se oli minun kirjoittamani, kun taas opinnäytetyön toimeksiantaja antoi ideoita käsikirjoitukseen. Pelinsuunnittelu oli helppoa ja siinä käytin hyödyksi aikaisempaa kokemustani peleistä. Myös pieni taustatutkimus tehtiin äänipeleistä ja selvitettiin, mitä ne tarvitsevat.

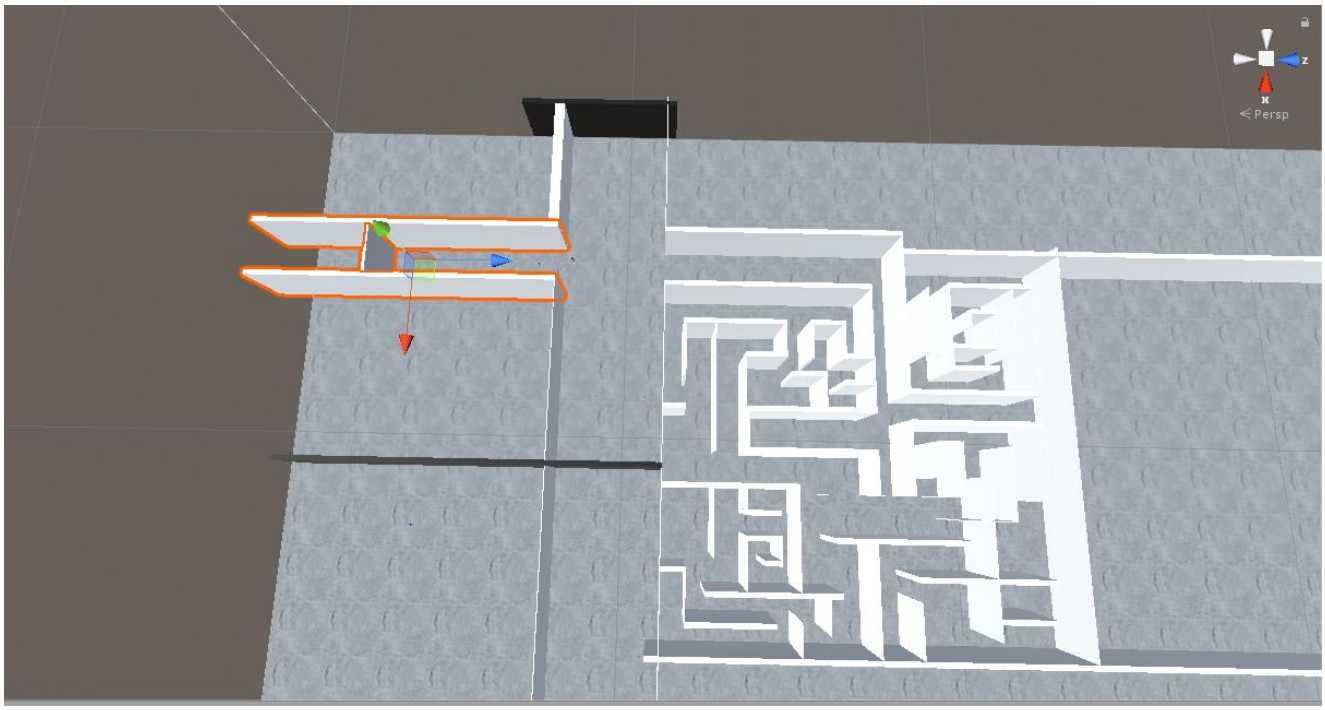
Tämän projektin suunnitelmissa otettiin huomioon tavoitteet alusta lähtien. Pääidean lisäksi otettiin myös huomioon, että peli on kiinnostava ja immersiivinen pelaajalle. Tätä varten pelattavuus ja pelisuunnittelu sekä ohjelmointi otettiin huomioon erityisesti.

Tekstuureiden ja modeleiden löytäminen peliprojektin grafiikkaa varten, oli yksi hankalimmista osioista. Käyttämällä Unityn nettikauppaa sekä useita muita sivuja, joissa oli vapaasti käytettäviä assetteja, löydettiin tarvittavat grafiikat. Pelattavuutta varten valittiin yksinkertaiset ja useassa pelissä käytetyt kontrollit.

5.3 3D-objektit ja kenttäsuunnittelu

Pelimaailmaan tuotiin ilmaisia internetistä löydettyjä 3D-objekteja. Objektit tallennettiin ensin FBX-muodossa tietokoneelle, minkä jälkeen Unityssä projekti-ikkunassa klikataan hiiren oikealla ja valitaan import asset, joka avaa tietokonekansion, josta voi etsiä halutun assetin. Yksi tärkeä asia, joka pitää ottaa huomioon, kun siirtää assetteja Unityyn, on mittakaava. Unityssä yksikköinä ovat metrit, kun taas useissa muissa ohjelmissa ne saattavat olla ihan muuta, esimerkiksi sentit. Tämä tarkoittaa sitä, että objektit voivat olla paljon pienempiä tai suurempia kuin Unityssä jo olevat objektit. Tämä on helppo korjata vaihtamalla objektin skaalaa transform komponentissä inspector-ikkunassa. Toinen tärkeä osa on, että normaalikartat eivät välttämättä automaattisesti tule osaksi objektin materiaaleja, mutta jos näin käy, ne voi lisätä projekti-ikkunasta valitsemalla haluttu materiaali ja lisäämällä oikea normaalikartta Inspector-ikkunassa olevaan Normal Map –kohtaan.

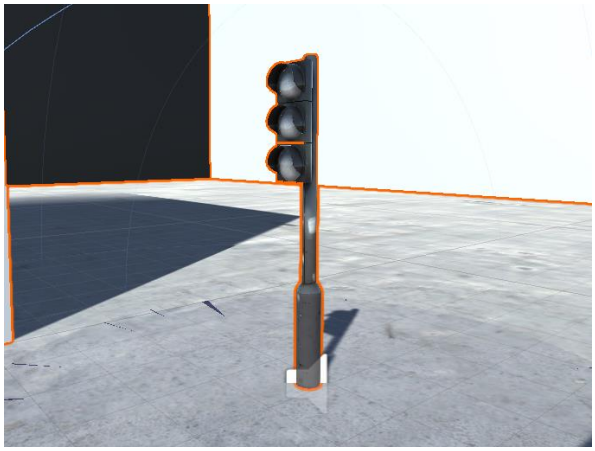
Projektiin lisätiin useita 3D-kuutioita, joista tehtiin seinän näköisiä objekteja (GameObject, josta 3D Object ja Cube). Ne sijoitettiin useaan kohtaan pelimaailmassa näyttämään kaupungin kaduilta (KUVA 2).



KUVA 2. Pelin kenttä

Objekteille lisättiin myös colliderit, jotta pelaajahahmo ei mene läpi. Colliderina käytettiin cube collideria, koska objektien muodot eivät olleet monimutkaisia ja se on suorituskyvyltään kevyin.

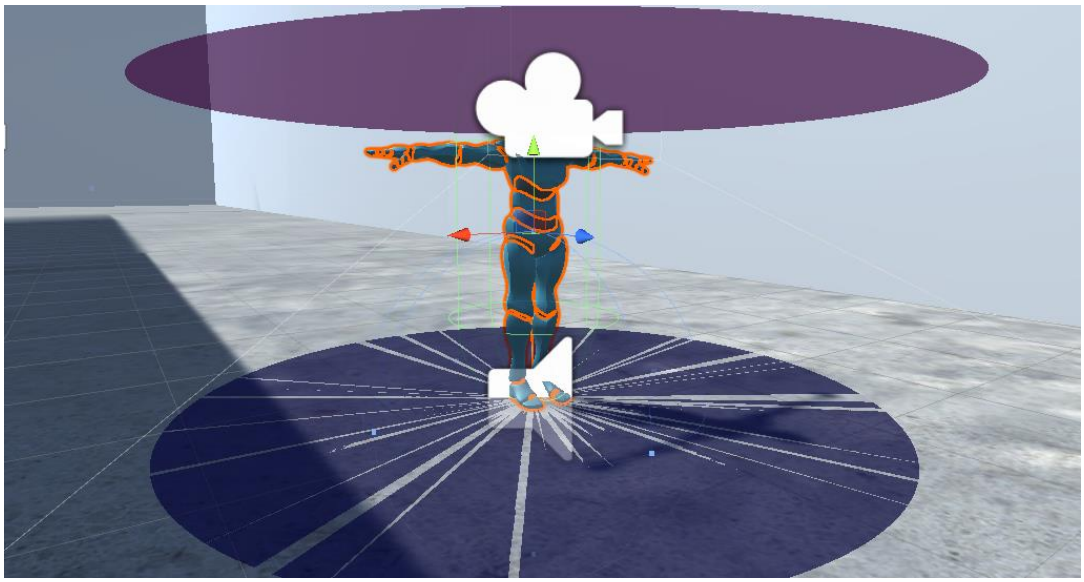
Pelimailmaan lisättiin myös liikennevalot (KUVA 3), lisäämään visuaalista puolta tekemällä siitä mielenkiintoisemman, vaikka tämä ei ole tärkeä tässä projektissa. Liikennevalo-objektille lisättiin yksinkertainen animaatio, joka vaihtaa vihreän, keltaisen ja punaisen valon kokoa tietyn ajan välein vuorotellen antaen vaikutelman oikeasta liikennevalojen vaihtelusta. Kenttäsuunnittelu on loppuen lopuksi hyvin yksinkertainen, ja se koostuu muutamasta huoneesta, joiden on tarkoitus olla katuja, sekä labyrintistä.



KUVA 3. Liikennevalo

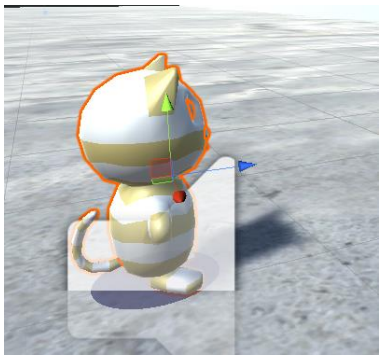
5.4 Pelaaja

Pelaaja on tärkein hahmo pelissä, sillä se antaa käyttäjän olla vuorovaikutuksessa pelimaailman kanssa (KUVA 4).



KUVA 4. Pelaaja skene -näkymässä

Käyttäjä voi kontrolloida pelaajaa erilaisilla syötteillä, jotka tekevät eri asioita, esimerkiksi kävellä, hyppiä tai ampua. Tässä peliprojektissa pelaajan tavoite on saada kissa kiinni (KUVA 5), mikä tulee selväksi heti pelin käynnistyessä soivasta äänidialogista. Pelaajan esteenä ovat liikkuvat autot (KUVA 6), jotka kulkevat, kun liikennevalo on punainen pelaajalle, ja pysähtyvät, kun se on vihreä. Liikennevalon jälkeen tulee labyrintti, jonka pelaaja voi suunnistaa läpi joko kissaa seuraamalla tai kuuntelemalla, missä se on.

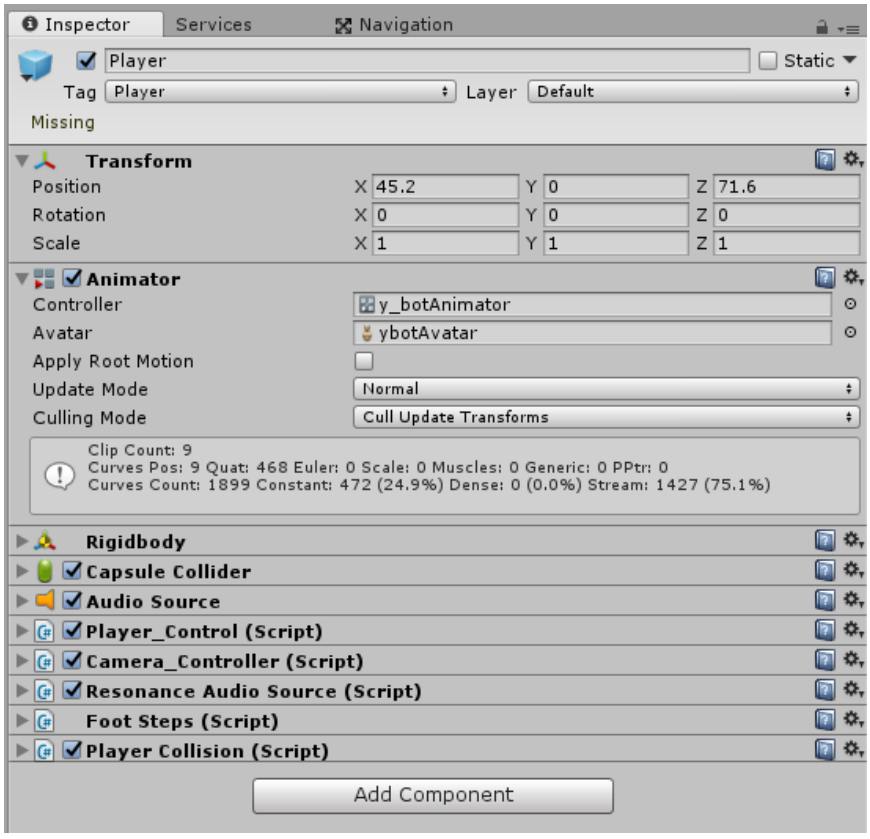


KUVA 5. Kissa skene -näkymässä

KUVA 6. Auto skene -näkymässä

5.4.1 Pelaajan fysiikat

Pelaajahahmoon on tärkeä liittää komponentit RigidBody ja Collider. Tässä projektissa pelaajassa käytettiin capsule collideria, koska hahmo ei tarvitse tarkkaa Mesh Collideria, vaan kevyempi collider käy. Pelaajan liikuttaminen kappaleen kuvassa näkyy pelaajahahmo, ja kuvassa näkyy hierarkiaikkunassa komponenttina RigidBody ja Capsule Collider (KUVA7).



KUVA 7. Inspector -kuva pelaajasta.

Rigidbody sallii fysiikoiden vaikuttamisen pelaajahahmoon, mikä antaa mahdollisuuden liikuttaa hahmoa fysiikan avulla, ja Capsule Collider antaa hahmolle fyysisen ominaisuuden ja sallii törmäyksen muiden Collidereiden kanssa.

5.4.2 Rigidbody

Ridibody-komponentti liittää sen liikuttamisen Unityn fysiikkamoottorille. Jopa ilman koodia Rigidbody-objektiin vaikuttaa painovoima, ja se reagoi törmäyksiin muiden objektien kanssa, jos oikea collider-komponentti on osana. (Unity Rigidbody script reference 2018.)

Rigidbodilla on myös skriptaus-API, joka sallii voimien vaikutuksen objektille ja sen kontrolloinnin fyysisesti realistisella tavalla. Esimerkiksi auton käyttäytymisen voi tehdä lisäämällä fyysisen voiman vaikuttamaan auton pyöriin. Tällä tiedolla fysiikkamoottori voi hallita auton liikkumista, esimerkiksi kiihdyttämällä realistisesti ja saada auto käyttäytymään oikein törmäyksessä. (Unity Rigidbody script reference 2018.)

Skriptauksessa suositellaan käytettävän FixedUpdate-funktiota voimien lisäämiseen ja Rigidbody-asetuksien muuttamiseen Updaten sijaan, joka on tarkoitettu joka toisella framella eli ruudunpäivityksellä päivitykseen. FixedUpdate kutsutaan välittömästi ennen jokaista fysiikkapäivitystä, joten kaikki tehdyt muutokset käsitellään heti jokaisella ruudunpäivityksellä. (Unity Rigidbody script reference 2018.)

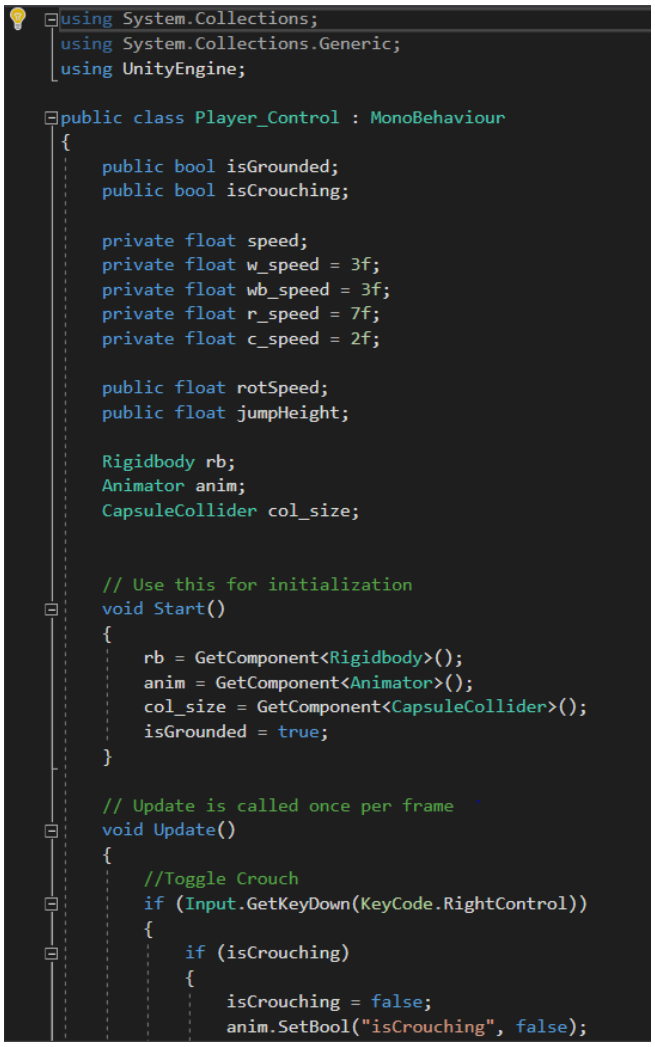
Yleinen ongelma aloitettaessa Rigidbodien kanssa on, että pelin fysiikka näyttää toimivan “hidastetusti”. Tämä yleensä johtuu hahmoissa käytetystä skaalasta. Oletusarvona painovoimaasetus olettaa, että maailman yksi yksikkö vastaa yhden metrin etäisyyttä. Jos suurta skaalaa käyttää objekteilla, joiden tarkoitus on olla pieniä, niiden putoaminen näyttää hidastetulta. Fysiikkamoottori luulee niiden olevan isoja objekteja, jotka tippuvat pitkiä etäisyyksiä. (Unity Rigidbody script reference 2018.)

5.5 Skriptit

Skriptit määrittelevät pelin sisällä tapahtuvat toiminnot. Skripti-tiedostot liitetään peliobjekteihin, ja tällöin objekti, johon skripti on liitetty, saa käyttöönsä skriptissä määritellyt ominaisuudet. Skriptien määrää ei ole rajoitettu ja niitä voi olla yhdessä objektissa useampikin, mutta useampi skripti, joka tekee samaa asiaa, voi aiheuttaa yhteensopimattomuutta. (Creating and Using Scripts 2018.)

5.5.1 Pelaajan liikuttaminen

Pelaajan liikuttamiseen luotiin “Player_Control”-Skripti. Skriptin alkuun on määritelty muutamia komponentteja ja niiden arvoja. Pelihahmon liikkumisnopeudet kävelessä, juostessa, kyykkykävelyssä ja taaksepäin kävelyssä. Myös boolian tilat, kuten isGrounded ja isCrouching, on määritelty, kuten myös Rigidbody, Animator ja CapsuleCollider. Boolian tila isGrounded määrittelee, onko pelihahmo kosketuksissa pelimaailman maan kanssa: jos isGrounded on true, pelaaja voi hypätä, kun taas pelaajan ollessa ilmassa isGrounded on false, jolloin ei voi hypätä enää uudelleen niin kauan kuin pelihahmo on ilmassa. IsCrouching tila määrittelee, onko pelaaja kyykyssä. Pelaajan ollessa kyykyssä isCrouching on true, jolloin pelaaja ei voi hypätä, ja Capsule Collider on yhden metrin pituinen kahden metrin sijaan, jolloin liikkumisnopeus hidastuu. IsCrouching on false, kun pelaaja seisoo pystyssä. Määriteltyjen arvojen ja luokkien jälkeen pelin alussa tapahtuvat asiat ohjelmoidaan void Start() -lausekkeessa. Tässä määritellään pelihahmon Rigidbody, Animator ja Capsule Collider (KUVA 8).



```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Player_Control : MonoBehaviour
{
    public bool isGrounded;
    public bool isCrouching;

    private float speed;
    private float w_speed = 3f;
    private float wb_speed = 3f;
    private float r_speed = 7f;
    private float c_speed = 2f;

    public float rotSpeed;
    public float jumpHeight;

    Rigidbody rb;
    Animator anim;
    CapsuleCollider col_size;


    // Use this for initialization
    void Start()
    {
        rb = GetComponent<Rigidbody>();
        anim = GetComponent<Animator>();
        col_size = GetComponent<CapsuleCollider>();
        isGrounded = true;
    }

    // Update is called once per frame
    void Update()
    {
        //Toggle Crouch
        if (Input.GetKeyDown(KeyCode.RightControl))
        {
            if (isCrouching)
            {
                isCrouching = false;
                anim.SetBool("isCrouching", false);
```

KUVA 8. Player_Control-skripti osa 1

Seuraavaksi on määritelty void Update() -lauseke, joka tarkistaa pelaajan antamia käskyjä jokaisella ruudunpäivityksellä ja tarpeen mukaan päivittää pelihahmon tilaa ja liikuttaa sitä muutosten mukaan. (KUVAT 9-11).

```
            col_size.height = 2;
            col_size.center = new Vector3(0, 1, 0);

        }
        else
        {
            isCrouching = true;
            anim.SetBool("isCrouching", true);
            speed = c_speed;
            col_size.height = 1;
            col_size.center = new Vector3(0, 0.5f, 0);
        }
    }
    var z = Input.GetAxis("Vertical") * speed *Time.deltaTime;
    var y = Input.GetAxis("Horizontal") * rotSpeed;

    transform.Translate(0, 0, z);
    transform.Rotate(0, y, 0);

    /* if (Input.GetKey(KeyCode.Space) && isGrounded == true)
     {
         rb.AddForce(0, jumpHeight, 0);
         anim.SetTrigger("isJumping");
         isCrouching = false;
         isGrounded = false;

     }*/
    if (isCrouching)
    {
        //Crouch controls
        if (Input.GetKey(KeyCode.UpArrow))
        {
            anim.SetBool("isWalking", true);
            anim.SetBool("isRunning", false);
            anim.SetBool("isIdle", false);

        }
        else if (Input.GetKey(KeyCode.DownArrow))
        {
            anim.SetBool("isWalking", true);
            anim.SetBool("isRunning", false);
            anim.SetBool("isIdle", false);
```

KUVA 9. Player_Control-skripti osa 2

```
            }
            else if (Input.GetKey(KeyCode.DownArrow))
            {
                anim.SetBool("isWalking", true);
                anim.SetBool("isRunning", false);
                anim.SetBool("isIdle", false);
            }
            else
            {
                anim.SetBool("isWalking", false);
                anim.SetBool("isRunning", false);
                anim.SetBool("isIdle", true);
            }
        }
        else if (Input.GetKey(KeyCode.RightShift))
        {

            //Running controls
            if (Input.GetKey(KeyCode.UpArrow))
            {
                speed = r_speed;
                anim.SetBool("isWalking", false);
                anim.SetBool("isIdle", false);
                anim.SetBool("isRunning", true);

            }
            else if (Input.GetKey(KeyCode.DownArrow))
            {
                speed = wb_speed;
                anim.SetBool("isWalking", true);
                anim.SetBool("isRunning", false);
                anim.SetBool("isIdle", false);
            }
            else
            {
                anim.SetBool("isWalking", false);
                anim.SetBool("isRunning", false);
                anim.SetBool("isIdle", true);
            }
        }
        else if (!isCrouching)
```

KUVA 10. Player_Control-skripti osa 3

Viimeisenä on määritelty void onCollisionEnter() -lauseke, joka pelihahmon ollessa kosketuksessa maahan muuttaa boolean isGrounded-tilan true-tilaan, joka sallii pelaajan hyppimisen. Player_Control -skriptissä void Update() -lausekkeessa on myös määritelty tunnistus pelaajan napin painalluksista, jotta peli tunnistaa painallukset ja pelihahmon liikuttaminen onnistuu. (KUVA 11).

```
        {

            //Standing controls
            if (Input.GetKey(KeyCode.UpArrow))
            {
                speed = w_speed;
                anim.SetBool("isWalking", true);
                anim.SetBool("isRunning", false);
                anim.SetBool("isIdle", false);

            }
            else if (Input.GetKey(KeyCode.DownArrow))
            {
                speed = wb_speed;
                anim.SetBool("isWalking", true);
                anim.SetBool("isRunning", false);
                anim.SetBool("isIdle", false);
            }
            else
            {
                //speed = w_speed;
                anim.SetBool("isWalking", false);
                anim.SetBool("isRunning", false);
                anim.SetBool("isIdle", true);
            }

        }
    }
        void OnCollisionEnter()
        {
            isGrounded = true;
        }
    }
```

KUVA 11. Player_Control-skripti osa 4

5.5.2 Pelaajan kamera

Camera_Controller -skriptissä määritellään pelaajan käytössä olevan kameran toiminnallisuuksia. Tämä sallii pelaajan katsoa ympärilleen ensimmäisestä kuvakulmasta, minkä lisäksi pelaaja voi kääntää pelihahmon siihen suuntaan. mihin hän haluaa. Pelaaja voi myös katsoa skriptin ja tietokonehiiren avulla horisontaalisesti ja vertikaalisesti tiettyyn pisteeseen asti, joka on määritelty skriptissä. Näin pelaaja ei voi vertikaalisesti esimerkiksi katsoa taakseen vaan pelihahmon pitää kääntyä horisontaalisesti (KUVA 12).

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Camera_Controller : MonoBehaviour

{


    public enum RotationAxis
    {
        MouseX = 1,
        MouseY = 2
    }
    public RotationAxis axes = RotationAxis.MouseX;

    public float minimumVert = -50.0f;
    public float maximumVert = 45.0f;
    public float sensHorizontal = 10.0f;
    public float sensVertical = 10.0f;

    public float _rotationX = 0;

    void Update()
    {


        if (axes == RotationAxis.MouseX)
        {
            transform.Rotate(0, Input.GetAxis("Mouse X") * sensHorizontal, 0);
        }
        else if (axes == RotationAxis.MouseY)
        {
            _rotationX -= Input.GetAxis("Mouse Y") * sensVertical;
            _rotationX = Mathf.Clamp(_rotationX, minimumVert, maximumVert); //Locks vertical angle within min and max.

            float rotationY = transform.localEulerAngles.y;

            transform.localEulerAngles = new Vector3(_rotationX, rotationY, 0);


        }
    }
```

KUVA 12. Camera_Controller-skripti

5.5.3 Pelaajan törmäys

PlayerCollision -skriptissä määritellään, pelaajan törmäämin seiniin, autoihin ja kissaan. (KUVAT 13 JA 14).

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class PlayerCollision : MonoBehaviour
{
    public Player_Control movement;
    //  [SerializeField]

    public GameObject hitByCarText;
    public GameObject victoryText;
    public AudioClip vicSound;
    public AudioClip crash;
    public AudioClip victory;
    public AudioClip coll;
    private AudioSource AudioSource;
    public GameObject target;
    public GameObject trigger;
    public GameObject trigger1;
    public GameObject trigger2;
    public GameObject trigger3;
    public GameObject trigger35;
    public GameObject trigger4;
    public GameObject trigger5;
    public GameObject trigger6;
    public GameObject trigger7;

    //Searches and moves toward the target
    void Start()
    {

        target = GameObject.FindGameObjectWithTag("Target");
    }
    private void Awake()
    {
        AudioSource = GetComponent<AudioSource>();
    }
    void OnCollisionEnter(Collision col)
    {
        if (col.gameObject.tag == "Car") //Collides with Car
        {
```

KUVA 13. PlayerCollision-skripti osa 1

```
        {

            hitByCarText.SetActive(true);
            movement.enabled = false; // Disable player movement.
            FindObjectOfType<GameManager>().EndGame();
            AudioSource.PlayOneShot(crash);

        }
        if (col.gameObject.tag == "Kissa")
        {
            victoryText.SetActive(true); //Victory Text is active when collides
            movement.enabled = false; // Disable player movement.
            FindObjectOfType<GameManager>().EndGame(); // Ends the game
            AudioSource.PlayOneShot(victory);
            AudioSource.PlayOneShot(vicSound);



        }
        if(col.gameObject.tag == "Wall")
        {
            AudioSource.PlayOneShot(coll);
        }
    }
    private void OnTriggerEnter(Collider other)
    {

        //Collides with Triggers.
        if (other.gameObject.tag == "Trigger0")
        {

            trigger.SetActive(false);
            Debug.Log("Trigger 0");
            target.transform.position = new Vector3(89.2f, 1f, 179f);

        }
        if (other.gameObject.tag == "Trigger1")
        {

            trigger1.SetActive(false);
            Debug.Log("Trigger 1");
            target.transform.position = new Vector3(94.5f, 1f, 221.2f);
```

KUVA 14. PlayerCollision-skripti osa 2

Pelaajan törmätessä seinään peli soittaa äänitehosteen. Autoon törmätessä peli soittaa ääniehosteen sekä poistaa pelaajan liikkumisen ja näyttää tiedon tekstinä pelaajalle, että hän on törmännyt autoon. Muutaman sekunnin jälkeen peli alkaa alusta. Pelaajan törmätessä “triggereihin” kissa siirtyy eteenpäin lähemmäs pelin loppua (KUVA 15 ja 16).

```
        }
        if (other.gameObject.tag == "Trigger2")
        {

            trigger2.SetActive(false);
            Debug.Log("Trigger 2");
            target.transform.position = new Vector3(119.53f, 1f, 220.8f);
        }
        if (other.gameObject.tag == "Trigger3")
        {

            trigger3.SetActive(false);
            Debug.Log("Trigger 3");
            target.transform.position = new Vector3(118.39f, 1f, 154.55f);
        }
        if (other.gameObject.tag == "Trigger35")
        {
            trigger35.SetActive(false);
            Debug.Log("Trigger3.5");
            target.transform.position = new Vector3(121.1f, 1f, 151.6f);
        }
        if (other.gameObject.tag == "Trigger4")
        {

            trigger4.SetActive(false);
            Debug.Log("Trigger 4");
            target.transform.position = new Vector3(107.67f, 1f, 117.31f);
        }
        if (other.gameObject.tag == "Trigger5")
        {

            trigger5.SetActive(false);
            Debug.Log("Trigger 5");
            target.transform.position = new Vector3(61.1f, 1f, 115.1f);
        }
        if (other.gameObject.tag == "Trigger6")
        {

            trigger6.SetActive(false);
            Debug.Log("Trigger 6");
            target.transform.position = new Vector3(62.59f, 1f, 151.43f);
        }
```

KUVA 15. PlayerCollision-skripti osa 3

```
        }
        if (other.gameObject.tag == "Trigger7")
        {
            trigger7.SetActive(false);
            Debug.Log("Trigger 7");
            target.transform.position = new Vector3(87.6f, 1f, 152.2f);
        }

    }
}
```

KUVA 16. PlayerCollision-skripti osa 4

Pelaajan törmätessä kissaan peli soittaa äänitehosteen sekä näyttää tiedon tekstinä pelaajalle, että hän on voittanut pelin ja peli aloittaa itsensä alusta

5.5.4 Pelaajan askeleet ja TerrainDetector

TerrainDetectorin tehtävä on havaita, minkälaisella alustalla pelaaja kävelee, ja sen perusteella soittaa erilaisia askelten ääniä. Skripti hakee TerrainDetector()-lausekkeessa terrain-datan ja splatmap-datan ja tallentaa ne, joiden avulla se havaitsee, missä pelaajaa kävelee (KUVA 17 ja 18).

```
using UnityEngine;

public class TerrainDetector
{
    private TerrainData terrainData;
    private int alphamapWidth;
    private int alphamapHeight;
    private float[,,] splatmapData;
    private int numTextures;

    public TerrainDetector()
    {
        terrainData = Terrain.activeTerrain.terrainData;
        alphamapWidth = terrainData.alphamapWidth;
        alphamapHeight = terrainData.alphamapHeight;

        splatmapData = terrainData.GetAlphamaps(0, 0, alphamapWidth, alphamapHeight);
        numTextures = splatmapData.Length / (alphamapWidth * alphamapHeight);
    }

    private Vector3 ConvertToSplatMapCoordinate(Vector3 worldPosition)
    {
        Vector3 splatPosition = new Vector3();
        Terrain ter = Terrain.activeTerrain;
        Vector3 terPosition = ter.transform.position;
        splatPosition.x = ((worldPosition.x - terPosition.x) / ter.terrainData.size.x) * ter.terrainData.alphamapWidth;
        splatPosition.z = ((worldPosition.z - terPosition.z) / ter.terrainData.size.z) * ter.terrainData.alphamapHeight;
        return splatPosition;
    }

    public int GetActiveTerrainTextureIdx(Vector3 position)
    {
        Vector3 terrainCord = ConvertToSplatMapCoordinate(position);
        int activeTerrainIndex = 0;
        float largestOpacity = 0f;

        for (int i = 0; i < numTextures; i++)
        {
            if (largestOpacity < splatmapData[(int)terrainCord.z, (int)terrainCord.x, i])
            {
                activeTerrainIndex = i;
                largestOpacity = splatmapData[(int)terrainCord.z, (int)terrainCord.x, i];
            }
        }

        return activeTerrainIndex;
    }

}
```

KUVA 17. TerrainDetector-skripti osa 1

```
using UnityEngine;

public class FootSteps : MonoBehaviour
{
    [SerializeField]
    public AudioClip[] stoneClips;
    [SerializeField]
    public AudioClip[] mudClips;
    [SerializeField]
    public AudioClip[] grassClips;

    private AudioSource AudioSource;
    private TerrainDetector terrainDetector;

    private void Awake()
    {
        AudioSource = GetComponent<AudioSource>();
        terrainDetector = new TerrainDetector();
    }

     void Step()
    {
        AudioClip clip = GetRandomClip();
        AudioSource.PlayOneShot(clip);
    }

    private AudioClip GetRandomClip()
    {
        int terrainTextureIndex = terrainDetector.GetActiveTerrainTextureIdx(transform.position);

        switch(terrainTextureIndex)
        {
            case 0:
                default:
                return stoneClips[UnityEngine.Random.Range(0, stoneClips.Length)];
            case 1:
                return mudClips[UnityEngine.Random.Range(0, mudClips.Length)];
            case 2:
                return grassClips[UnityEngine.Random.Range(0, grassClips.Length)];
        }

    }
}
```

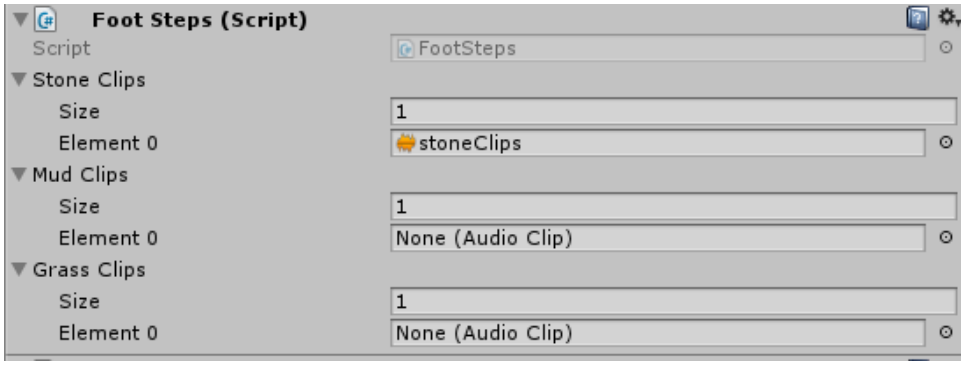
KUVA 18. TerrainDetector-skripti osa 2

Footstep-skriptissä on määritelty taulut eri askeläänityypeille, esimerkiksi kiville, mudalle ja ruoholle. Skriptissä on myös määritelty audio source. Pelin käynnistyessä Awake()-lauseke hakee audio source -komponentin ja terrain detectorin (KUVA 19).

```
using UnityEngine;

public class FootSteps : MonoBehaviour
{
    [SerializeField]
    public AudioClip[] stoneClips;
    [SerializeField]
    public AudioClip[] mudClips;
    [SerializeField]
    public AudioClip[] grassClips;

    private AudioSource AudioSource;
    private TerrainDetector terrainDetector;

    private void Awake()
    {
        AudioSource = GetComponent<AudioSource>();
        terrainDetector = new TerrainDetector();
    }

     void Step()
    {
        AudioClip clip = GetRandomClip();
        AudioSource.PlayOneShot(clip);
    }

    private AudioClip GetRandomClip()
    {
        int terrainTextureIndex = terrainDetector.GetActiveTerrainTextureIdx(transform.position);

        switch(terrainTextureIndex)
        {
            case 0:
                default:
                return stoneClips[UnityEngine.Random.Range(0, stoneClips.Length)];
            case 1:
                return mudClips[UnityEngine.Random.Range(0, mudClips.Length)];
            case 2:
                return grassClips[UnityEngine.Random.Range(0, grassClips.Length)];
        }

    }
}
```

KUVA 19. FootSteps-skripti

Step()-lausekkeessa audio source soittaa satunnaisia askeläänіä, jotka on lisätty tauluun inspector-ikkunassa (KUVA 20).

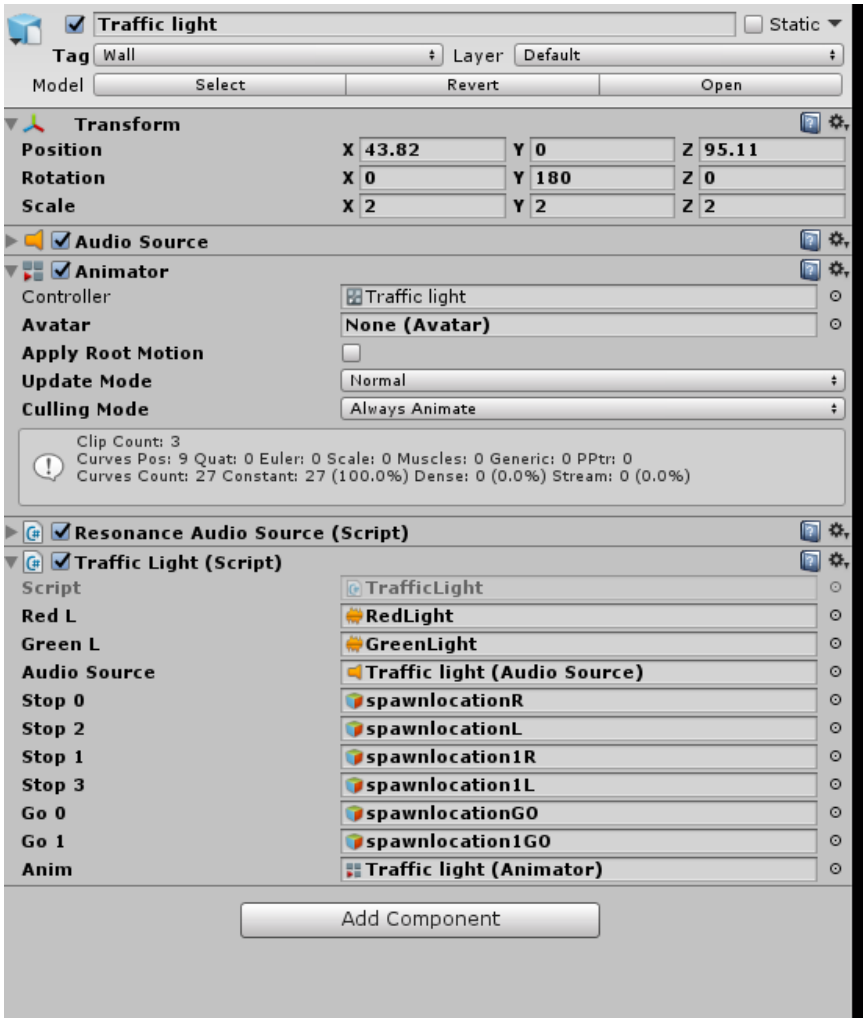


KUVA 20. FootSteps Inspector-ikkunassa

Aina kun liikkumisanimaatiossa pelihahmon jalat koskettavat maata, kutsutaan Void Step()-lauseke (KUVA 19).

5.5.5 Pelaajan esteet

Pelaajan tavoitteena on ottaa kissa kiinni, ja pelaajan esteenä on ensimmäisenä liikkuvat autot, jotka liikkuvat aina, kun liikennevalo on punaisena pelaajalle, ja ne pysähtyvät, kun valo on vihreä. Liikennevaloon on liitetty Animator, Resonance Audio Source- ja TrafficLight-skripti inspector-ikkunassa (KUVA 21).



KUVA 21. Liikennevalot Inspector-ikkunassa

Liikennevalo-skripti toteuttaa liikennevalon animaation sekä aktivoi ja deaktivoi Stop- ja Go-objektit tietyin aikavälein (KUVA 22 ja 23).

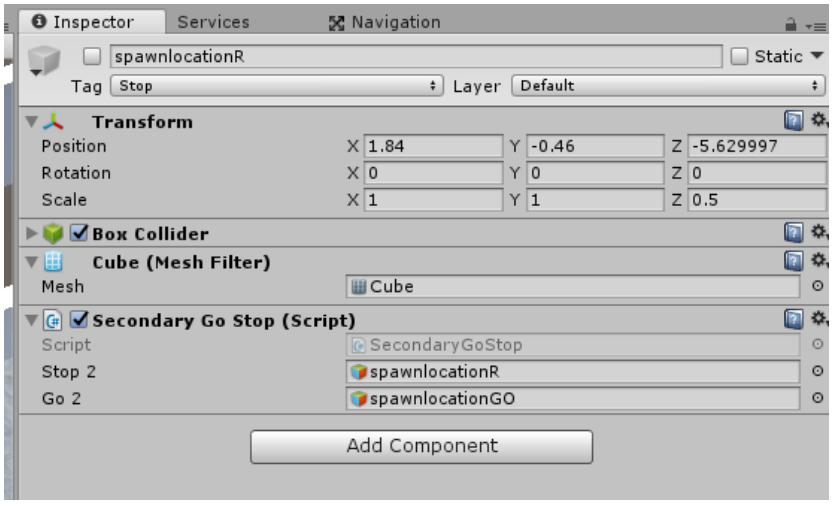
```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
[RequireComponent(typeof(AudioSource))]
public class TrafficLight : MonoBehaviour
{
    public AudioClip redL;
    public AudioClip greenL;
    public AudioSource audioSource;

    public GameObject stop0;
    public GameObject stop2;
    public GameObject stop1;
    public GameObject stop3;

    public GameObject go0;
    public GameObject go1;

    public Animator anim;
    // Use this for initialization
    void Start()
    {

        anim = GetComponent<Animator>();
        StartCoroutine("Trafficlight");

    }
    private void Awake()
    {
        audioSource = GetComponent<AudioSource>();

    }

    private IEnumerator Trafficlight()
    {
        while (true)
        {
            // Playing Green
            anim.SetBool("RedLight", false);
            anim.SetBool("GreenLight", true);
            anim.SetBool("YellowLight", false);
            //Play Greenlight sound
```

KUVA 22. Liikennevalo-Skripti osa 1

```
            audioSource.clip = greenL;

            audioSource.Play();
            // Set Stops active, disable go's
            stop0.SetActive(true);
            stop1.SetActive(true);
            stop2.SetActive(true);
            stop3.SetActive(true);
            go0.SetActive(false);
            go1.SetActive(false);
            yield return new WaitForSeconds(5.0f);
          //  Playing Yellow
            anim.SetBool("RedLight", false);
            anim.SetBool("GreenLight", false);
            anim.SetBool("YellowLight", true);
            yield return new WaitForSeconds(5.0f);
            // Playing Red
            // disable Stops active, set go's active
            anim.SetBool("RedLight", true);
            anim.SetBool("GreenLight", false);
            anim.SetBool("YellowLight", false);

            //Play Red Light Sound
            audioSource.clip = redL;
            audioSource.Play();
            stop0.SetActive(false);
            stop1.SetActive(false);
            stop2.SetActive(false);
            stop3.SetActive(false);
            go0.SetActive(true);
            go1.SetActive(true);

            yield return new WaitForSeconds(10.0f);

        }
    }
}
```

KUVA 23. Liikennevalo-skripti osa 2

Spawnlocation-objekteissa on Go- and Stop-skriptit (KUVA 24).



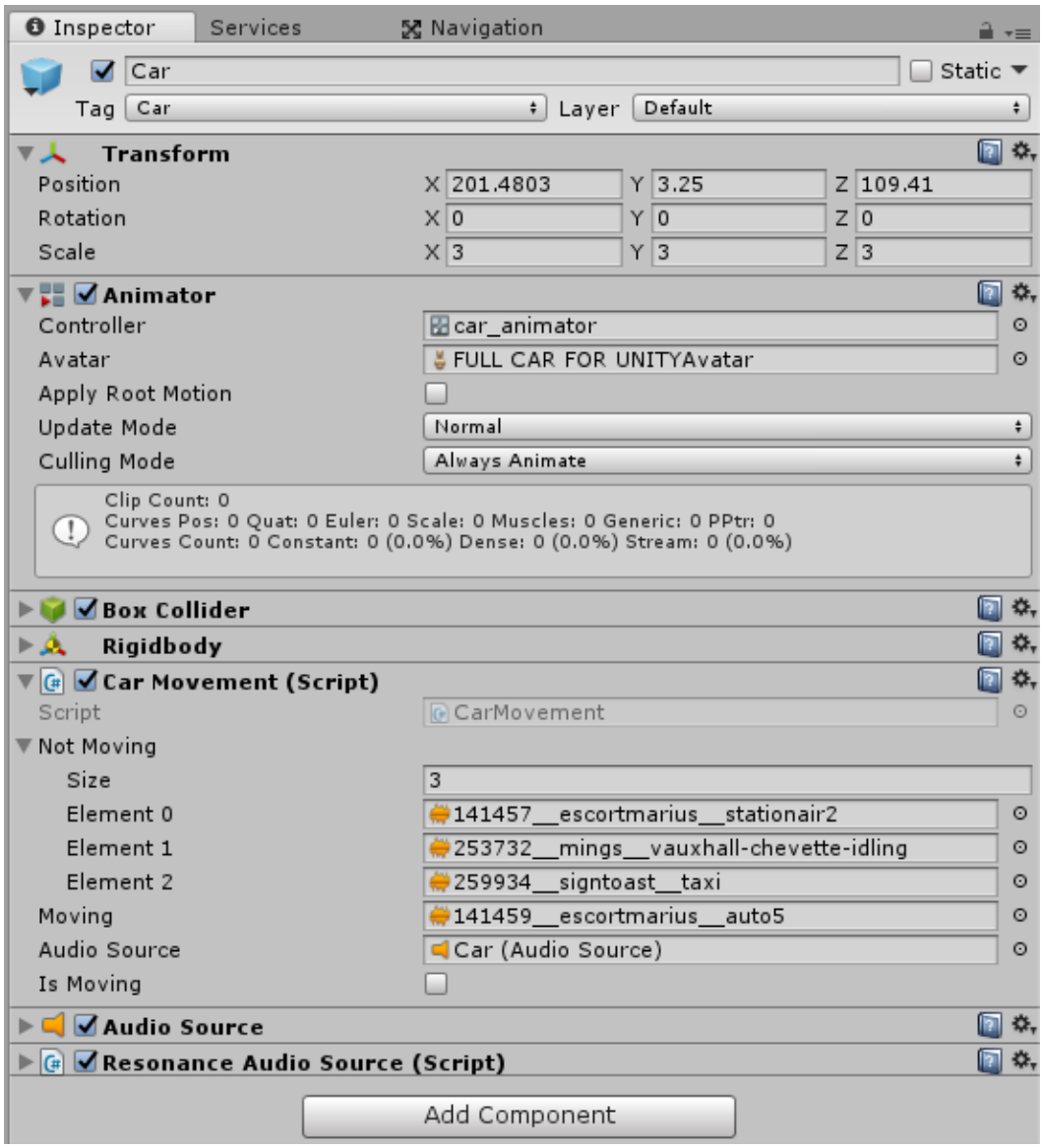
KUVA 24. Go ja Stop spawn -location inspector-ikkunassa

Go-objekti ja skripti ovat aktiivisia, kun liikennevalo on punainen. Objekteissa on yksinkertainen On-TriggerStay Collider, joka auton koskiessa siihen aktivoi sen takana olevan Go-objektin, jossa on samanlainen skripti. Stop-objektissa on samanlainen skripti, mutta se on aktiivisena, kun liikennevalo on vihreä (KUVA 25).

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class SecondaryGoStop : MonoBehaviour
{
    public GameObject stop2;


    public GameObject go2;

    // Use this for initialization
    void Start()
    {

    }

    void OnTriggerEnter(Collider other)
    {

        if (other.gameObject.tag == "Car")
        {

           // Debug.Log("Car on Stop 1");
            stop2.SetActive(true);

            go2.SetActive(false);

        }
    }
}
```

KUVA 25. Esimerkki stop- ja go-skripteistä

Autossa on komponentit Animator, Box Collider, Rigidbody ja skriptit CarMovement ja Resonance Audio Source (KUVA 26).



KUVA 26. Auto Inspector-ikkunassa

CarMovement-skriptin alussa on määritelty auton nopeus, AudioSource, AudioClipit, Rigidbody ja animator, minkä lisäksi boolian tila isMoving on määritelty (KUVA 27).

```
using System;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class CarMovement : MonoBehaviour, IPoolable
{

    public AudioClip[] notMoving;
    public AudioClip moving;
    public AudioSource audioSource;

    public bool isMoving;
    private float CarSpeed;
    private float ForSpeed = 30f;

    Rigidbody CarRb;
    Animator CarAnim;
    public event Action OnDestroyEvent = delegate { };
    // Use this for initialization
    private void OnDisable() { OnDestroyEvent(); }
    public void Start()
    {

        isMoving = true;
        CarRb = GetComponent<Rigidbody>();
        //CarAnim = GetComponent<Animator>();

        // isGrounded = true;

    }
    private void Awake()
    {
        audioSource = GetComponent<AudioSource>();

    }


    // Update is called once per frame

    private void Update()
```

KUVA 27. CarMovement-skripti osa 1

Void Update() -lauseekkeessa on määritelty, että kun isMoving on true, auto liikkuu eteenpäin sekä soittaa auton liikkumisääniä. Auto pysähtyy isMovingin ollessa false, ja mikä soittaa auton tyhjäkäynti-ääniä (KUVA 28).

```
    {
        if (isMoving == true) // If true, Car moves.
        {
            transform.Translate(0f, 0f, ForSpeed*Time.deltaTime);
            //  Is moving
            audioSource.clip = moving;
            audioSource.loop = true;
            audioSource.Play();

        }
        if (isMoving == false) // If false, Car doesn't moves.
        {
            transform.Translate(0f, 0f, 0f); //  Is not moving
            audioSource.clip = moving;

            audioSource.Stop();
            Invoke("RandomNotMovingSound", 5);


        }
    }

    void OnTriggerStay(Collider other)
    {

        if (other.gameObject.tag == "Stop")
        {

            //Collided with Stop set isMoving false
            isMoving = false;
        }
        else
        {
            if (other.gameObject.tag == "Go")
            {
                //Collided with Go set isMoving true
                isMoving = true;
            }
        }
    }
    void RandomNotMovingSound()
    {
```

KUVA 28. CarMovement-skripti osa 2

Auton koskiessa Stop-objektiin void OnTriggerStay -lausekkeessa Stop-objekti muuttaa isMoving-tilan falseksi eli auto pysähtyy. Kun Stop-objekti deaktivoituu, Go-objekti aktivoituu ja auton isMoving-tila muuttuu true-tilaan eli auto liikkuu taas. Autossa on myös void OnCollisionEnter -lauseke, joka auton törmätessä 3D-kuutio-objektiin nimeltä “Wall” eli seinään deaktivoi auton (KUVA 29).

```
    void RandomNotMovingSound()
    {
        if (gameObject.activeInHierarchy)
        {

            audioSource.clip = notMoving[UnityEngine.Random.Range(0, notMoving.Length)];
            audioSource.Play();
        }
    }

    private void OnCollisionEnter(Collision collision)
    {
        if (collision.gameObject.tag == "Wall") // if Collides with wall set the gameObject active false.
        {

            gameObject.SetActive(false);

        }
    }


}
```

KUVA 29. CarMovement-skripti osa 3

Kentän alusta löytyy Car Spawn-peliobjektit, joissa on CarSpawn-skripti. (KUVA 30).

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class CarSpawn : MonoBehaviour
{

    public float spawnCarRate;
    public float spawnCarRepeat;

    [SerializeField]
    private CarMovement prefab;
    private Pool pool;
    public bool invoke = true;


    private void Start()
    {

        pool = Pool.GetPool(prefab);
        SpawnCar();

            InvokeRepeating("SpawnCar", spawnCarRate, spawnCarRepeat);

        }
    private void Update()
    {
        if (invoke == false)
        {
            CancelInvoke("SpawnCar");
        }
    }
    private void SpawnCar()
        {

        var car = pool.Get(transform.position, transform.rotation) as CarMovement;


    }
    }
```

KUVA 30. CarSpawn-skripti

Tämän skriptin tehtävä on tietyin aikavälein luoda auto. Auton luonti tehdään Pool-skriptin kautta, mistä kerron enemmän seuraavassa kappaleessa.

5.5.6 Pool, PoolPreparer ja IPoolable

Nämä kolme skriptiä olivat mielestäni kaikista hankalimmat ja vaativat todella paljon hienosäätöä, jotta sain ne toimimaan oikein. Pool on tekniikka luoda objekteja ennen kuin niitä tarvitaan, mikä vähentää objektien luonti- sekä niiden tuhoamispyynnöt. Tämä vähentää ruudunpäivityksen hidastumista sekä turhaa Unityn “roskien keräämistä” eli garbage collectionia. Skripti antaa halutuille objekteille viitteen Pool-skriptiin, jotka kutsutaan pelin käynnistyksessä. (Unity3d-object-pooling 2017.)

IPoolable-skriptillä on vain yksi tehtävä: se pitää kirjaa objekteista, jotka on tuhottu (KUVA 31).

```
using System;

public interface IPoolable
{
    event Action OnDestroyEvent;
}
```

KUVA 31. IPoolable-skripti

Koska haluamme, käyttää objekteja uudestaan ilman niiden tuhoamista uudelleen luontia, tämä skripti kutsutaan aina, kun objekti kutsutaan OnDisable-lausekkeessa. Kun “poolable”-objekti poistetaan käytöstä, pool-skripti lisää sen automaattisesti saatavalla olevaksi objektiksi ja kun se kutsutaan, se ottaa seuraavan objektin jonosta ja siirtää sen haluttuun paikkaan ja uudelleen aktivoi sen. (Unity3d-object-pooling 2017.)

Suurin osa työstä tehdään Pool-skriptissä. Class alkaa staattisella metodilla, joka hakee ja esilämmittää poolin (KUVA 32).

```
public static Pool GetPool(IPoolable prefab)
{
    if (pools.ContainsKey(prefab))
    {
        if (pools[prefab] != null)
            return pools[prefab];
        else
            pools.Remove(prefab);
    }

    var pool = new GameObject("Pool-" + (prefab as Component).name).AddComponent<Pool>();
    pool.Initialize(prefab);
    pools.Add(prefab, pool);
    return pool;
}
```

KUVA 32. Pool-skripti osa 1

GetPool ja Prewarm tekevät hyvin samaa asiaa. Ensimmäisenä GetPool tarkistaa, onko “pool” jo olemassa prefabille. Jos se on jo olemassa, se palauttaa sen, ja jos se ei ole olemassa tai se on tuhottu, se luo poolin uudelleen ja käynnistää sen, jos se tapahtuu Get()-kutsun kautta, poolin koko on oletusarvo (KUVA 33).

```
public static Pool Prewarm(IPoolable prefab, int initialSize)
{
    if (pools.ContainsKey(prefab))
    {
        if (pools[prefab] != null)
        {
            Debug.LogError("Pool already created, can't prewarm");
            return pools[prefab];
        }
        else
            pools.Remove(prefab);
    }

    var pool = new GameObject("Pool-" + (prefab as Component).name).AddComponent<Pool>();
    pool.Initialize(prefab, initialSize);
    pools.Add(prefab, pool);
    return pool;
}
```

KUVA 33. Pool-skripti osa 2

The Initialize -metodi on vastuussa sellaisten peliobjektien luomisesta, jotka pitää “poolata” (KUVA 34).

```
private void Initialize(IPoolable poolablePrefab, int initialSize = DEFUALT_POOL_SIZE)
{
    this.prefab = (poolablePrefab as Component).gameObject;
    for (int i = 0; i < initialSize; i++)
    {
        var pooledObject = (Instantiate(this.prefab) as GameObject).GetComponent<IPoolable>();
        (pooledObject as Component).gameObject.name += " " + i;

        pooledObject.OnDestroyEvent += () => AddObjectToAvailable(pooledObject);

        (pooledObject as Component).gameObject.SetActive(false);
    }
}
```

KUVA 34. Pool-skripti osa 3

Get-metodi nappaa seuraavan pool-objektin jonosta ja palauttaa sen, jos objekteja ei ole vapaana, Get-metodi kasvattaa pool-kokoa 10 %. Get-metodi myös ottaa paikan ja suunnan huomioon, jotta objekti voidaan lisätä oikeinpäin ennen kuin se aktivoidaan (KUVA 35).

```
    private IPoolable Get()
    {
        lock (this)
        {
            if (objects.Count == 0)
            {
                int amountToGrowPool = Mathf.Max((disabledObjects.Count / 10), 1);
                Initialize(this.prefab.GetComponent<IPoolable>(), amountToGrowPool);
            }

            var pooledObject = objects.Dequeue();

            return pooledObject;
        }
    }

    public IPoolable Get(Vector3 position, Quaternion rotation)
    {
        var pooledObject = Get();

        (pooledObject as Component).transform.position = position;
        (pooledObject as Component).transform.rotation = rotation;
        (pooledObject as Component).gameObject.SetActive(true);

        return pooledObject;
    }
```

KUVA 35. Pool-skripti osa 4

PoolPreparer-skripti lisätään peliobjektiin, joka tarvitsee poolin. Siihen lisätään prefabit, jotka halutaan luoda pelin käynnistyessä, ja suurin osa Awake-lausekkeesta (KUVA 36) on mukana vain varmistaakseen, että poolia ei ladata useasti. OnValidate()-lauseke tarkistaa, että systeemiä ei käytetä vahingossa (KUVA 37). (Unity3d-object-pooling 2017.)

```
using System.Collections.Generic;
using System.Linq;
#if UNITY_EDITOR
using UnityEditor;
#endif
using UnityEngine;

public class PoolPreparer : MonoBehaviour
{
    [SerializeField]
    GameObject[] prefabs;

    [SerializeField]
    private int initialPoolSize = 100;

    private void Awake()
    {
        foreach (var prefab in prefabs)
        {
            if (prefab == null)
            {
                Debug.LogError("Null prefab in PoolPreparer");
            }
            else
            {
                IPoolable poolablePrefab = prefab.GetComponent<IPoolable>();
                if (poolablePrefab == null)
                {
                    Debug.LogError("Prefab does not contain an IPoolable and can't be pooled");
                }
                else
                {
                    Pool.Prewarm(poolablePrefab, initialPoolSize);
                }
            }
        }
    }

    private void OnValidate()
    {
        List<GameObject> prefabsToRemove = new List<GameObject>();
        foreach (var prefab in prefabs.Where(t => t != null))
```

KUVA 36. PoolPreparer-skripti osa 1

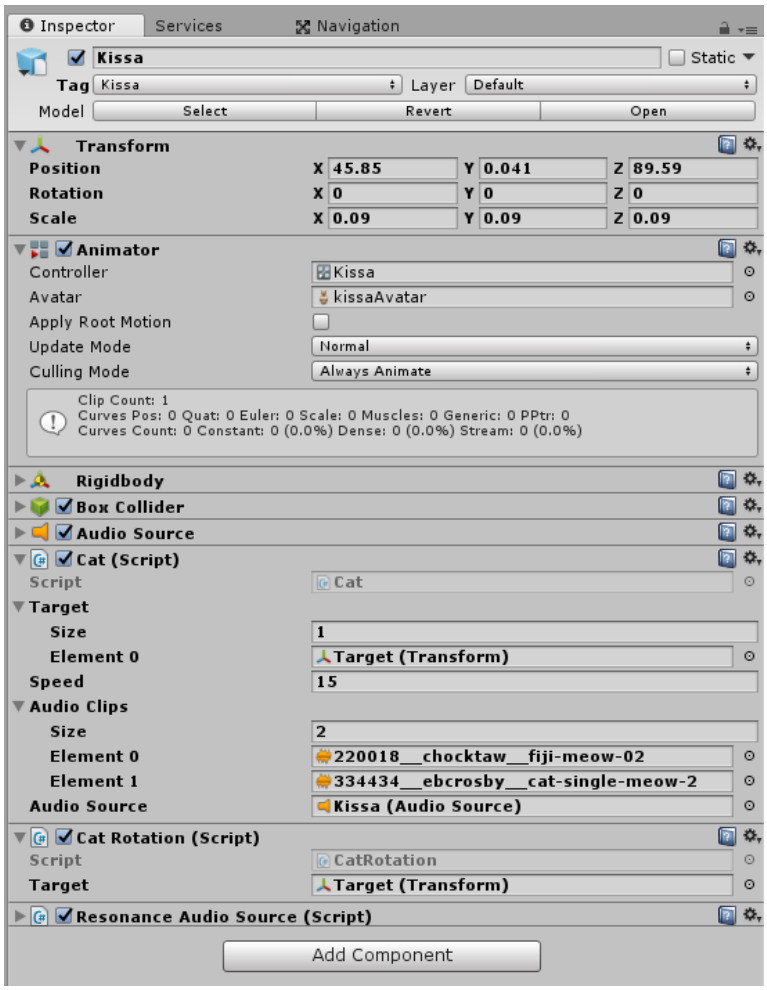
```
        foreach (var prefab in prefabs.Where(t => t != null))
        {
#if UNITY_EDITOR
            if (PrefabUtility.GetPrefabType(prefab) != PrefabType.Prefab)
            {
                Debug.LogError(string.Format("{0} is not a prefab.  It has been removed.", prefab.gameObject.name));
                prefabsToRemove.Add(prefab);
            }
#endif
            IPoolable poolablePrefab = prefab.GetComponent<IPoolable>();
            if (poolablePrefab == null)
            {
                Debug.LogError("Prefab does not contain an IPoolable and can't be pooled.  It has been removed.");
                prefabsToRemove.Add(prefab);
            }
        }

        prefabs = prefabs
            .Where(t => t != null && prefabsToRemove.Contains(t) == false)
            .ToArray();
    }
}
```

KUVA 37. PoolPreparer-skripti osa 2

Tämä pool-systeemi on hyvin yksinkertainen ja todella kevyt. Systeemi ei ole muokattavissa, eli jos haluaa alustaa eri pool-kokoja, pitää lisätä useita poolpreparer-skriptikomponentteja. Automaattisesti koon lisääminen on hyvin yksinkertaista ja nostaa sitä vain 10 %, jos ei itse alusta suoraan haluttuun kokoon, se voi aiheuttaa ruudunpäivityksen hidastumista. (Unity3d-object-pooling 2017.) Tätä systeemiä käytetään autopeliobjektin prefabin alustamiseen, ja kun auto törmää seinään, sen lisäämiseen takaisin jonoon. Näin vältetään ruudunpäivityksen hidastumista, jos peli loisi objektit joka kerta erikseen. Tämä lisää hieman muistin käyttöä.

5.5.7 Pelaajan tavoite

Kissan kiinni ottaminen on pelin tarkoitus, ja kun pelaaja saa sen kiinni, peli päättyy siihen. Kissa on 3D-hahmo, jossa on komponentteina Rigidbody, Box collider, koska colliderin ei tarvitse olla tarkka, sekä Audio Source ja skriptit Cat, Cat Rotation ja Resonance Audio Source (KUVA 38).



KUVA 38. Kissa inspector-ikkunassa

Cat-skriptissä on määritelty taulukko eli array kohteista, joita kohti kissa menee, kissan nopeus sekä Audio Clip ja Audio Source. Void Start() -lausekkeessa on määritelty CallAudio()-luokka, void Awake() -lausekkeessa aina pelin käynnistyessä skripti kutsuu Audio Source-komponentin (KUVA 39). Update() -lauseke on ohjelmoitu etsimään seuraava kohdeobjekti, jota kohti kissa menee (KUVA 39). CallAudio()-lauseke soittaa satunnaisia kissan ääniä tietyn ajan välein. Äänet on lisätty Inspector-ikkunassa (KUVA 38).

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
 [RequireComponent(typeof(AudioSource))]
public class Cat : MonoBehaviour {

    public Transform[] target;
    public float speed;
    private int current;

    public AudioClip[] audioClips;

    public AudioSource audioSource;
    void Start()
    {

        CallAudio();
    }
    private void Awake()
    {
        audioSource = GetComponent<AudioSource>();

    }

    // Update is called once per frame
    void Update () {
        if (transform.position != target[current].position)
        {
            Vector3 pos = Vector3.MoveTowards(transform.position, target[current].position, speed * Time.fixedDeltaTim
            GetComponent<Rigidbody>().MovePosition(pos);
        }
        else current = (current + 2) % target.Length;
    }
    void CallAudio()
    {
        Invoke("RandomSound", 2);
    }
    void RandomSound()
    {
        audioSource.clip = audioClips[Random.Range(0, audioClips.Length)];
        audioSource.Play();
        CallAudio();
```

KUVA 39. Cat-skripti

Pelin alussa kissa lähtee heti ensimmäistä kohdetta kohti, minkä jälkeen se jää siihen odottamaan, kunnes pelaaja koskettaa matkalla olevaa triggeriä (Trigger), minkä jälkeen se lähtee seuraavaa kohdetta kohti ja tekee tätä näin, kunnes pelaaja koskettaa viimeistä triggeriä, minkä jälkeen se pysähtyy viimeiseen kohteeseen (KUVA 39). Player Collision -skriptissä on määritelty kaikki pelaajan trigger-collisionit. Cat Rotation-skripti kääntää kissan katsomaan sitä kohdetta, johon se on menossa seuraavaksi (KUVA 40).

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class CatRotation : MonoBehaviour {
    public Transform target;


    void FixedUpdate () {

        Vector3 direction = target.position - transform.position;
        Quaternion rotation = Quaternion.LookRotation(direction);
        transform.rotation = rotation;

    }
}
```

KUVA 40. CatRotation-skripti

5.6 Google Resonancen käyttäminen

Jos haluaa, että käyttäjä huomaa äänen näkymän ulkopuolella, animoidaan äänen sijainti. Tämä auttaa käyttäjää löytämään äänen nopeammin. Myös äänien soittaminen useamman kerran auttaa. Toistuvat äänet auttavat käyttäjiä tunnistamaan ja etsimään ne. Esimerkiksi puhelimet soittavat useita auttaakseen meitä huomaamaan saapuvan puhelun ja etsimään puhelimen. Saman vaikutuksen voi saavuttaa käyttämällä ääniä, jotka sisältävät monia eri elementtejä. (Design tips 2018.)

On hyvä käyttää monimutkaisia ääniä, riittävää äänenvoimakkuutta, koko taajuuksien spektriä sekä välttää ääniä, jotka ovat liian hiljaisia, joista puuttuu korkeita taajuuksia tai jotka ovat liian yksinkertaisia, esimerkiksi siniaallon äänimerkki. Olisi hyvä, että visuaalinen ja audiotiivinen kokemus olisi synkronoitu, sekä varmistaa, että kuvat, jotka käyttäjät näkevät skenessä, vastaavat kuultua ääntä. Jos pelaaja esimerkiksi kuulee, että valtameren aallot törmäävät rannikkoon, mutta meressä kohtaus näyttää liikkumattomalta, kohtaus tuntuu vähemmän realistiselta (Design tips 2018.)

Koko kehitystyön ajan on varmistettava, että äänilähteet ja kokoonpanot tuottavat optimaalisia äänikokemuksia. Ääniä tulisi testata koko ympäristössä. Kaikissa alueissa, joita käyttäjät voivat tutkia, tulisi varmistaa, että ympäristö kuulostaa luonnolliselta. Tyypillisesti, kun käyttäjät voivat liikkua vapaasti ympäristössä, he siirtyvät jopa äänilähteisiin. Olisi hyvä varmistaa äänenlaatu, äänenvoimakkuus ja reagointikyky sekä se, että äänet ovat korkealaatuisia, ovat selkeässä mutta mukavassa äänenvoimakkuudessa ja sopeutuvat liikkumiseen realistisesti. Kuulokkeita tulisi käyttää testauksessa. Kuulokkeet ovat

välttämättömät, jotta tilaääni voidaan kokonaan kokeilla. Tietokoneen kaiuttimia tai muita kaiuttimia ei tulisi käyttää testauksessa. (Design tips 2018.)

5.6.1 Google Resonancen asentaminen Unitylle

Jotta Google Resonancea voi käyttää Unity-projektissa, se pitää asentaa ensin. Unityn käynnistyksen jälkeen luodaan uusi 3D-projekti. Kuten aiemmin luvussa 5.1 luvussa kerrottiin, sieltä valitaan Asset, sen alapuolelta Import Package, minkä jälkeen valitaan Custom Package ja ResonanceAudio-ForUnity_*. unitypackage-tiedosto, joka on ladattu. Import Package-ruudussa valitaan Import ja hyväksytään kaikki API-päivitykset, jos tätä kysytään. Jo luodussa Unity-projektissa voidaan käyttää Google Resonancea, jolloin valitaan Edit, Project settings ja audio, jotta AudioManager Settings aukeaa minkä jälkeen valitaan Resonance Audio Spatializer Pluginina sekä Ambisonic Decoder Plugin. Resonancen mukana tulee Audio Demo, jota voi kokeilla. Resonance Audiossa on useita komponentteja, joita voidaan käyttää.

ResonanceAudioListener parantaa Unityn AudioListener -ominaisuuksia ottamalla käyttöön muita parametreja, kuten globaaleja vahvistus- ja lähde-occlusion-maskeja. Se sisältää Ambisonic-äänikentän tallennuskomponentteja, jotka sallivat spatiaalisten audiolähteiden bakingin Ambisonic-äänikenttään. Se vaatii Unity AudioListenerin samassa peliobjektissa.

ResonanceAudioSource parantaa Unity Audiosource -ominaisuuksia ottamalla käyttöön muita valinnaisia parametreja, kuten suoruusmalleja, occlusionin ja renderointi -laadun. Se vaatii Unity AudioSourcen samassa peliobjektissa.

ResonanceAudioRoom simuloi huoneen vaikutuksia tiettyyn tilaan tuomalla dynaamisia alkuheijastuksia ja myöhäistä jälkikaiuntaa. Se käyttää peliobjektin Transform-ominaisuuksia ja soveltaa huoneen vaikutuksia sen mukaisesti. Vastaavat huoneen äänitehosteet ovat käytössä aina, kun AudioListener on huonemallin määritettyjen rajojen sisällä.

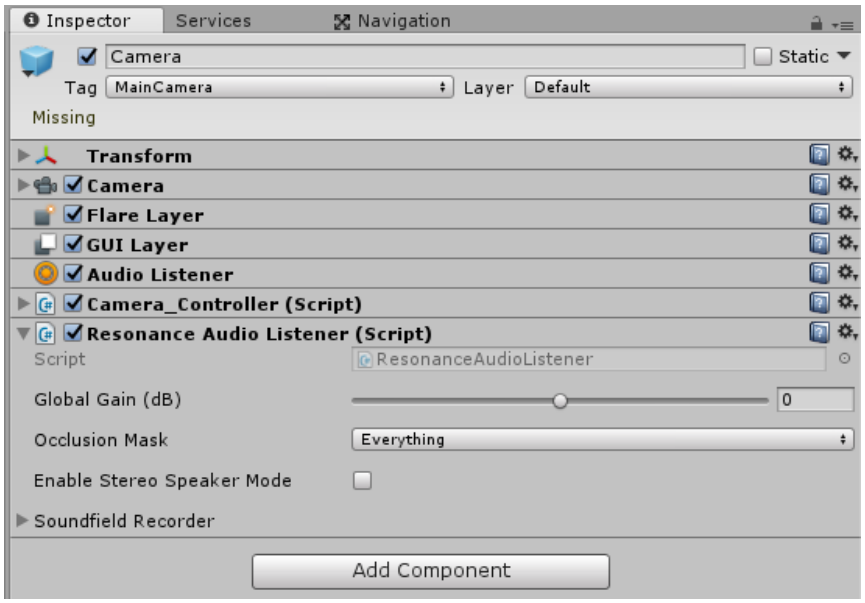
ResonanceAudioReverbProbe tarjoaa edistyksellisen vaihtoehdon tilojen hienompaan mallintamiseen ja vivahteikkaimpiin kaikuvaikutuksiin. ResonanceAudioSoundField edustaa koko 360 asteen äänentoistoa koodaamalla ääniaaltoja kuulijan ympärille virtuaalisena pallona. (Getting Started 2017.)

5.6.2 Google Resonancen toteutus

Objektit, jotka soittavat ääniä, tarvitsevat Resonance Audio Source-skriptikomponentin sekä Audio Source -komponentin. Audio Sourcen outputiin lisätään masteriksi ResonanceAudioMixer, Spatialize sallitaan sekä Loop sallitaan, jos soitetaan ääniä useasti. Gain lisää lähteelle vahvistusta suhteellisen äänenvoimakkuuden säätämiseksi. Bypass Room Effects -asetuksella voi säätää objektin jättämään huoneiden vaikutuksen huomiotta. Listener Directivity hallitsee lähteen kuuntelijan ääniherkkyyden mallia. Tämä voi muuttaa lähteen havaitun äänenvoimakkuuden riippuen siitä, mihin suuntaan kuuntelija katsoo suhteessa lähteeseen. Mallit on kohdistettu kuuntelijasta eteenpäin. Alpha ohjaa tasapainoa dipoli-kuvion ja ohimenevän mallin kuuntelijan herkkyyden välillä. Muutettaessa tätä arvoa voi tehdä erilaisia suuntausmalleja. Sharpness asettaa kuuntelijan suuntaavuuden mallien terävyyden. Korkeampi arvo johtaa kohdennetun suuntauksen lisääntymiseen. Source Directivity hallitsee lähteen äänenvoimakkuutta. Tämä voi muuttaa lähteen havaittua äänenvoimakkuutta riippuen siitä, mihin suuntaan se on suhteessa kuuntelijalle. kuviot kohdistetaan pääobjektista eteenpäin. Enable Occlusion määrittää, onko lähteen ääni suljettava, kun lähteen ja kuulijan välillä on muita kohteita. Quality asettaa laatutilan, jolla spatiaalinen audio renderoidaan. Korkeimmat laatutilat mahdollistavat entistä paremman tarkkuuden suuremman CPU:n käytön kustannuksella. (Developer guide 2017.)

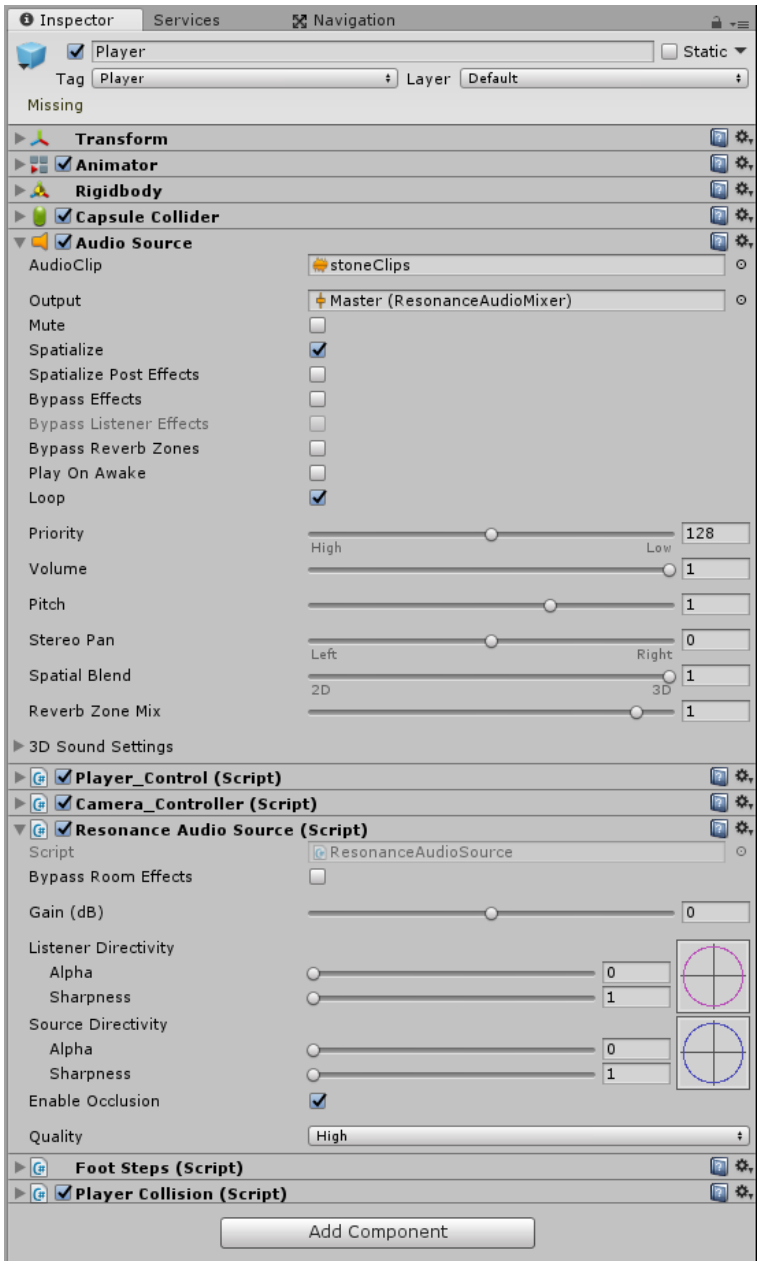
5.6.3 Google Resonance pelaajassa

Pelaajan kameraan on lisätty Resonance Audio Listener -skripti, joka toimii mikrofonin kaltaisena laitteena (KUVA 41).



KUVA 41. Resonance Audio Listener -skripti kamerassa

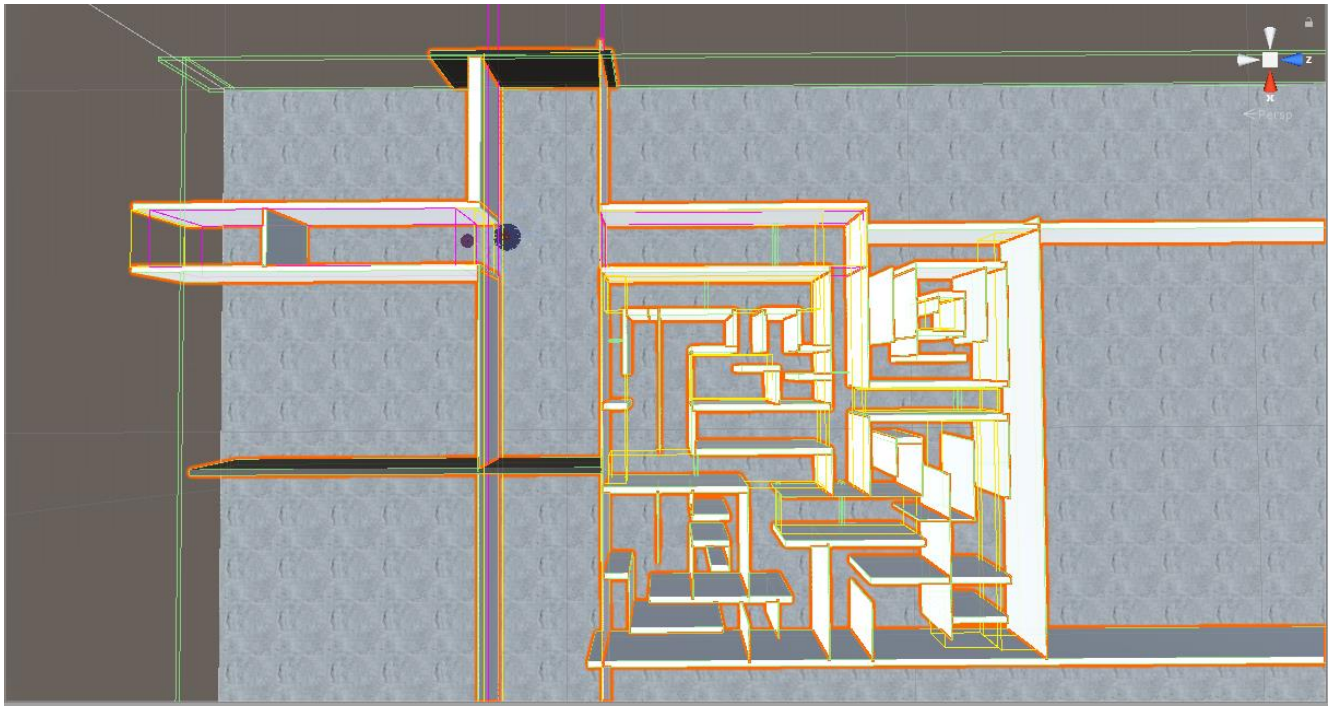
Kamera saa syötteen mistä tahansa äänilähteestä skenessä ja toistaa äänet. Jos pelaaja on Reverb-alueen rajoissa, se kohdistuu kaikkiin kuultaviin ääniin skenessä. Lisäksi äänitehosteet voidaan soveltaa kuuntelijalle ja reverbiä sovelletaan kaikkiin ääniin skenessä. Audio Listener toimii yhdessä Audio Sourcen kanssa (KUVA 42). Jos lähteet ovat 3D-muotoisia, kuuntelija jäljittelee äänen asemaa, nopeutta ja suuntausta 3D-maailmassa.



KUVA 42. Pelaajan Audio Source ja Resonance Audio Source -skripti inspector-ikkunassa

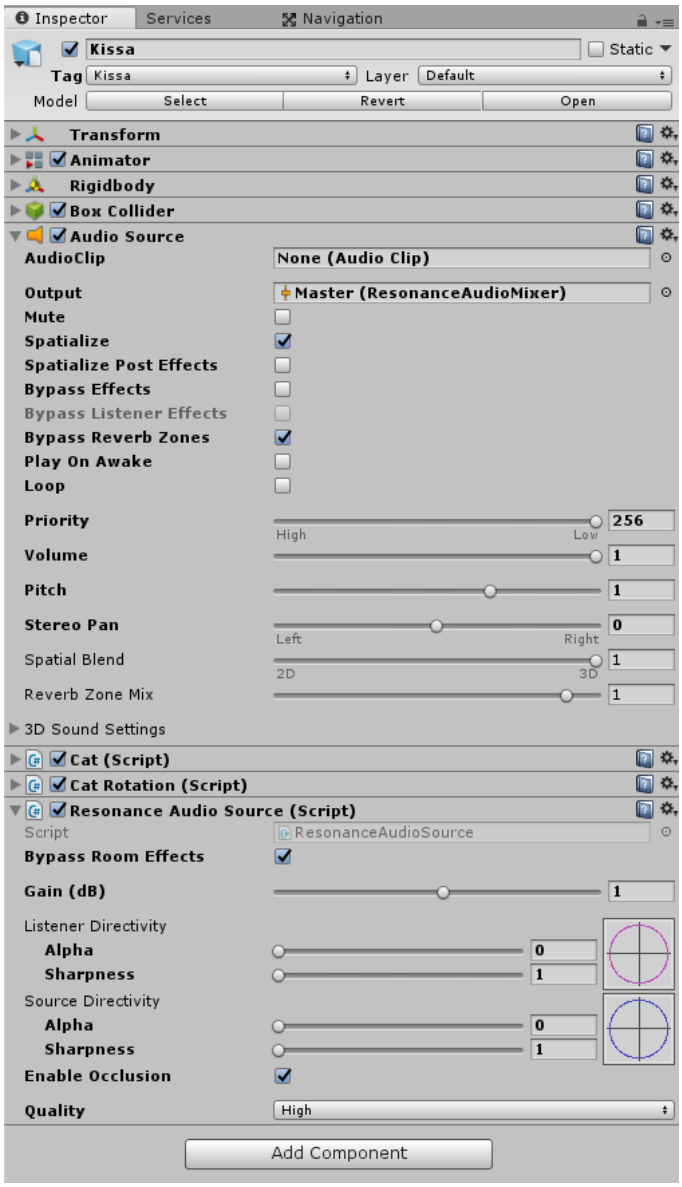
5.6.4 Google Resonance muissa objekteissa

Peliympäristö objekteihin on lisätty Resonance Audio Room. Audio room -ominaisuuksia, kuten pinta-aineita ja reverb-modifioijia, voi hienosäätää tarvittaessa. Tässä projektissa alleyway- ja alleway_1- eli kujahuoneissa käytettiin Wall eli seinä Brick Painted, Floor eli lattia Concrete Block Painted ja Ceiling eli katto ja Front Wall eli etummainen seinä ovat transparent. Street eli katuseinissä käytettiin Brick Painted, lattiassa Concrete Block Painted ja etummainen seinä sekä katto ovat transparent. Labyrintissa on käytetty monimutkaisuuden takia useita Resonance Audio Roomeja (KUVA 43).



KUVA 43. Esimerkki Audio room- ja reverb probe -komponentistä

Autossa, kissassa sekä liikennevalossa on käytetty Resonance Audio Sourcea ja Audio source komponentteja. Audio Sourcen output on master (ResonanceAudioMixer). Spatialize, Occlusion ja Loop on inspectorissa sallittu. Kissassa on Bypass Reverb Zones, ja Bypass Room Effects on sallittu, jotta kissan ääniä kuuluisi selkeästi (KUVA 44).



KUVA 44. Kissan Audio Source-komponentti ja Resonance Audio-skripti

5.7 Canvas

Canvas on alue, jota kaikki käyttöliittymäelementtien pitäisi käyttää. UI-elementit ovat kuvat, tekstit, napit, liukusäätimet ja syöttökentät. Canvas on peliobjekti, jossa on Canvas-komponentti, ja sen ja kaikkien UI-elementtien on oltava tällaisen Canvasin lapsi eli children. Uuden UI-elementin luominen, kuten kuvan luominen, luodaan valitsemalla peliobjekti, josta valitaan UI, minkä jälkeen valitaan Image-valikko, joka luo automaattisesti Canvaksen, jos skenessä ei ole vielä sitä. UI-elementti on luotu chil-

dreninä canvakseen. Canvas-alue näkyy näkymässä suorakulmiona, mikä helpottaa UI-elementtien sijoittamista ilman, että Game View näkyy kaikkina aikoina. Canvas käyttää EvenSystem-objektia auttaakseen Messaging-järjestelmää. (Canvas 2018.)

UI-elementit piirretään samassa järjestyksessä ruudulle, kuin ne näkyvät hierarkiassa. Ensimmäinen children piirretään ensin, toinen seuraavaksi ja niin edelleen. Jos kaksi UI-elementtiä ovat päällekkäin, myöhempi näkyy päällekkäin. Jos haluaa muuttaa elementtien järjestystä, sen voi tehdä vetämällä hierarkian elementit uudelleen. (Canvas 2018.)

5.7.1 Renderointi-tilat

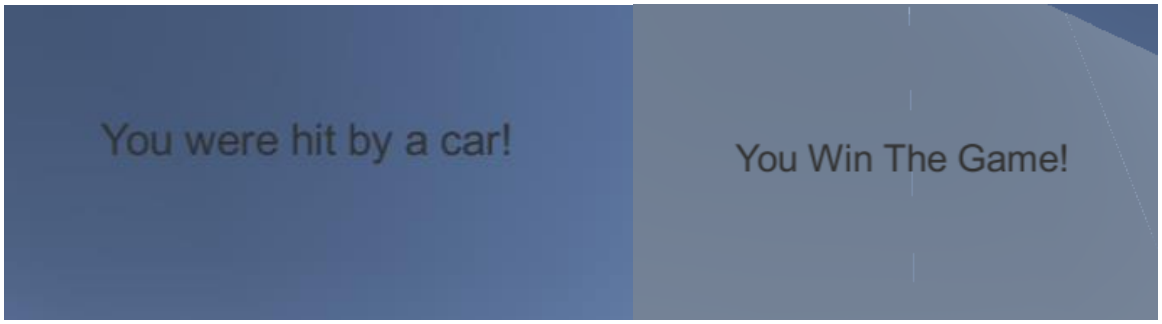
Canvaksella on myös Render mode -asetus, jota voidaan käyttää näyttämään näytön tai pelimaailman tilassa. Näytön-overlay-tilassa UI-elementit näkyvät kuvaruudun yläpuolelle näytettävään näyttöön, ja jos näyttöä muutetaan, Canvas muuttaa automaattisesti kokoa vastamaan tätä. Näytön kamera-tilassa Canvas on tietyn matkan päässä kamerasta. Kamera renderöi UI-elementit, joten kameran asetukset vaikuttavat siihen, miten UI näkyy. Asetettaessa kamera tiettyyn perspektiiviin UI-elementit näytetään perspektiivinä ja perspektiivisen vääristymän määrää voidaan ohjata kameran näkökentän avulla. Näytön kokoa muutettaessa se muuttaa resoluution tai kameran Canvaksen automaattisesti kokoa vastaavaksi. World Space -renderointitilassa Canvas toimii kuin mikä tahansa muu kohde näytössä. Canvas-koko voidaan asettaa manuaalisesti Rect-muunnoksen avulla ja UI-elementit tulevat 3D-sijoittelun perusteella esille muiden kohteiden edessä tai takana. Tämä on hyödyllinen UI-elementeille, joiden on tarkoitus olla osa maailmaa. Tätä kutsutaan myös nimellä ”Diegetic Interface”. (Canvas 2018.)

5.7.2 Canvas projektissa

Tässä projektissa Canvaksen UI-elementtejä käytetään kertomaan pelaajille, mitä pitää tehdä äänitehosteiden lisäksi, sekä kertomaan, kun pelaaja voittaa tai jää auton alle. Voitto- sekä auton alle jääminen -tekstit kutsutaan PlayerCollision-skriptissä, ja kummatkin näyttävät tekstin, joka tiedottavat pelaajalle. Pelin aloituksessa näkyy teksti, joka kertoo pelaajalle hänen tavoitteensa pelissä, äänitehosteen lisäksi (KUVA 45).

KUVA 45. Catch The Cat! -teksti

Tämän kutsuu GameManager skripti, joka myös käynnistää pelin uudestaan ja näyttää tekstin, kun pelaaja jää auton alle tai voittaa pelin (KUVAT 46 ja 47).



KUVA 46. You were hit by a car! -teksti

KUVA 47. You Win The Game! -teksti

6 JOHTOPÄÄTÖKSET

Työn tavoitteena oli suunnitella 3D-äänipeli myTrueSound Oy:lle. Toimeksiantona tehdyn demon tarkoituksena on antaa yritykselle ja lukijalle suuntaa siitä, kuinka luoda 3D-äänipeli Unity 3D -pelimoottorilla käyttäen Google Resonancea audio väliohjelmistona.

Ennen tätä työtä minulla oli perustiedot Unity 3D -pelimoottorista ja C#-ohjelmointikielestä. Google Resonancesta en ollut kuullut ennen kuin sain toimeksiannon. Tämäntyyppistä osaamista ei minulle ollut kertynyt kursseilta. Ennen tätä projektia olin luonnut vuoden 2015 kesällä pelipajalla pelin. Tämä työ tarjosi uutta tietoa siitä, kuinka kehittää peli käyttäen Unity 3D -pelimoottoria sekä kuinka Google Resonancea käytetään. Opin myös, kuinka yksinkertaisen demon luominenkin vaatii aikaa ja vaivaa.

Projektin aikana opin todella hyvin Unity -pelimoottorin, sen ominaisuudet ja kuinka ohjelmoida käyttäen Microsoftin Visual Studiota. Auton käyttäytymisen ohjelmointi oli mielenkiintoisin, sillä sen aikana opin ja sain ideoita, kuinka voin tulevaisuudessa ohjelmoida pelin vihollisten käyttäytymisen paremmin ja helpommin. Vaikein oli ohjelmoida Pool-systeemi, sen kanssa useampi viikko meni sen kanssa, jotta sain sen toimimaan oikein.

Lopputulos ei välttämättä ollut aivan se, mitä aluksi oli suunniteltu. Peliympäristö onnistui ja pelaaja pystyy kulkemaan siinä äänien avulla. Audiologinen ja taktillinen käyttöliittymä jäi pois ajan ja vaikeuden takia. Peliä voi vielä testata ja muokata palautteen avulla jatkokehitystä silmällä pitäen. Äänet voisi olla parempia ja monipuolisempia, mutta rajoitteena oli käyttää ilmaisia vapaasti käytettäviä ääniä, joten siihen nähden ne onnistuivat. Käytän tätä projektia ja oppimiani taitoja pohjana muihin Unity 3D -peliprojekteihin.

LÄHTEET

Brown, E. & Cairns, P. 2004 A Grounded Investigation of Game Immersion. New York: ACM, 1297-1300.

Canvas. 2018. Www-dokumentti. Saatavissa: https://docs.unity3d.com/Manual/UICanvas.html/. Viitattu 10.9.2018.

Components. 2018. Www-dokumentti. Saatavissa: https://docs.unity3d.com/Manual/UsingComponents.html. Viitattu 6.3.2018

Creating and Using Scripts. 2018. Www-dokumentti. Saatavissa: https://docs.unity3d.com/Manual/CreatingAndUsingScripts.html.Viitattu 7.9.2018.

Design tips. 2017. Www-dokumentti. Saatavissa: https://computer.howstuffworks.com/3dgraphics5.htm. Viitattu 9.9.2018.

Developer guide. 2017. Www-dokumentti. Saatavissa: https://developers.google.com/resonance-audio/develop/unreal/developer-guide/. Viitattu 9.9.2018.

Franklin. C. 2008. How Stuff Works 3D Graphics a. Www-dokumentti. Saatavissa: https://computer.howstuffworks.com/3dgraphics4.htm. Viitattu 24.8.2018.

Franklin. C. 2008. How Stuff Works 3D Graphics b. Www-dokumentti. Saatavissa: https://computer.howstuffworks.com/3dgraphics4.htm. Viitattu 24.8.2018.

Fullerton, T., Swain, C. & Hoffman S. 2004. Game Design Workshop. San Francisco: CMP Books.

GameObjects. 2018. Www-dokumentti. Saatavissa: https://docs.unity3d.com/560/Documentation/Manual/class-GameObject.html. Viitattu 10.3.2018.

Getting Started. 2017. Www-dokumentti. Saatavissa: https://developers.google.com/resonance-audio/develop/unreal/getting-started/. Viitattu 9.9.2018.

Gigi. 2011. Why Games Work and Science of Learning. Www-dokumentti. Saatavissa: http://www.goodgamesbydesign.com/2011/07/why-games-work-the-science-of-learning/. Viitattu 17.9.2018.

Isaza, M. 2010. Aaron Marks Special: Function of Game Sound Effects. Www-dokumentti. Saatavissa: http://designingsound.org/2010/10/15/aaron-marks-special-function-of-game-sound-effects/. Viitattu 31.8.2018.

Langley, B. 2015. Windows 10 and DirectX 12 released! Www-dokumentti. Saatavissa: https://blogs.msdn.microsoft.com/directx/2015/07/29/windows-10-and-directx-12-released/. Viitattu 24.8.2018.

Overview. 2018. Www-dokumentti. Saatavissa: https://developers.google.com/resonance-audio/develop/overview. Viitattu 9.9.2018.

Platform. 2018. Www-dokumentti. Saatavissa: https://unity3d.com/unity/features/multiplatformViitattu. 6.3.2018.

Public Relations. 2018. Www-dokumentti. Saatavissa: https://unity3d.com/public-relations. Viitattu 6.3.2018.

Puhakka, A. 2008. 3D-grafiikka. Helsinki: Talentum.

Quinn, A 2008. Types and roles of sound in games. Www-dokumentti. Saatavissa: http://www.aquinn.co.uk/wordpress/7. Viitattu 26.8.2018.

Rollings, A. & Morris, D. 2000. Game Architecture and Design. Arizona: The Coriolis Group, LLC.

Rutledge. 2014. Positive side of Video games part 3. Www-dokumentti. Saatavissa: https://en.paperblog.com/the-positive-side-of-video-games-part-iii-294723/. Viitattu 17.9.2018.

Science Buddies. 2011. Creating Video Games for the Blind. Www-dokumentti. Saatavissa: https://www.sciencebuddies.org/science-fair-projects/project-ideas/Games_p029/video-computer-games/creating-a-video-game-for-the-blind#background. Viitattu 17.9.2018.

Sharma, K. 2013. What “DirectX” really is, How it works. Www-dokumentti. Saatavissa: https://www.softnuke.com/explained-directx-how-it-works/. Viitattu 6.3.2018.

Singer, G. 2013. The History of the Modern Graphics Processor. Www-dokumentti. Saatavissa: https://www.techspot.com/article/650-history-of-the-gpu/ Viitattu. 24.8.2018.

Slick J. 2014a. What is 3d. 3D Defined. Www-dokumentti. Saatavissa: https://www.lifewire.com/what-is-3d-1951. Viitattu 6.3.2018.

Slick J. 2014b. 3d-model components. Www-dokumentti. Saatavissa:
Software. 2018. Www-dokumentti. Saatavissa: https://www.lifewire.com/3d-model-components-1952. Viitattu 6.3.2018.

Taylor, A. 2013. How to make a video game for the blind. Www-dokumentti. Saatavissa: https://www.popularmechanics.com/culture/gaming/a13065/how-to-mak-a-video-game-for-the-blind-15277536/. Viitattu 17.9.2018.

Unity Rigidbody script reference. 2018. Www-dokumentti. Saatavissa: https://docs.unity3d.com/ScriptReference/Rigidbody.html. Viitattu 7.9.2018.

Unity scripting languages. 2018. Www-dokumentti. Saatavissa: https://blogs.unity3d.com/2014/09/03/documentation-unity-scripting-languages-and-you/. Viitattu 9.3.2018.

Unity3d Object Pooling. 2017. Www-dokumentti. Saatavissa: https://unity3d.college/2017/05/11/unity3d-object-pooling/. Viitattu 9.9.2018.

Ward, J. 2008. What is a game engine? Www-dokumentti. Saatavissa: http://www.gamecareer-guide.com/features/529/what_is_a_game_.php. Viitattu 6.3.2018.

Variables. 2018. Www-dokumentti. Saatavissa: https://unity3d.com/learn/tutorials/topics/scripting/variables-and-functions. Viitattu 10.3.2018.