Karelia-ammattikorkeakoulu
Tradenomi, Tietojenkäsittely (AMK)

Pelaajalle viestiminen partikkeliefekteillä virtuaalitodellisuuspeleissä

Lauri Hiltunen

Opinnäytetyö, toukokuu 2022

www.karelia.fi

Karelia AMMATTIKORKEAKOULU	**OPINNÄYTETYÖ** **Toukokuu 2022** **Tietojenkäsittelyn koulutus** Tikkarinne 9 80200 JOENSUU +358 13 260 600 (vaihde)

Tekijä(t)
Lauri Hiltunen

Nimeke
Pelaajalle viestiminen partikkeliefekteillä virtuaalitodellisuuspeleissä

Tiivistelmä

Tässä opinnäytetyössä tarkastellaan, miten partikkeliefektejä voidaan käyttää osana virtuaalitodellisuuspelin käyttöliittymää ja miten niillä voidaan viestiä pelaajalle olennaista tietoa, kuten pelihahmon elinvoiman määrä. Työssä keskitytään pelimaailman sisäisiin käyttöliittymän osiin ja niissä käytettäviin partikkeliefekteihin. Työssä toteutettiin Kurja Demo -pelin käyttöliittymään partikkeliefektien avulla elinvoiman ja taikavoiman viestimisen pelaajalle. Lisäksi peliin toteutettiin partikkeliarvoitus, joka pelaajan täytyy ratkaista, jotta voi edetä pelissä.

Opinnäytetyössä käydään läpi erilaisia käyttöliittymien esitystapoja ja tarkastellaan partikkeliefektien toimintaa yleisellä tasolla. Käytännön osuuden toteutus tapahtui Dreal-yrityksessä. Toteutukseen käytettiin GIMP-kuvanmanipulointiohjelmaa, Blender-mallinnusohjelmaa, Visual studio -koodieditoria ja Unity-pelimoottoria. Lisäksi työssä vertailtiin toteutettuja käyttöliittymän osia toisen virtuaalitodellisuuspelin käyttöliittymän vastaaviin osiin, jotka ovat toteutettu eri menetelmillä.

Käyttöliittymänelementtien lisäksi työn tuloksena syntyi jatkokehitys ideoita. Vertailu toiseen peliin havainnollisti, että pelaajalle tietoa viestiviä käyttöliittymän osia on mahdollista toteuttaa eri tavoin. Partikkeliefektit osana käyttöliittymää ovat toimiva tapa viestiä pelaajalle virtuaalitodellisuuspeleissä.

Kieli suomi	Sivuja 37 Liitteet 0 Liitesivumäärä 0

Asiasanat
virtuaalitodellisuus, käyttöliittymä, partikkeliefekti

Karelia
UNIVERSITY OF APPLIED SCIENCES

THESIS
May 2022
Degree Programme in information technology

Tikkarinne 9
80200 JOENSUU
FINLAND
+ 358 13 260 600 (switchboard)

Author (s)
Lauri Hiltunen

Title
Informing player with particle effects in virtual reality game

Abstract

In this thesis goal is to study how to inform player with particle effects as a part of a user interface in virtual reality game and how to use them to relay useful information to the player, for example current amount of health that player character has. Thesis focuses on parts of the user interface and particle effects on them that are inside a game world. User interface elements of Kurja Demo game which informs player of their health and spell power were made in this thesis. Furthermore, a particle puzzle was made that player must solve so that they can advance in the game.

In the thesis I go through different ways to present the user interface and studied how particle effects work on general level. Practical part of the thesis was done in Dreal company. Programs that were used are GIMP image manipulation program, Blender modelling program, Visual studio code editor and Unity game engine. Additionally, comparisons were made to a different virtual reality game's user interface, that were made with different methods.

In addition to the game's user interface elements also ideas of future improvement were made. Comparisons to another game demonstrated that it is possible to use different methods to relay similar information as a part of user interface. Particle effects are an functional way to inform player in virtual reality games.

Language
Finnish

Pages 37
Appendices 0
Pages of Appendices 0

Keywords
virtual reality, user interface, particle effect

Sisältö

1 Johdanto

Virtuaalitodellisuus (VR) käsitteenä on jo ollut olemassa kymmeniä vuosia, mutta sen käyttö peleissä on ruvennut vasta lähiaikoina yleistymään, laitteiden helppokäyttöisyyden parantuessa ja niiden hintojen tullessa halvemmaksi. Jo 1990-luvulla oli yrityksiä tehdä VR-pelejä kotikäyttäjille, mutta yleistymään ne alkoivat vasta 2010-luvulta eteenpäin ja nykyisin VR-laitteistoja on saatavilla useita erilaisia. Suurin osa VR-laitteista vaatii johdollisen yhteyden tietokoneeseen, jotta niillä voi pelata VR-pelejä, ja jotkut vaativat lisälaitteiden sijoittelua pelitilaan, pelaajan sijainnin seuraamiseksi. Alan kehitys on nyt lähtenyt nousuun ja nyt myös saatavilla on ilman tietokoneliitäntää ja lisäosia vaatia VR-laseja, mutta näissä tapauksissa suorituskyky on rajattuna VR-lasien omaan laskentatehoon. Johdottomien VR-laitteiden suorituskyky on verrattavissa tehokkaaseen älypuhelimeen, joten parhaimman näköisten pelien pelaamiseen tarvitsee tietokoneyhteyden. Kuitenkin nykyisin on saatavilla ohjelmia, joilla voi yhdistää lasit tietokoneeseen langattomasti. Tällainen järjestely kuitenkin vaatii tehokkaan langattoman verkkoyhteyden, jolloin tietokoneelle voidaan jättää pelin tehovaatimukset ja laseilla voidaan nauttia peleistä ilman datakaapelia.

Suurin ero VR-peleissä tavallisiin näytöiltä katseltaviin peleihin on siinä, että VR-lasit päässä ainoa asia, jonka pelaaja voi nähdä on VR-maailma, johon hänet laitetaan. VR-peleissä pelimaailma ympäröi pelaajan kokonaan josta johtuen tunne siihen osana olemisesta on vahva. Pelaaja on suoraan pelihahmon paikalla pelimaailmassa VR-pelissä. Pelaajan hahmo kääntää päätään samoin tavoin kuin pelaaja, ja hahmon kädet liikkuvat samalla tavalla kuin pelaajan kädet liikkuvat. Tämä myös tarkoittaa sitä, että käyttöliittymät pitää suunnitella eri tavoin VR-peleissä.

Partikkeliefektit ovat joukko pisteitä kolmiulotteisessa tilassa. Jokaisella yksittäisellä partikkelilla on elinkaari, jonka aikana ne voivat muuttua, riippuen annetuista parametreistä. Peleissä partikkeliefekteillä yleensä toteutetaan visuaalisia tehosteita. Pelissä tapahtuva pommin räjähdys voi esimerkiksi koostua liekeistä, savusta, sirpaleista, kipinöistä ja shokkiaallosta. Jokainen edellä mainituista asioista voidaan toteuttaa partikkeliefektein.

Tässä opinnäytetyössä tarkastellaan, miten partikkeliefektejä voidaan käyttää osana VR-pelin käyttöliittymää ja miten niillä voidaan viestiä pelaajalle olennaista tietoa, kuten pelihahmon senhetkinen elinvoiman määrä. Työssä keskitytään pelimaailman sisäisiin käyttöliittymän osiin ja niissä käytettäviin partikkeliefekteihin. Työssä käytetään mahdollisimman yksinkertaisia partikkeliefektejä, jotta ne vievät mahdollisimman vähän laskentatehoa ilman, että niiden ulkoasu kärsii liikaa.

Opinnäytetyön tekemisen aikana olin osana tekemässä Kurja Demo -nimistä peliä Dreal-yrityksen mukana. Tarkoituksena oli oppia ja kartuttaa kokemusta Unity-pelimoottorilla, Blender-mallinnusohjelmalla ja GIMP-kuvanmanipulointiohjelmalla työskentelystä sekä yleisestä pelienkehittämisestä virtuaalitodellisuusalustalla. Tarkoituksena oli myös tarkoituksena oppia lisää Unity-pelimoottorissa käytettävästä C# -ohjelmointikielestä.

Raportissa käyn läpi työvälineet, joita käytin. Esittelen teoriaa käyttöliittymistä ja partikkeliefekteistä ja kerron miten itse toteutin peliin käyttöliittymän osia. Lisäksi esittelen toteuttamani partikkeliarvoituksen, joka pelaajan tulee ratkaista edetäkseen pelissä. Vertailen tekemiäni käyttöliittymän osia toiseen peliin. Lopuksi pohdin opinnäytetyön tuloksia ja mietin jatkokehityskohteita.

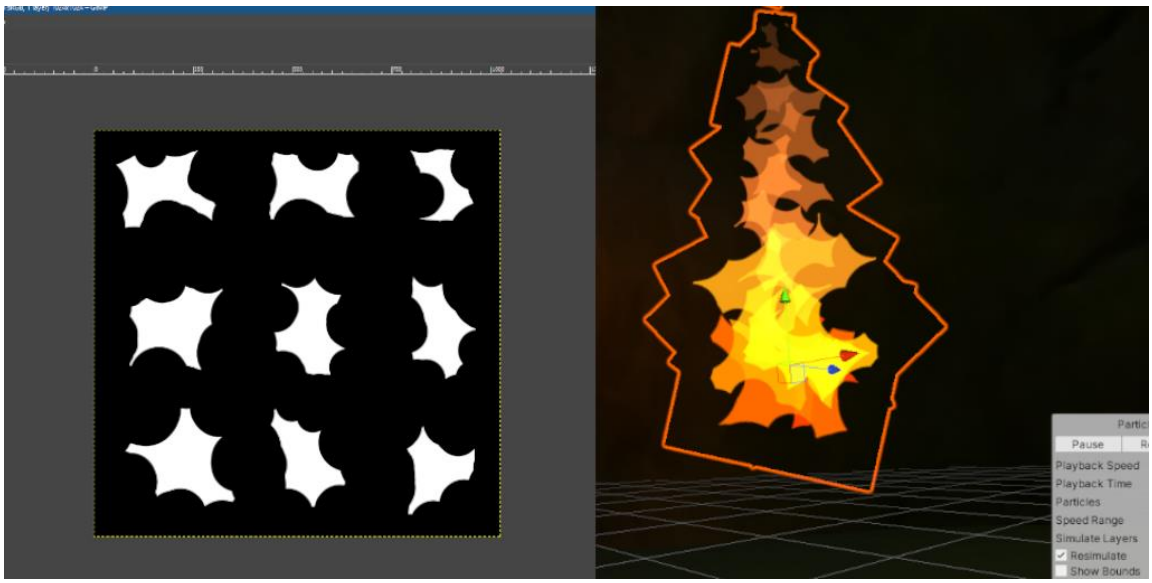
2 Työvälineet

2.1 GIMP-kuvanmanipulointiohjelma

GIMP on akronyymi sanoista GNU Image Manipulation Program sanoista. GNU taas tarkoittaa yleistä julkista lisenssiä, joka on vapaa, joten ohjelman nimi jo kertoo, että se on ilmainen. Ohjelmaa voi käyttää myös ammattikäytössä ilman lisämaksuja. (GNU 2007.) GIMP-ohjelman taustalla ja tekijänä ei ole yhtiötä, vaan se on tehty vapaaehtoisten ryhmävoimin. Suurin osa tekijöistä on kotoisin Euroopan alueelta, ja kieliä, joilla GIMP on saatavilla, on tällä hetkellä 80. (GIMP 2022.)

GIMP mahdollistaa kuvien muokkauksen sekä luomisen ja ohjelma on monipuolinen, mutta tässä opinnäytetyössä oli tarkoituksena tehdä mahdollisimman vähän suorituskykyä käyttäviä partikkeleita, joten tekemäni

tekstuurit olivat suhteellisen pieniä ja yksinkertaisia. Tekstuurit ovat mitä tahansa kaksiulotteisia kuvia, joita voidaan hyödyntää Unity-pelimoottorissa objektien materiaaleina. Partikkelijärjestelmiin tekstuurien ei tarvitse tarvitse olla vain yksi kuva, vaan niitä voidaan toteuttaa myös tekstuuriatlaksena (kuva 1). Tekstuuriatlaksissa yhdessä tekstuurissa on monta kuvaa, jotka partikkelijärjestelmä voi pilkkoa omiksi kuviksi sekä selata niitä käyttäjän määrittelemillä asetuksilla. Tällä tavoin kuvia selaamalla sarjakuvamaisesti on mahdollista saada aikaan animoituja partikkeliefektejä.



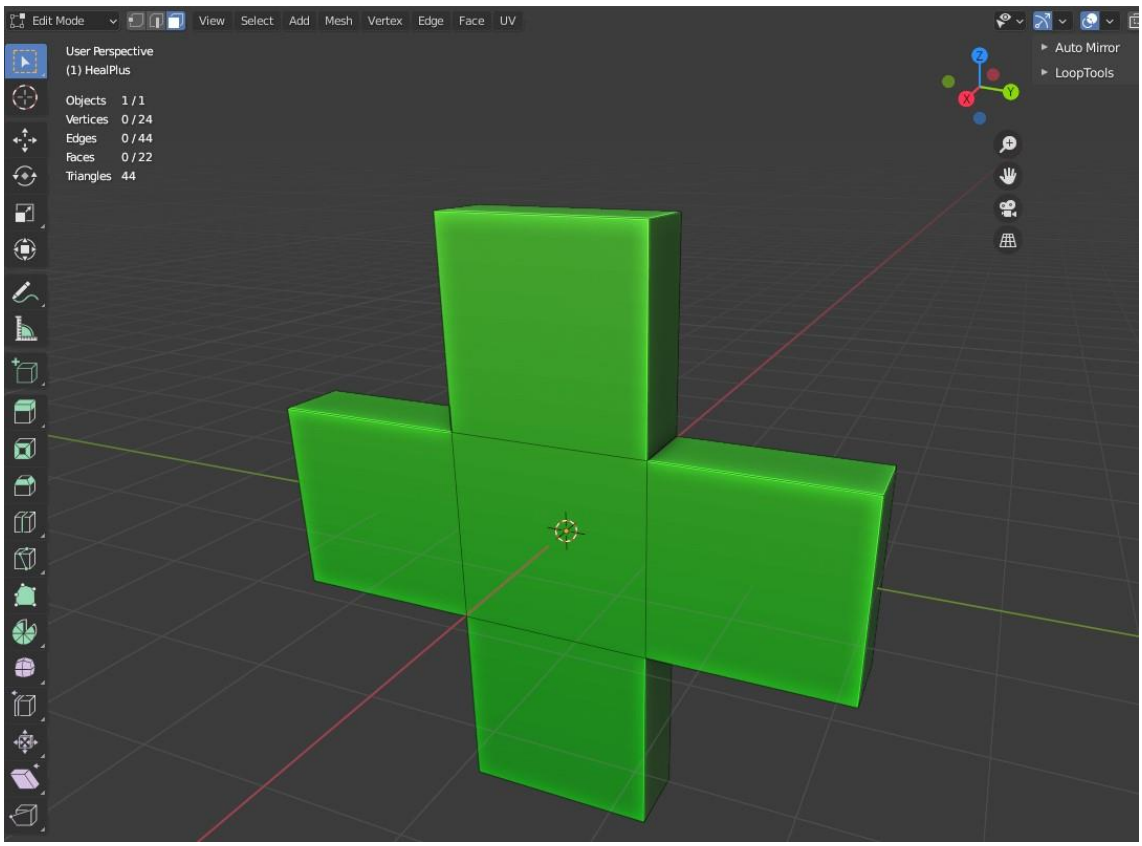
Kuva 1. Vasemmalla yksinkertainen kolme kertaa kolme tekstuuriatlas GIMP-kuvamanipulointiohjelmassa. Oikealla sama tekstuuri käytössä Unity-pelimoottorin partikkelijärjestelmässä. (Hiltunen.)

2.2 Blender-mallinnusohjelma

Blender on kolmiulotteinen, ilmainen 3D-mallinnusohjelma. Sen julkaisija on Blender Foundation. Se tukee täyttä 3D-tuotantoa, johon kuuluvat muun muassa mallinnus, renderöinti, luurankojen toteutus, animointi ja simulaatiot. Kehittyneet käyttäjät voivat jopa muokata sovellusta omaan tarkoitukseen sopivammaksi Python-ohjelmointikielellä. Blender-ohjelman lisenssi antaa vapauden käyttää sitä myös ammattilais- tai opetustarkoituksiin ilmaiseksi ja ohjelmaa voi kuka tahansa jakaakin, jos sen tekee ilmaiseksi. (Blender 2022.)

Blenderillä voi toteuttaa paljon asioita, mutta tässä opinnäytetyössä olennaisinta on 3D-mallien tuottaminen Unity-ympäristöön. 3D-mallin tekemiseen on monta

erilaista tekniikkaa. Yleisiä tekniikoita ovat esimerkiksi laatikkomallinnus tai veistäminen, joista ensimmäistä itse käytin enimmäkseen, koska sillä tavalla on helppoa pitää mallit mahdollisimman yksinkertaisina. Laatikkomallinnuksella tarkoitetaan sitä, että mallinnus aloitetaan yksinkertaisesta muodosta, kuten kuutiosta, jota editoidaan eteenpäin. Esimerkiksi voidaan tehdä kämmen laatikkona, josta sormet sitten laajennetaan omina ulokkeinaan. Mallien yksinkertaisuus on hyödyksi, jottei suorituskyky ei kärsi turhaan ylimääräisistä yksityiskohdista (kuva 2).

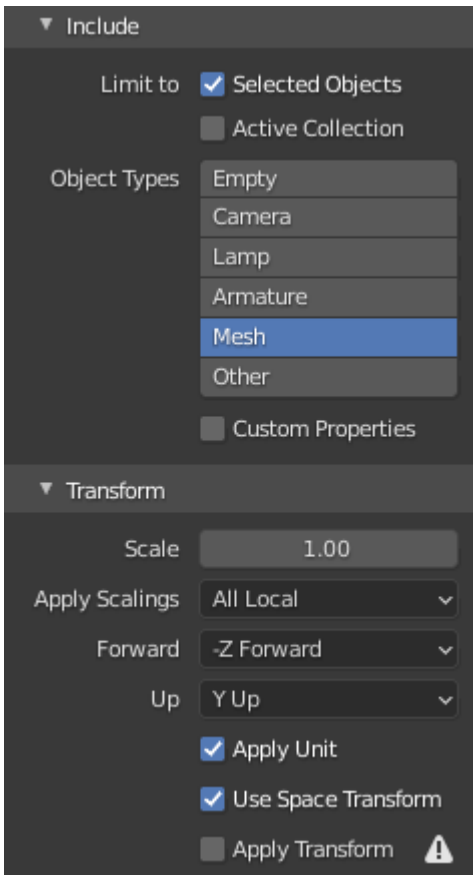


Kuva 2. Esimerkki yksinkertaisesta 3d-mallista, jota voidaan käyttää mesh-partikkelina Unityssa (Hiltunen).

Jotta Unity osaa käyttää Blenderillä toteutettua 3D-mallia, se tulee viedä Unitylle sopivaan tiedostomuotoon. Unity tukee suoraan Blenderin omaa blend-tiedostomuotoa, jotta Unity osaa kääntää tiedoston oikein, pitää käyttäjän koneelle asentaa Blender-ohjelma. Jos projektissa on useampi tekijä niin jokaisen tulisi siten asentaa Blender työskentelykoneelleen, jotta Unityn projektin tiedostot toimisi virheettömästi. Tämä on tarpeetonta vaivaa niille, jotka eivät ikinä aio käyttää Blenderiä. Parempi ratkaisu onkin käyttää yleisesti käytössä olevaa FBX-tiedostomuotoa, jota monet ohjelmat tukevat. FBX-

tiedostomuodosta on tullut alan epävirallinen standardi, koska se tukee 3D-grafiikan lisäksi myös animaatioita ja tekstuureita. Tarpeen mukaan FBX-tiedostot ovat muutettavissa toisiin tiedostomuotoihin. (Marxent 2020.) Eri ohjelmat käyttävät akseleita ylöspäin suunnan määrittämiseen (FBX 2022). Esimerkiksi Blender käyttää oletuksena Z-akselia ylöspäin ja Unity käyttää Y-akselia ylöspäin, joten sen muokkaus on olennaista viedessä FBX-muotoon.

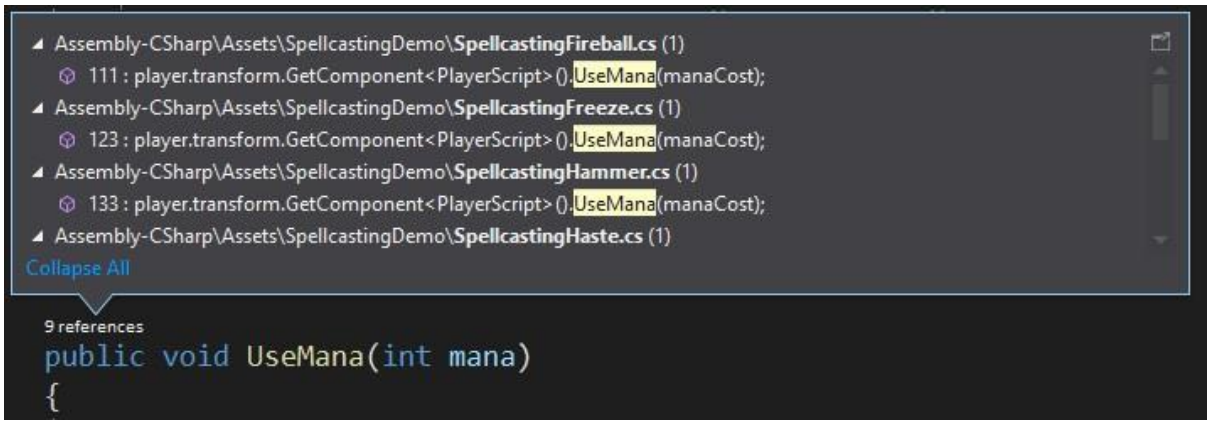
Mallin viemisessä tulee ottaa erityisesti huomioon X-, Y-, Z-akselit sekä objektin skaala (kuva 3). Näillä arvoilla Unity tietää, mitkä ovat objektin eteenpäin ja ylöspäin osoittavat puolet. Unityssa yksi mittayksikkö on metri, mutta jos Unityyn viedään esine, joka on metrin mittainen skaala-arvolla kaksi, se ilmestyy Unityyn kaksi metrisenä. Tällaisessa tapauksessa objektia sisään viemisen jälkeen joutuisi muokkaamaan Unityn puolella, jotta objekti olisi metrin mittainen. Yleisesti ottaen on helpompaa vain säätää arvot viedessä oikein Blenderin puolella, jottei niistä koidu ongelmia myöhemmin.



Kuva 3. Esimerkki Blenderissä valittavista asetuksista FBX-tiedostomuotoon viedessä (Hiltunen).

2.3 Visual Studio -koodieditori

Visual Studio on Microsoftin tekemä koodieditori Windows-käyttöjärjestelmäympäristöön. Se tukee Unity kehittämistä laajennuksen avulla, joka helpottaa paljon ohjelmointia Unityn tukemalla C#-ohjelmointikielellä. Ohjelmassa on mahdollisena laajennuksena IntelliSense-ohjelmointiapu, mikä selittää ja ehdottaa funktioita sekä muuttujia helpottaen ohjelmointia. Editori myös ilmoittaa syntaksivirheistä, jotta ne on helppoa huomata ja korjata. Havaitsin hyödylliseksi myös CodeLens-laajennuksen, koska se näyttää funktioitten päällä kätevästi, missä funktiota kutsutaan, joten on helppoa nähdä kaikki viittaukset siihen (kuva 4). CodeLens voi auttaa löytämään viittaukset koodin osiin, muutoksiin niissä, niihin liittyviin virheisiin ja testeihin (Microsoft 2022).



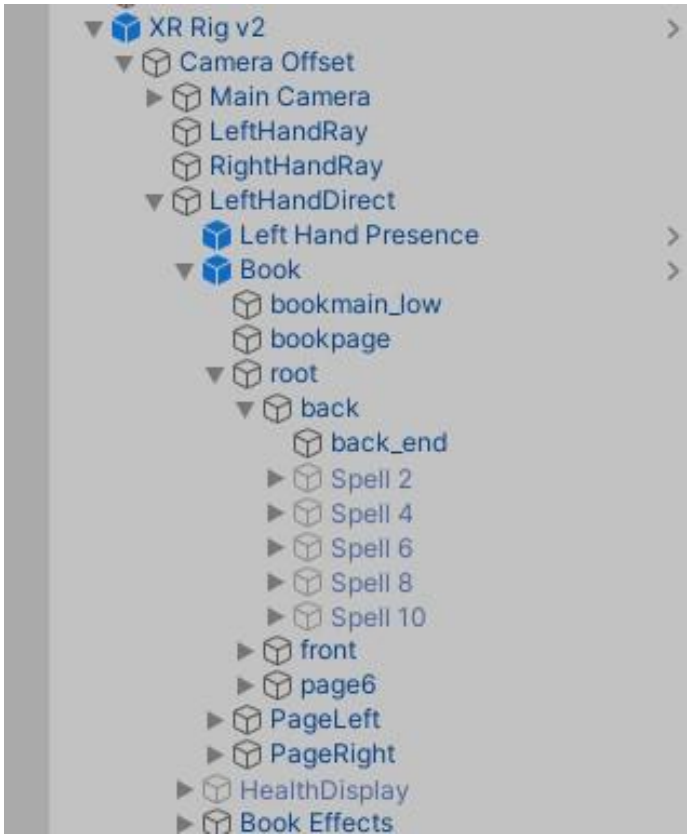
Kuva 4. Visual Studio CodeLens laajennuksen avulla viittaukset funktioon on vaivatonta löytää (Hiltunen).

2.4 Unity-pelimoottori

Unity on tällä hetkellä yksi suosituimmista pelimoottoreista, millä voidaan tehdä kaksi- ja kolmiulotteisia pelejä sekä sovelluksia. Se tukee montaa alustaa ja niiden käyttöjärjestelmiä, kuten mobiililaitteita, pöytätietokoneita sekä konsoleita (Unity 2022a). Unitylla voi aloittaa vaivattomasti ja ilmaiseksi pelien tekemisen, jos vain on intoa oppia. Työhön tarvitaan vain minimivaatimukset täyttävä tietokone, jolle voi asentaa pelimoottorin sekä paljon aikaa opetella paljon uusia asioita. Kun tekijät ansaitsevat Unitylla tehdyillä peleillä tai keräävät rahoitusta sillä tehtäviin projekteihin yli 100 000 $ vuodessa, heidän pitää ostaa maksullinen lisenssi pelimoottorille. Lisenssejä on Plus-, Pro-, ja Enterprice-

versiot ja niiden hinnat alkavat 40 $ per kuukausi. Halvin Plus-lisenssi käy, jos pysyy alle 200 000 $:n liikevaihdossa tai rahoituksessa vuodessa, mutta yli mentäessä täytyy ostaa Pro- tai Enterprise-lisenssi. Enterprise-lisenssit on tarkoitettu isommille kehitystiimeille ja lisenssi pitää ostaa vähintään 20 käyttäjälle. (Unity 2022b.)

Unityssa käytetään ohjelmointikielenä C#-kieltä (puhuttaessa se lausutaan C sharp), jolla tuotetulla ohjelmointiskriptillä voidaan hallita pelimoottorin peliobjekteja. Peliobjekti voi olla mikä tahansa pelissä oleva yksittäinen objekti, kuten esimerkiksi pelaajahahmo, valo, pöytä tai vaikkapa taivaankappale. Peliobjektien välisistä interaktioista muodostuu itse peli ja siinä tapahtuvat asiat (Unity 2022c). Unityssa peliobjekteilla on hierarkia, jossa peliobjekti voidaan laittaa toisen peliobjektin alle, jolloin siitä tulee sen peliobjektin lapsi (kuva 5). Etuna tällä on se, että lapsipeliobjektit perivät ominaisuuksia hierarkian vanhemmiltaan, kuten skaalan tai liikkeen ilman erillistä määrittelyä.



Kuva 5. Esimerkki Unityn peliobjektien hierarkiasta. Sisennetyt peliobjektit ovat lapsia hierarkiassa ylempänä oleviin peliobjekteihin (Hiltunen).

Peliobjekteihin voidaan C#-skriptien lisäksi lisätä jo valmiita komponentteja, joita pelimoottori tarjoaa. Unityssa komponentit ovat aina kiinnitettyinä peliobjekteihin. Valmiita komponentteja Unityssä on monia erilaisia, kuten vaikkapa äänilähde tai peliobjektia siirtävä animaatio, mutta tässä opinnäytetyössä tarkastellaan partikkelijärjestelmää. Unity-pelimoottori on monipuolinen työkalu pelien tuottamiseen ja sen partikkelijärjestelmä-komponentilla voidaan toteuttaa käytännössä loputtomasti erilaisia tehosteita. Rajoituksena on vain tekijän mielikuvitus ja osaaminen.

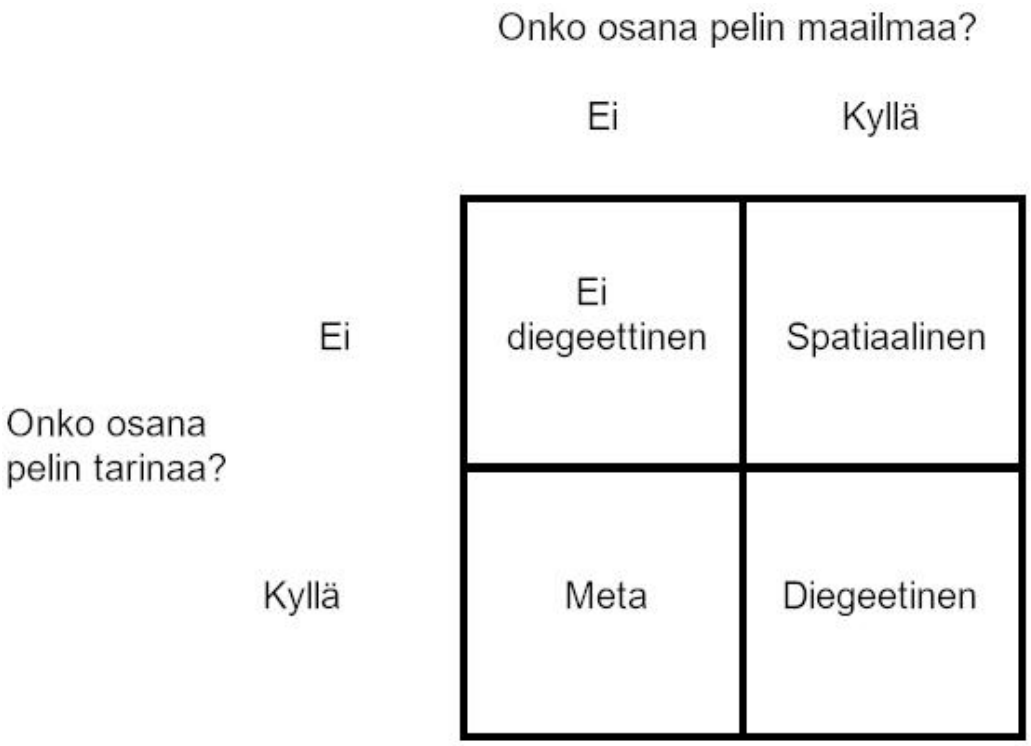
3 VR-pelien käyttöliittymien esitystapoja

Pelien tekeminen on luovaa työtä ja se antaa mahdollisuuden toteuttaa itseään loputtomilla eri tavoilla, etenkin kun uudet teknologiat, kuten virtuaalitodellisuus (VR) yleistyvät kotikäytössä. Kuitenkin peleistä ja niiden yksityiskohdista on jo kertynyt paljon tutkimusta, jotka osoittavat, mitkä asiat toimivat paremmin, kun etsitään tietynlaista vaikutusta pelaajien pelielämykseen. Silloin kun toiveena on mahdollisimman immersiivinen peli, on todettu, että pelaajan huomio pitäisi kiinnittää vain pelin sisäisiin asioihin. Käyttöliittymäkin tällöin on tärkeää saada upotettua osaksi pelin maailmaa ja tarinaa. (Iacovides, Cox, Kennedy, Cairns, Cairns & Jennet 2015.)

VR-peleissä tämä asia korostuu vielä enemmän, koska ne ovat erilaisia käyttökokemukseltaan verrattuna litteältä näytöltä katsottaviin peleihin. Tietysti pitää ottaa myös huomioon pelin teema, koska esimerkiksi digitaaliset näytöt eivät sovellu fantasiapelin ympäristöön, ellei osata aluksi selittää pelaajalle, miksi ne ovat osana pelin maailmaa ja tarinaa. VR-peleissä käyttöliittymät on suunniteltava jo lähtökohtaisesti eri tavalla kuin litteältä näytöltä katsottavissa peleissä. VR-ympäristössä käyttäjä on osana VR-maailmaa, joten käyttöliittymänkin tulee olla maailman sisällä. VR-lasien linssit ovat lähellä käyttäjän silmiä, joten suoraan näytölle laitettavat elementit, kuten heijastusnäyttö, lyhennettynä HUD (Heads-up display) ei toimi.

Käyttöliittymän esitystapoja on neljä erilaista: ei-diegeettinen, meta, spatiaalinen ja diegeettinen (kuva 6). Kun pelin käyttöliittymä ei ole osana pelin maailmaa tai pelin tarinaa, se on ei-diegeettinen käyttöliittymä. Ei-diegeetisiä käyttöliittymiä ovat esimerkiksi heijastusnäytöt, jotka vain pelaaja näkee. Jos pelin

käyttöliittymä on osana pelin maailmaa, mutta ei pelin tarinaa, se on spatiaalinen käyttöliittymä. Spatiaalisia käyttöliittymiä ovat esimerkiksi pelin maailmaan ilmestyvä inforuutu, joka kertoo suoraan pelaajalle, mistä ohjaimen napista voi tarttua esineisiin. Jos taas pelin käyttöliittymä on osana pelin tarinaa, mutta ei osana pelin maailmaa, se on meta-käyttöliittymä. Meta-käyttöliittymiä ovat esimerkiksi ruudulle ilmestyvä veriroiske, joka kertoo pelaajahahmon vahingoittuvan. Diegeettisellä käyttöliittymällä taas peleissä tarkoitetaan sitä, että on osana pelin maailmaa sekä pelin tarinaa. Diegeettisiä käyttöliittymiä ovat esimerkiksi pelihahmon kädessä pidettävä kartta ja kompassi. (Andrews 2010.) Diegeetisyyden käsite on muokattu peleihin diegeettisyysteoriasta, jota käytetään elokuvissa, teatterissa ja kirjallisuudessa (Russel 2011).



Kuva 6. Erilaisia käyttöliittymien esitystapoja havainnollistettuna (Hiltunen).

Vaikkakin tärkein käyttöliittymän tehtävä on antaa oleellista tietoa pelaajalle, pelin immersion ja pelattavuuden kannalta olisi suotavaa, että käyttöliittymä olisi suunniteltu siten, että se olisi diegeettinen sekä helppokäyttöinen. Diegeettinen käyttöliittymä voi olla pelaajalle haasteellisempaa omaksua, mutta kun pelaaja on kokenut, diegeettiset käyttöliittymät parantavat pelaajien pelielämystä verrattuna ei-diegeettisiin käyttöliittymiin, koska jälkimmäiset voivat olla usein pelielämystä häiritseviä. Pelaajat käyttävät aikaansa katsomalla käyttöjärjestelmän elementtejä, olettaen, että ne kertovat oleellista tietoa, mutta

pätevät pelaajat jo kokemuksellaan tietävät, mitä käyttöliittymä kertoo ja voivat enemmän keskittyä itse pelaamiseen. Täten pelaajat kokevat pelin immersiivisemmäksi, kun pelissä ei ole ei-diegeettistä käyttöliittymää viemässä pelaajan huomiota itse pelissä tapahtuvista asioista. (Iacovides, Cox, Kennedy, Cairns, Cairns & Jennet 2015.) Itse olen samaa mieltä edellä mainitun kanssa ja koenkin pelien immersion tärkeäksi VR-peleissä, koska se saa minut nauttimaan niistä enemmän. Mielestäni immersiolla peleissä tarkoittaa sitä, miten vahvasti pelaaja uppoutuu pelikokemukseensa, unohtaen kaiken muun sillä hetkellä. Immersiolle on useita määritelmiä ja jotkut jakavat sen myös useampiin alaluokkiinsa, kuten aistillinen, haastepohjainen ja kuvitteellinen immersio (Ermi & Mäyrä 2005).

4 Partikkeliefektit

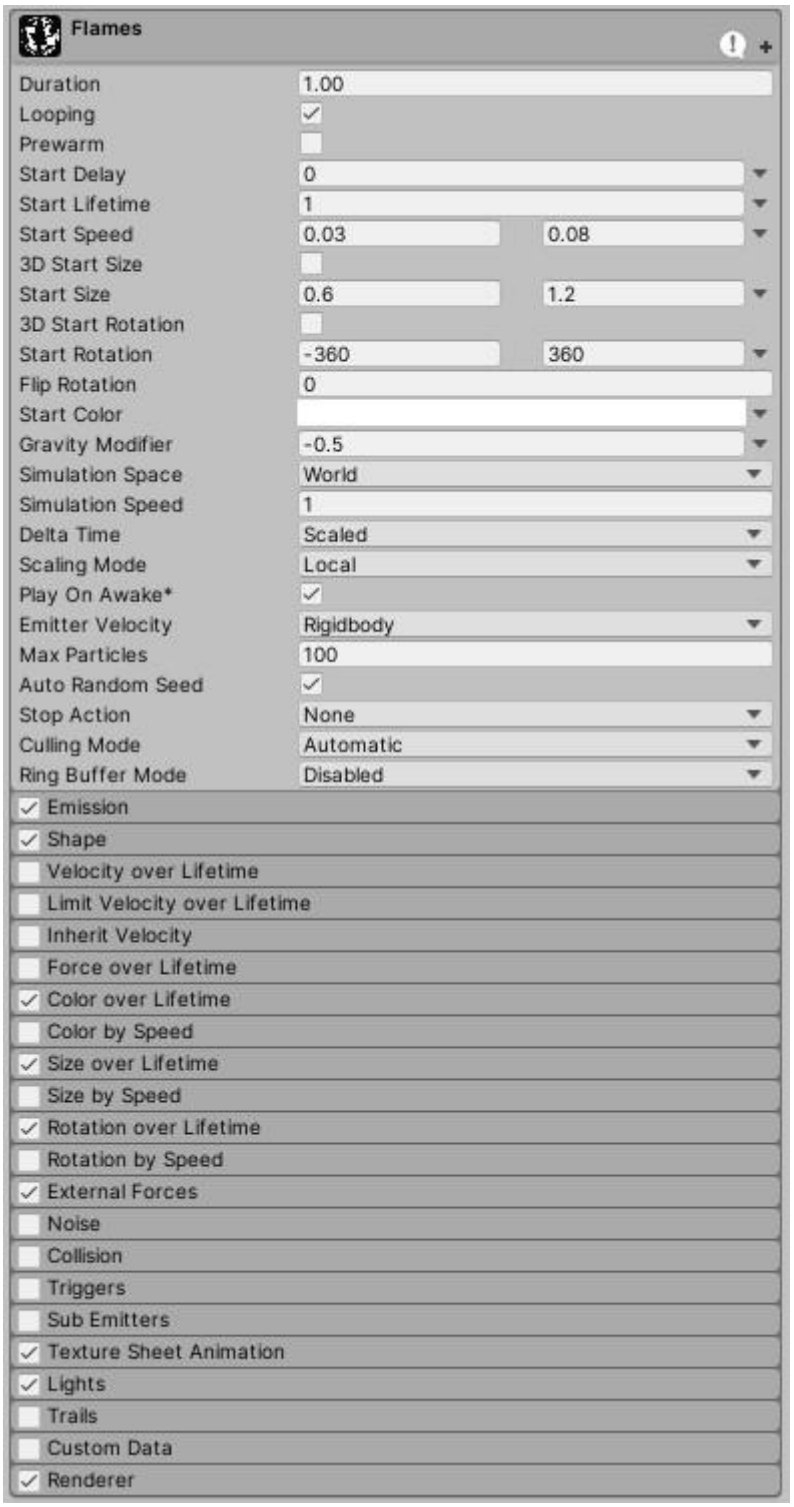
4.1 Yleistä partikkeliefekteistä

Partikkeliefekti on kokoelma pisteitä kolmiulotteisessa tilassa. Partikkelit, toisin kuten normaalit geometriset objektit, eivät ole staattisia, vaan niillä on elinkaari. Ne syntyvät, muuttuvat elinaikanaan ja sitten kuolevat pois. Partikkeliefektin parametreja säätämällä voidaan vaikuttaa partikkeleiden elinkaaren tapahtumiin, jotta saadaan aikaan haluttu efekti. (Lander 1998.) Peleissä partikkeliefekteillä toteutettavia asioita voivat olla esimerkiksi liekit, savu, kipinät tai veriroiskahdus.

Suurissa partikkelijärjestelmissä yksittäinen partikkeli käyttäytyy satunnaisesti ja tärkeämpää havainnoitsijalle usein onkin vallitseva kokonaisuus. Esimerkiksi yksittäisen vesipisaran elinkaaren seuraaminen ei yleensä ole oleellista koko sateen vaikutuksen kannalta. Yksittäiset partikkelit ovatkin kaoottisia, koska niillä ei ole täysin ennalta määrättyä tiettyä polkua vaan niillä on sattumanvarainen elementti. Tätä asiaa kutsutaan stokastiseksi prosessiksi, joka saa partikkelit vaikuttamaan luonnollisilta. (Lander 1998.) Luonnollinen käyttäytyminen on tärkeää, koska partikkeleilla usein halutaan kuvata asioita, joilla ei ole tarkkaan määrättyä muotoa. Tarkoilla parametrien säätelyillä kuitenkin voidaan myös toteuttaa partikkeliefektit siten, että yksittäiset partikkelit seuraavat joka kerta tarkkaa sijaintia sekä liikettä.

4.2 Partikkelijärjestelmät Unityssa

Unityssa partikkelit voivat olla kaksiulotteisia (2D) kuvia, kuvasarjoja tai kolmiulotteisia meshejä. Partikkelit kuitenkin aina toimivat kolmiulotteisessa tilassa, joten myös 2D-partikkeleita voidaan kiertää kolmannen ulottuvuuden kautta, tarpeen vaatiessa. Unityn partikkelijärjestelmät ovat modulaarisia ja oletuksena suurin osa moduuleista on kytketty pois päältä (kuva 7). Valmiiksi käytössä ovat vain moduulit, joita käytetään melkein jokaisessa partikkelijärjestelmässä. Esimerkiksi ottamalla käyttöön Size over Lifetime -moduulin voidaan säätää partikkelien kokoa tai Color over lifetime -moduulilla partikkelien väriä voi säätää niiden elinkaaren aikana. Riippuen siitä, mitä muokataan, voi muuttujina olla esimerkiksi vakioarvo, satunnainen arvo kahden liukuluvun väliltä tai kaari. Kaari tässä tapauksessa tarkoittaa, että jokin arvo muuttuu elinaikanaan kaaren korkeuden mukaan. Partikkelijärjestelmissä ja niitten moduuleissa on paljon muuttujia ja jokaista voidaan säätää joko editorin kautta tai C# -skriptillä. Unityn partikkelit voivat reagoida ympäristöön törmäyksillä tai lähettämällä valoa ja näitä asioita voidaan toteuttaa moduuleiden avulla.



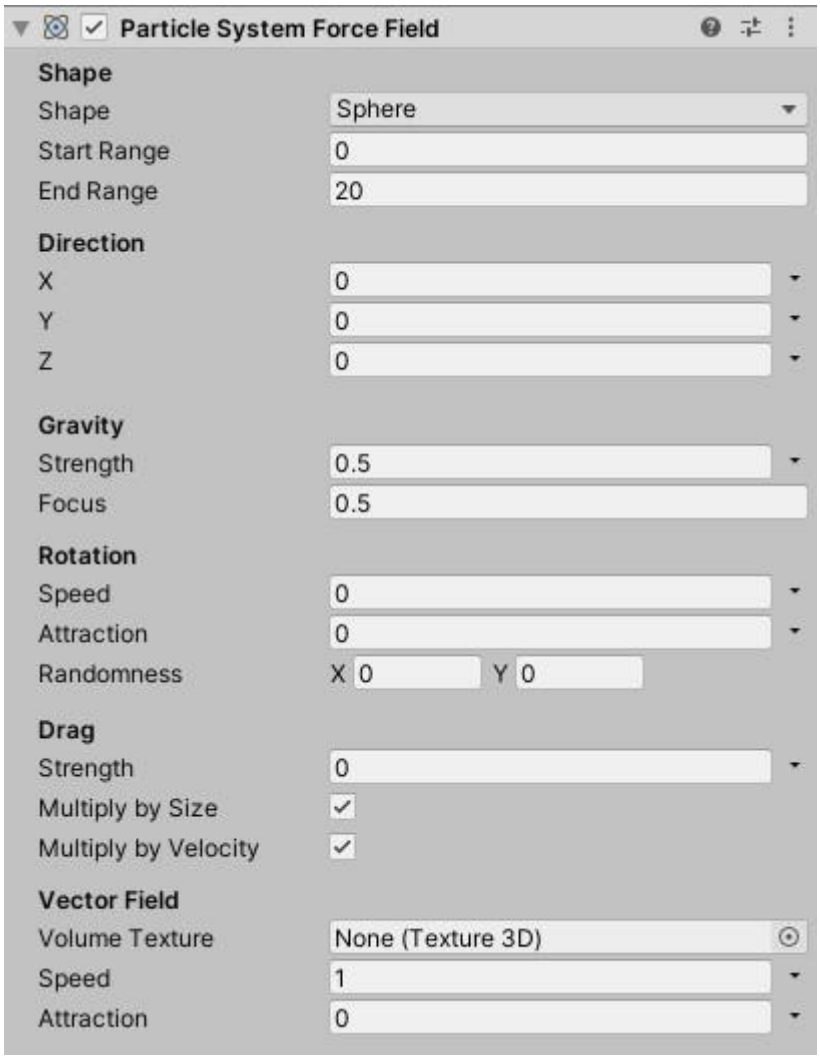
Kuva 7. Unityssa oleva Flames partikkelijärjestelmä, jossa on otettuna käyttöön useita moduuleita. Valintaruksi nimen vasemmalla puolella tarkoittaa, että moduuli on käytössä. (Hiltunen.)

Unity käsittelee partikkelijärjestelmät reaaliaikaisesti, joten partikkelijärjestelmät voivat helposti olla raskaita suorituskyvyn kannalta, etenkin, jos niitä on paljon päällekkäin tai yhteen järjestelmään on laitettu paljon emittereitä. Emitterillä tarkoitetaan lähdettä, josta partikkelit syntyvät. Se säätää, paljonko partikkeleita

syntyy määrättynä aikajaksona. Oletuksena partikkelijärjestelmässä on yksi emitteri käytössä, mutta niitä voi tarpeen vaatiessa lisätä Sub Emitters -moduulilla. Pelien sulavuuden kannalta monesti kannattaakin miettiä, mitkä efektit ovat tärkeitä ja riittääkö yksinkertainen ja tyylitelty ratkaisu toivotun efektin toteuttamiseen realismin tavoittelun sijaan. Suurimmat vaikuttajat suorituskykyyn ovat partikkeleiden määrä, niitten tekstuuri, 3D-malli, jota ne käyttävät ja se, mitä varjostinta ne käyttävät. Varjostin tarkoittaa ohjelmaa, joka kertoo näytönohjaimelle, miten pikselit pitäisi piirtää. Unityssa on oma pintavarjostintyyppinsä (englanniksi surface shader), joka kykenee reagoimaan ympäröivään valaistukseen (Unity 2022d). Jokainen näistä kohdista oli erityisen tärkeää huomioida, koska työhön liittyvä peliprojekti Kurja Demo on langattomille VR-laseille suunnattu peli. Langattomissa VR-laitteissa tällä hetkellä suorituskykybudjetti on verrattavissa mobiililaitteeseen, kuten tablettiin tai tehokkaaseen älypuhelimeen.

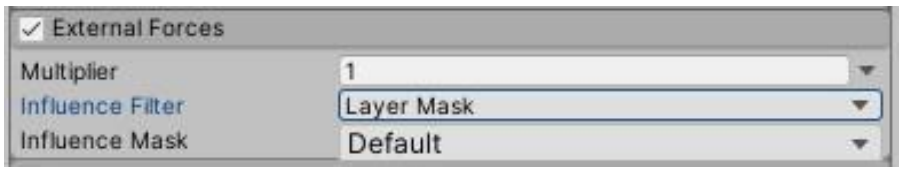
4.3 Partikkelivoimakentät Unityssa

Partikkelijärjestelmiin voi joskus tulla tarvetta vaikuttaa niitten ulkopuolelta tarkoilla parametreilla. Tähän on valmiiksi saatavilla oleva komponentti. Unity tarjoaa valmiina komponenttina partikkelijärjestelmävoimakentän, jolla voidaan käsitellä partikkeleitten käyttäytymistä partikkelijärjestelmien ulkopuolisesti. Käsittely tapahtuu antamalla partikkeleille pelimoottorin simuloimaa voimaa, kuten painovoima tai tuuli (kuva 8).



Kuva 8. Unityn partikkelijärjestelmävoimakentän muuttujia (Hiltunen).

Oletuksena partikkelijärjestelmävoimakenttä ei vaikuta ympärillään oleviin partikkeleihin, vaan partikkelijärjestelmistä täytyy aktivoida erikseen aina External Forces -moduuli, jotta kyseiseen järjestelmään voidaan vaikuttaa ulkoisilla voimakentillä. External Forces -moduuli myös antaa mahdollisuuden eritellä tarkemminkin, mitkä voimakentät vaikuttavat kyseiseen partikkelijärjestelmään (kuva 9).



Kuva 9. Partikkelijärjestelmän External Forces -moduuli ja sen sisältämät muuttujat (Hiltunen).

5 VR-pelit

5.1 Kurja Demon kuvaus ja käyttöliittymä

VR-peliprojekti, johon osallistuin tekijänä, on nimeltään Kurja Demo, joka on tehty VR-alustoille, kuten Oculus Quest ja Oculus Quest 2. Pelissä pelataan taikurin oppilaana, joka tempaistaan seikkailuun. Seikkailtaessa pelaajan tulee selvittää arvoituksia ja taistella hirviöitä vastaan. (Kurja Demo 2021.) Koska peli on toteutettu johdottomille VR-laseille niin käytössä oleva laskentateho on rajallinen. Tämän takia pelin pitämiseksi sujuvana täytyy pelin sisältö pitää yksinkertaisena ja piirtoetäisyys rajattuna. Piirtoetäisyydellä tarkoitan tässä tapauksessa sitä, miten kauan pelaaja näkee. Tätä rajoitetaan sumulla, joka värjää pelimaailman määritetyllä värillä tihenevästi etäisyyden kasvaessa, kunnes se on ainoa asia, joka on nähtävissä.

Lähtökohtaisesti Kurja Demo oli tarkoituksena tehdä koko pelin siten, että siinä ei ole yhtään kielen käyttöä tekstinä tai puheena. Käyttöliittymä Kurja Demossa onkin suurimmaksi osaksi diegeettinen. Pelissä käyttöliittymää opetetaan käyttämään lyhyillä spatiaalisilla animaatioilla, jotka rikkovat neljännen seinän. Neljäs seinä -ilmaisu tulee kuvitteellisesta seinästä teatterilavan ja katsojien välillä. Jos se ”rikotaan” niin se tarkoittaa, että jokin teatteriesityksessä suoraan viestii katsojalle oikeaan maailmaan. Tässä tapauksessa suoraan viestitään pelaajalle se, mistä ohjaimen napista tapahtuu mitäkin. Kun Kurja Demossa pelaaja vahingoittuu, ilmestyy partikkeliefekti, joka on esitystavaltaan meta-käyttöliittymä.

5.2 Half-Life: Alyx vertailukohteena

Half-Life: Alyx on Valve:n tekemä VR-peli, jossa pelaaja laitetaan Alyx-nimisen hahmon tilalle tulevaisuuden maailmaan, jossa maapallon on vallannut sen ulkopuolinen taho. Pelissä on suuressa osassa ympäristön tutkiminen ja taistelu erilaisia vihollisia vastaan. Tässä pelissä ei taiota, kuten Kurja Demossa, mutta siinä pelaaja saa käyttöönsä aseita, jotka käyttävät ammuksia. Pelissä on diegeettinen käyttöliittymä suurimmaksi osin, mutta neljännen seinän rikkovia spatiaalisia apuvihjeitä löytyy maailmasta, kuten esimerkiksi, miten kiivetä tikkaita pelissä.

Half-Life: Alyx on kriitikkojen suosiossa. Esimerkiksi Metacritic-sivustolla perustuen 69:n kriitikon arvosanoihin, pelille on annettu 93 yhteisarvosanaksi asteikolla 0–100 (Metacritic 2022). Itse olen pelannut pelin alusta loppuun kaksi kertaa ja olen samaa mieltä kriitikkojen kanssa, että peli on erittäin hyvin toteutettu jokaisella osa-alueella. VR-pelejä on nykyisin jo todella paljon saatavilla ja syy miksi valitsin vertailukohteeksi Half-Life: Alyx -pelin, koska molemmissa peleissä on pääasiassa samanlainen käyttöliittymätyyppi. Lisäksi haluan esittää, miten jossain toisessa pelissä sama asia on toteutettu eri tavalla saman tyyppisessä käyttöliittymässä, mutta lopputuloksena pelaajalle välittyy käytännössä sama tieto: miten paljon pelaajalla on käytössään resursseja. Taikavoimia vertaan ammuksiin ja elinvoimaa elinvoimaan.

Half-Life: Alyx -pelissä viestitään elinvoimia pelaajan vasemman käden hanskan kämmenselässä olevalla näytöllä. Näytössä on kolme sydämen kuvaa, jotka vaihtavat kokoaan, riippuen pelaajan elinvoiman määrästä (kuva 10). Näyttö on tyylitelty vahvasti, minkä takia pelihahmon tarkan elinvoiman määrän selvittäminen voi olla haastavaa pelaajalle. Kuitenkin näyttö on porrastettu tarpeeksi erikokoisiin osiin, jotta nopea elinvoiman määrän arviointi tilanteessa on mahdollista.

Kuva 10. Pelaajan elinvoima viestitään pelaajalle tyylitellyillä sydänkuvioilla. Ylempänä arviolta noin 50 % ja alempana 100 % elinvoimaa. Hanskassa oleva alempi näyttö kertoo resin-resurssin määrän, jolla pelaaja voi päivittää aseitaan, mutta sama näyttö myös näyttää sillä hetkellä käytössä olevan aseen kaikkien ammusten lukumäärän. (Half-Life: Alyx, 2020.)

Jokaisessa Half-Life: Alyx -pelin aseessa on omanlaisensa indikaattorit siihen, miten paljon niissä on ammuksia jäljellä. Otan esimerkiksi pelin ensimmäisenä saatavan aseen, joka on pistooli (kuva 11). Pistoolista on helppo nähdä nopealla vilkaisulla, onko siinä ammuksia jäljellä, mutta ammusten tarkka laskeminen vaatii hetken keskittymistä. Pistoolin tilan viestimiseen pelaajalle käytetään ääniä, valoja ja materiaaleja. Tämän lisäksi on helppoa huomata vilkaisulla, onko pistoolin lipas paikallaan vai ei.

Kuva 11. Valot aseessa näyttävät pelaajalle, kuinka monta ammusta on jäljellä. Ylempänä täysi lipas paikallaan, josta yksi ammus on jo piipussa. Alempana tyhjä ase ja piippu, sekä aseen lipas on poistettu. (Half-Life: Alyx, 2020.)

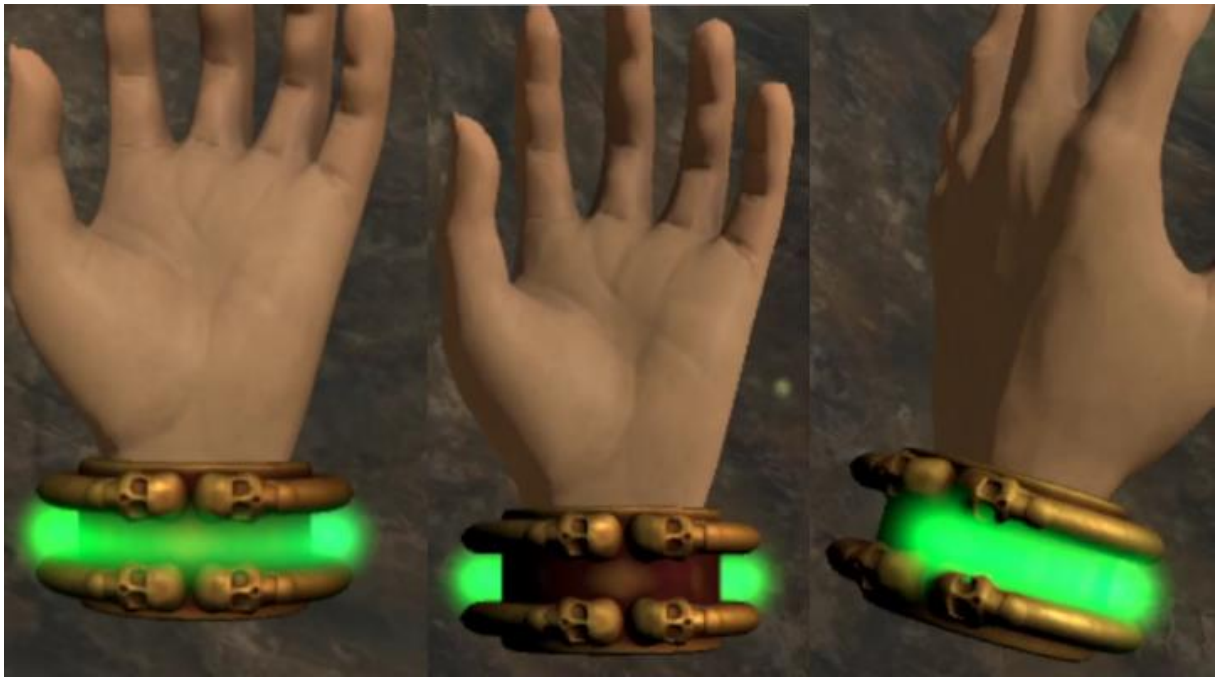
6 Pelaajalle viestintä partikkeleilla Kurja Demossa

6.1 Elinvoimaranneke

Kurja Demossa pelaaja voi vahingoittua monissa eri muodoissa ja paikoissa. Tätä varten on peliin rakennettu pelaajan yhtenä ominaisuutena elinvoima, jonka määrä on myös hyvä viestittää pelaajalle. Jos pelaaja vahingoittuu, hänen elinvoimansa laskevat ja siten myös rannekkeessa olevan elinvoima mittarin

tulee näyttää oikea elinvoiman määrä. Kun pelaaja tietää, kuinka paljon elinvoimaa on jäljellä, hän voi pelaaja punnita riskinottoa paremmin.

Keino viestiä elinvoiman määrä pelaajalle Kurja Demossa haluttiin toteuttaa diegeettisesti. Kurja Demossa pelaajalla ei ole varsinaista kehoa, vaan pelaajalla on leijuvat kädet. Virtuaalitodellisuuspeleissä yleisin keino toteuttaa pelaajan interaktiot ympäristön kanssa on käsin koskettamalla tai osoittamalla, joten käyttöliittymän elementtien liittäminen pelihahmon käsiin on luonnollista, koska ne ovat pelaajan näkökentässä melkein koko pelaamisen ajan. Kurja Demossa päädyimme vasempaan käteen kiinnitettyyn rannekkeeseen, josta voi nähdä pelaajan senhetkisen elinvoiman määrän tavalla, joka vastaa rannekellon katsomista (kuva 12). Partikkelit syntyvät rannekkeen ympärillä peräkkäin elliptisen ympyrän reunalla jatkuvasti päivittyen, saaden aikaan kuvitelman ranteen ympärillä pyörivästä liikkeestä.



Kuva 12. Vihreät partikkelit ympäröivät pelaajahahmon rannetta enemmän tai vähemmän pelaajahahmon elinvoiman määrän mukaan. Vasemmalta oikealle: 100 %, 50 % ja 25 % elinvoimaa jäljellä. (Hiltunen.)

Rannekkeen pelaaja saa itselleen suorittamalla tehtävän, jossa hänen tulee ratkaista arvoitus. Arvoituksessa pelaajan täytyy käännellä vipuja oikeisiin

asentoihin. Tätä helpottamaan peliin on luotu vihjekäärö, josta pelaaja voi päätellä oikeat vipujen asennot. Tällä tavoin ranneke sisältyy pelin tarinaan ja etenemiseen.

Ranneke on pelaajan vasemman käden peliobjektin lapsena, jolloin se automaattisesti seuraan rannekkeen sijaintia pelimaailmassa. Partikkelijärjestelmä taas on kiinnitetty rannekkeen peliobjektin lapseksi. Partikkelijärjestelmässä ovat käytössä Main-, Emission-, Shape-, Color over Lifetime- sekä Renderer-moduulit. Partikkeleiden tekstuurina käytin Unityn oletuspartikkelitekstuuria, mutta materiaalin varjostinta on muokattu siten, että se ei reagoi sumuun. Partikkeleiden renderöintimoodi on asetettu siten, että niiden etupuoli on aina kohdistettu kameraan päin.

Shape-moduulissa säädetään partikkelijärjestelmän muoto. Muodoksi on valittuna ympyrä, mutta koska ranneke ei ole symmetrinen ympyrä niin sen Y-akselia on skaalattu pienemmäksi. Tällä tavoin saadaan partikkelijärjestelmälle toivottu elliptinen muoto. Partikkelit tässä tapauksessa halutaan vain syntyvän litistetyn ympyrän ulkopinnalla, ja se määritellään Radius thickness -kentässä.

Pelaajan elinvoiman määrän muuttuessa pitää partikkelijärjestelmän muuttua sen mukana, jotta se viestittää pelaajalle elinvoiman oikean määrän. Tämä tapahtuu C# -skriptissä. Partikkelijärjestelmän Shape-moduulissa oleva ympyrän kaaren arvo määrätään asteilla. C# -skriptissä voidaan säätää partikkelijärjestelmän ympyrän kaaren arvoa kertolaskulla ja rajoittamalla tulosta Unityn Mathf -matematiikkafunktiokokoelman Clamp-funktiolla (kuva 13). Minimi- ja maksimiarvojen rajoittaminen Clamp-funktiolla estää partikkeleita joko menemästä väärään suuntaan tai päällekkäin. Lopputuloksena tässä on rannekkeen ympärille syntyvien partikkelien kaaren rajoittaminen sitä pienemmäksi, mitä vähemmän pelaajalla on elinvoimaa. Pelaaja näkee elinvoimansa siitä, miten pitkällä alueella on partikkeleja vasemman käden ranteen ympärillä.

```
private void UpdateWristHealth()
{
    ParticleSystem.ShapeModule wHShape = wristHealthDisplay.shape;
    wHShape.arc = Mathf.Clamp(currentHealth * 3.6f, 0.001f, 360.0f);
}
```

Kuva 13. UpdateWristHealth -funktiossa partikkelijärjestelmän muotoa muokataan pelaajan elinvoiman määrän mukaan (Hiltunen).

Kurja Demossa yksi taioista muokkaa pelaajan kokoa suuremmaksi joksikin aikaa, jonka jälkeen pelaajan koko palaa automaattisesti takaisin normaaliksi. Elinvoimarannekkeen partikkelijärjestelmä tulee näyttää suhteessa samalta pelaaja hahmon koosta riippumatta. Pelaajahahmon koon mukaan arvoja tulee muuttaa C# -skriptissä. Olennaiset asiat, joita tulee muokata, ovat Shape-moduulin muodon skaala, Main-moduulin partikkelien koko sekä Emission-moduulin partikkelien määrä. Kaikki edellä mainittujen pelaajan normaaliskaalan arvot kerrotaan funktion sisään ottamalla scaleAmount-muuttujalla (kuva 14). Partikkelien värit määritellään Color over Lifetime -moduulissa. Sieltä partikkelien väriksi valitsin väriliu'un, joka menee vihreästä hieman sinertäväksi elinkaaren lopussa.

```
IEnumerator ScalePlayerSize(float scaleAmount)
{
    float currentTime = 0.0f;
    float scaleTime = 1.0f;
    //particles
    ParticleSystem.ShapeModule hdShape = healthDisplay.shape;
    hdShape.radius = 0.05f * scaleAmount;
    ParticleSystem.MainModule hdMain = healthDisplay.main;
    hdMain.startSize = 0.05f * scaleAmount;
    ParticleSystem.EmissionModule hdEmission = healthDisplay.emission;
    hdEmission.rateOverTime = 50 * scaleAmount;

    //player
```

Kuva 14. Pelaajan skaalan muuttuessa pitää myös muokata elinvoimarannekkeen partikkelijärjestelmän arvoja, jotta ulkoasu säilyy yhtenäisenä (Hiltunen).

6.2 Taikavoimamittari

Kurja Demossa pelaaja voi edetessään oppia taikomaan useilla eri taioilla ja jokaisella taialla on erilaiset ominaisuudet. Yksi keino tasapainottaa taikoja tai rajoittaa niiden käyttämistä on antaa niille hinta, joka pelaajan pitää maksaa, jotta voi niitä taikoa. Tässä päädyimme yksinkertaiseen taikavoimaan, jota pelaajalla on rajattu määrä, mutta joka regeneroituu itsestään ajan kuluessa. Taikavoiman määrä on pelaajan hyvä tietää, koska eri taiat maksavat eri

määrän taikavoimaa. Koska pelissä taiottaessa pelaaja tarvitsee taikasauvan taikoessaan, niin oli luonnollista yhdistää osana sitä taikavoimamittari.

Taikasauva ja taikakuvioiden piirtäminen sillä on toteutettu diegeettisesti ja taikavoimamittari on osana taikasauvaa. Kun pelaaja ottaa taikasauvan esille niin pelaajan käsi piilotetaan, koska koimme sen turhaksi. Tämä myös viestii tavallaan pelaajalle sen, että kun hänellä on käsi näkyvissä niin hän voivat tarttua sillä ympäristössä oleviin esineisin, mutta kun taikasauva on käden tilalla niin pelaaja ei voi enää tarttua mihinkään sillä. Mittari ympäröi taikasauvan vartta, jatkuvasti päivittyen määrättyjen arvojen mukaisesti (kuva 15).

Kuva 15. Taikavoimamittarin viestimä taikavoiman määrä. Vasemmalta oikealle: 100 %, 50 % ja 25 % taikavoimaa jäljellä. (Hiltunen.)

Taikasauvan keräämiseksi ja elvyttämiseen toimintakuntoon pelaajan tulee suorittaa tehtäviä ja niiden lisäksi pelaajan pitää kerätä taiat taikakirjaansa, ennen kuin voi niitä käyttää. Taikasauva on täten osana pelin maailmaa ja tarinaa.

Taikavoimamittari on taikasauva peliobjektin lapsena, jotta se seuraa taikasauvan sijaintia pelimaailmassa tarkasti. Partikkelijärjestelmän päämoduulin lisäksi ovat käytössä Emission-, Shape, Velocity over Lifetime, Color over Lifetime ja Renderer-moduulit. Partikkeleiden tekstuurina käytin

Unityn oletuspartikkelitekstuuria, mutta materiaalin varjostinta on muokattu siten, että se ei reagoi sumuun. Partikkeleiden etupuoli on aina kohdistettu kameraan päin Renderer-moduulin Render Moden Billboard -asetuksella.

Partikkelijärjestelmälle määritellään ympyrän muoto Shape-moduulissa, joka on karkeasti taikasauvan päädyn kokoinen halkaisijaltaan. Velocity over Lifetime -moduulissa annetaan partikkeleille niiden elinkaaren aikana tapahtuvaa liikettä Y-akselilla, jolla ne saadaan liikkumaan taikasauvan varren suuntaan. Koska partikkelit syntyvät jatkuvasti partikkelijärjestelmän muodon reunalle peräkkäin ympyrässä ja liikkuvat tasaisesti Y-akselilla niin luodaan aikaan kuvitelma, että partikkelijärjestelmä olisi pyörivässä liikkeessä. Kuitenkin itse partikkelijärjestelmä liikkuu ainoastaan sauvan pelimaailman sijainnin mukana ja sille ei ole määritelty muuta liikettä.

Partikkelijärjestelmä on suurilta osin toteutettu kuten aiemmin kuvattu elinvoimarannekkeen partikkelijärjestelmä. Ero kuitenkin on siinä, miten pelaaja näkee arvon määrän. Elinvoima rajoitti ympyrän kaaren kokoa, mutta taikavoimamittarissa rajoitetaan Y-akselilla tapahtuvaa partikkelien elinkaaren aikana tapahtuvaa liikettä. Sillä rajoitetaan, kuinka kauas partikkelit liikkuvat taikasauvan päädystä kärkeen päin. Sopivat arvot tälle löytyivät pääasiassa testauksen kautta kokeilemalla.

Pelaajan taikavoimien määrää hallitaan funktiolla pelaajan C#-skriptissä, josta pelaajan taikavoiman määrää seurataan. Unityn Mathf -matematiikkafunktiokokoelman Clamp-funktiolla rajoitetaan Y-akselilla tapahtuvaa liikettä (kuva 16). Clamp-funktio estää arvoa menemästä yli tai alle toivottujen arvojen. Funktiota on tarkoitus kutsua, kun pelaajan taikavoiman määrä muuttuu, jotta taikavoimamittari näyttää määrän oikein. Pelaajalle tämä viestittyy lyhyempänä efektinä sauvan varren ympärillä sitä mukaa mitä vähemmän taikavoimia on käytettävissä.

```
private void UpdateWandMana()
{
    ParticleSystem.VelocityOverLifetimeModule ySpeed = wandManaDisplay.velocityOverLifetime;
    ySpeed.y = Mathf.Clamp(currentMana * 0.002f, 0.0001f, 0.2f);
}
```

Kuva 16. Taikavoimamittarin partikkelien elinaikainen liike Y-akselilla hallitaan C#-skriptissä UpdateWandMana-funktiolla (Hiltunen).

Color over Lifetime -moduulista partikkelien väriksi valitsin väriliu'un, joka alkaa sinisestä ja päättyy violettiin. Kuten aiemmin elinvoimarannekkeen partikkelien kanssa, päätin tietoisesti olla vaihtelematta partikkelien väriä taikavoimien määrän vaihtelun aikana.

6.3 Partikkeliarvoitus

Pelaajille on Kurja demossa erilaisia pulmia, joiden ratkaisemisesta he saavat palkintoja. Palkintoina voi olla esimerkiksi hattuja pelaajaa seuraavalle lentävälle kallolle tai esineitä, kuten parannusjuomapullo. Tästä partikkeliarvoitusaskareesta pelaaja saa itselleen palkinnoksi taikaikkunan, jolla voi nähdä muuten näkymättömiä asioita pelimaailmassa (kuva 17). Tätä taikaikkunaa pelaaja tarvitsee pelissä etenemiseen ja pelimaailman tutkimiseen.

Kuva 17. Taikaikkunan läpi katsottaessa pystyy näkemään piilotettuja asioita ja vihjeitä (Hiltunen).

Pelaajan liikkuessa tarpeeksi lähelle aluetta, jossa partikkeliarvoitus on, pelaaja nopeasti ympäröidään kasvavalla puolipallolla. Ontto puolipallo estää pelaajaa liikkumasta pois alueelta, ennen kuin tehtävä on ratkaistu pelaajan toimesta. Puolipallo toimii myös taustana ja on musta väriltään, jotta kirkkaat partikkelit erottautuvat selkeästi siitä. Arvoituksessa on kolme partikkelijärjestelmää, joita

pelaaja voi hallita fysiikkapohjaisella vivulla. Vivussa on tartuntakahva, johon pelaaja voi tarttua (kuva 18). Partikkelien nopeus ja kohina on sitä suurempi, mitä kauempana vivun kärki on esimerkkikuviosta. Tarkoituksena olisi saada villisti liikkuvat partikkelit toimimaan kuten esimerkkipartikkelikuviossa partikkelit koko ajan toistuvat. Tämä toteutuu pitämällä vivun kärkeä paikallaan esimerkkikuvion päällä tarpeeksi pitkään, jolloin etenemisindikaattorina toimiva partikkelijärjestelmän partikkelit kasvattavat kokoaan, kunnes ne ovat maksimikoossaan ja vaihtavat väriään. Kun pelaaja on suorittanut yhden osion, esimerkkikuvio ja hallittavat partikkelit häviävät. Kuvioina ovat neliö, kolmio ja ympyrä. Jokainen näistä kolmesta pitää suorittaa samalla tavalla, minkä jälkeen alueen ympäröivä musta puolipallo häviää kutistumalla. Tämän jälkeen taikaikkuna ilmestyy ilmaan leijumaan pelaajan kerättäväksi. Jokaisessa vaiheessa on mukana myös ääniefektejä viestimässä etenemisestä visuaalisten indikaattorien lisäksi. Tätä kaikkea hallitsee c#-skripti.

Kuva 18. 1 kuvassa on indikaattori, joka ilmoittaa kasvamalla etenemisestä. 2 kuvassa on hallittava partikkelijärjestelmä, joka pitäisi saada muistuttamaan 3:n partikkelijärjestelmää pitämällä vivun kärkeä sen päällä. (Hiltunen.)

Vipu, jolla pelaaja hallitsee partikkelijärjestelmiä, on toteutettu tartuttavaksi VR-ohjaimella ja on paikallaan fysiikkanivelellä. Vipua voi käännellä, muttei liikuttaa pois paikoiltaan. Vivun kärjessä on peliobjekti, josta partikkelijärjestelmiä hallitseva c#-skriptissä oleva itseään toistava rutiini mittaa jatkuvasti etäisyyttä kymmenesosasekunnin välein. Vivun sijainti on yhtä kaukana jokaisesta kohteesta, joten se yltää jokaiseen kohteeseen kääntämällä. Etäisyyden mukaan partikkelijärjestelmien valittuja arvoja muutetaan suuremmaksi tai pienemmäksi. Arvojen muuttumisen pelaaja suoraan näkee partikkelien käyttäytymisessä. Käytössä hallittavissa partikkelijärjestelmissä ovat Emission-, Size over Lifetime-, Rotation over Lifetime-, Noise- ja Renderer-moduulit. Poikkeuksena on ympyräpartikkelit, joissa Rotation over Lifetime -moduulilla ei ole käyttötarkoitusta, koska partikkelit on kohdistettu pelaajaa päin, joten ympyrä näyttää aina samalta riippumatta sen kiertoasteista. Noise-moduulissa voi partikkelien liikkeelle antaa turbulenssimaista kohinaa (kuva 19). Tässä tapauksessa vivun kärjen etäisyys esimerkkikuviosta muokkaa kohinan voimakkuutta ja kolmio- sekä neliöpartikkelien kiertoa.

Kuva 19. Partikkelijärjestelmän Noise-moduulin muuttujia (Hiltunen).

Jo aikaisessa vaiheessa partikkeliarvoitusta tehdessäni huomasin, että vivun aivan tarkalleen paikallaan pitäminen ei ole realistinen tavoite pelaajalle.

Toimivaksi tavaksi tehdä tämä oli toteuttaa koodiin ehtona ratkaisu, että jos vivun kärki on vain tarpeeksi lähellä kohdetta, niin kohina- ja kiertoarvo asettuu nollaksi pysäyttäen partikkelit. Tämä tarpeeksi lähellä oleva arvo löytyi testaamalla. Indikaattorina toimivat partikkelit kasvavat asteittain, skriptissä muokatulla partikkelien syntymiskoolla osoittaen pelaajalle edistymisestä. Lopulta ne muuttuvat hallittavien partikkelien värisiksi ilmoittaen osion valmistumisesta.

7 Pohdinta

7.1 Yleistä

Dreal-yrityksessä on viisi henkilöä, joista neljä ovat varsinaisia pelikehittäjiä tai tekijöitä. Tämä oli meille ensimmäinen suurempi projekti yhdessä, joten projektin tarkoituksena oli oppia ryhmänä työskentelyä sekä ottaa tiiminä haltuun kaikki tarvittavat työkalut, jota pelituotanto vaatii Unity-ympäristössä. Aiempia pelijulkaisuja on muutamilla ryhmän tekijöillä, joten aivan aloittelijoista ei ole kysymys. Tärkeänä osana oli jokaiselle lisäkokemuksen kartuttaminen, toisten kanssa tekeminen ja ainakin toistaiseksi koimme jokainen erittäin tärkeäksi sen, että tekeminen olisi hauskaa. Kehittäminen alkoi yleensä ideoitten kokeilemisella. Useat näistä ideoista jäivät elämään toisiltamme saadun palautteen sekä jatkokehityksen kautta. Tämä prosessi opetti paljon hyödyllisiä asioita ja haastamaan ongelmanratkaisukykyjämme. VR-alustan suorituskyvyn kartoittaminen tuli kokeilun ja testauksen kautta myös selväksi tiimille. Toimivan pelikokonaisuuden kannalta tämä on tärkeää tietoa ja osaamista.

Koska pelintekijän näkökulma on lähtökohtaisesti erilainen kuin pelaajan, tällä hetkellä emme vielä osaa sanoa sitä, miten intuitiivinen käyttöliittymä kokonaisuutena on käyttää suuremmalla käyttäjäkunnalla. Odotamme testausta ja käyttökokemuksia isommalla ryhmällä käyttäjiä ja odotamme palautetta demon pelaajilta, kun niitä tulee enemmän ajan myötä.

Partikkeliarvoituksen koki osa tiiminjäsenistä ja muista lähipiirin testaajista haastavaksi ymmärtää, joten sitä pitikin muokata helpommaksi, jotta lopputulos oli sulavampi. Sitä tehdessä opinkin hyvin, miten voi olla vaikeaa saada pelaajat

ymmärtämään omasta mielestäni loogista tai muuten selkeää asiaa. Tämä korostaa tarvetta saada testaajia peliin toteutetuille asioille, jotta ne eivät ole liian vaikeita tai turhauttavia, eivätkä estä pelistä nauttimista.

7.2 Elinvoiman toteutus

Testattaessa elinvoimaranneketta myös vaihtelin partikkelien värejä, kun pelaaja vahingoittuu. Pelaajalla ollessa alle 50 % elinvoimia väri vaihtui keltaiseksi ja alle 25 % punaiseksi, mutta lopulta otin värinvaihtelun pois, koska se ei omasta mielestäni näyttänyt hyvältä eikä se tuntunut tarpeelliselta. Elinvoima- ja taikavoimamittareista ei ole helppoa tulkita yksittäistä tarkkaa arvoa, kuten esimerkiksi onko pelaajalla tietyllä hetkellä 72 vai 76 elin- tai taikavoimaa jäljellä. Kuitenkin kokeneempi pelaaja pystyy arvioimaan summittaisen arvon nopealla vilkaisulla. Eri mieltä voi olla siitä, mikä tiedonvälityskeino on toimivin. On varmasti myös sellaisia pelaajia, jotka toivoisivat tarkkaa arvoa tällaisen summittaisen arvon sijaan.

Elinvoiman määrän viestiminen molemmissa Half-Life: Alyx- ja Kurja Demo -peleissä on aika samanlainen ajatustasoltaan omasta mielestäni, koska summittaisen määrän arviointi vilkaisulla molemmissa tapauksissa on nopeaa tehdä. Tarkan määrän arviointi molemmissa tapauksissa taas on haastavaa, ellei mahdotonta, mutta toisaalta omasta mielestäni myös monesti tarpeetonta. Esimerkiksi, jos vihollisen ammus tekee 30 vahinkoa ja pelaaja kuolee elinvoiman ollessa nolla tai pienempi niin ei ole ollenkaan pelaajan kannalta merkitystä sillä, onko hänellä 1 vai 29 elinvoimaa. Jokainen ammus tappaa pelaajan joka tapauksessa. Tällaisessa tilanteessa pelaajan täytyy vain yrittää välttää jokaista vaaraa, joka hänen eteensä laitetaan, jottei hänen pelihahmonsa menehtyisi. Pelatessani Half-Life: Alyx -peliä itselle tulikin tunne, että elinvoiman summittainen arviointi auttoi riskien ottamisessa ja siinä, miten varovaisesti käyttäydyn taistellessani. Mielestäni Kurja Demossa oleva elinvoimaranneke sai minut käyttäytymään samalla tavalla. Väittäisin, että molemmissa peleissä elinvoimamittarit täyttävät tarkoituksensa, vaikkakin ovat tekniseltä toteutukseltaan todella erilaisia. Jatkokehittämistä vaatisi se, että Kurja Demossa elinvoimarannekkeen ominaisuutta ei opeteta pelaajalle mitenkään, vaan se pitäisi itse oivaltaa. Toisin kuten Half-Life: Alyx -pelissä

pelaajalle selitetään selkeästi mitä vasemman käden hanskan sydännäyttö tekee.

7.3 Ammukset ja taikavoima

Ammusten määrä Half-Life: Alyx -pelissä on nopea arvioida vilkaisulla ja sen lisäksi niiden tarkka määrä on selvitettävissä, toisin kuten toteuttamassani taikavoimamittarissa. Jo nopealla tarkastelulla huomaa, että Alyxin pistooli on moninkertaisesti monimutkaisempi kuin Kurja Demon taikasauva. Kuitenkin molempia pelejä pelatessa ja kokemuksen kasvaessa pelaaja saa tuntuman siitä, että nopea vilkaisu taikavoiman tai ammusten määrästä auttaa arvioimaan tilannetta, joten omasta mielestäni indikaattorien tarkoitus on saavutettu molemmissa peleissä suurimmaksi osin, vaikkakin laadun tasoissa on valtava ero. Half-Life: Alyx -pelissä on helppo ymmärtää, että ammuksia kuluu aina tarkka määrä, kun aseilla ampuu. Kurja Demossa taikojen vaatimat määrät voit tietää vain arviolta ja kokemuksen kautta. Taikojen vaatimia määriä ei kerrota pelaajalle mitenkään, mikä voi aiheuttaa hämmennystä pelaajalle. Tässä olisi jatkokehityksen tarvetta tulevaisuudessa. Mahdollista olisi muuttaa taikomisjärjestelmää yhdenmukaisemmaksi tai tehdä jokaiselle taialle indikaattori, miten paljon ne vaativat taikavoimaa. Taikasauvaan voisi myös tehdä pykäliä, joista voisi arvioida tarkemmin taikavoiman määrän ja taiat voisivat aina viedä tietyn pykälä määrän taikavoimaa.

7.4 Ohjelmat

Opinnäytetyötä tehdessäni opin paljon Blenderin ja GIMP:n käyttämisestä, mutta näissä vielä on paljon lisää opeteltavaa ja olen varma, että vaatii satoja tunteja ja monia uusia haasteita, että näistä ohjelmista osaa ottaa suurimman osan ominaisuuksista hyödyksi itseni toteuttaessa. Ymmärrän näitten ohjelmien peruskäyttöä jo hyvin. Kurja Demoon tein pelikenttiin karkeat esimerkit asioista näillä ohjelmilla, kuten esimerkiksi patsaita tai puita, joista meidän graafikkomme sitten teki lopulliset versiot.

Unity ja sen käyttämä C#-ohjelmointikieli ovat tulleet minulle jonkin verran paremmin tutuiksi, mutta niistäkin oppii vielä tehdessä lisää uusia asioita.

Kuitenkin tunnen jo osaavani käyttää niitä pelituotannossa ja ongelmienratkaisussa hyvin. Kurja Demoa tehtäessä minun piti toteuttaa siihen asioita, kuten minipelejä tai tärkeitten esineitten paikalleen palautusskripti. Näitä helposti rajattavia kokonaisuuksia tehtäessä opin paljon Unitysta ja monessa tapauksessa saamani palautteen ansiosta muokkasin niitä paremmiksi. Monet asiat olivat haastavia toteuttaa ja vaativat paljon itseopiskelua, kuten tekemäni taikaikkuna tai parannusjuoma, jonka nestepinta hölskyy pullon liikkeen mukana ja vähenee kun nestettä kaataa pois pullosta. VR-kehitystä auttamaan on Unityssa XR Interaction Toolkit -viitekehys, joka käytännössä on kokoelma skriptejä, jotka auttavat toteuttamaan asioita, kuten esineisiin tarttuminen. Kuitenkin tarttuminen kiinnittyy aina objektin vipupisteeseen tai sille voi määritellä lapsiobjektissa tarttumiskohdan. Kuitenkin halusimme, että esineistä tartuttaessa tartuntakohta voi olla missä vain kohtaa esinettä niin tätä varten piti tehdä oma skripti, joka laajensi valmista tartuntaskriptiä. Moniin perusasioihin, kuten tartuttavien esineiden käsittelyyn, heittelyyn, kantamiseen liikkuessa ja muihin esineisiin törmäilyyn piti tutustua tarkemmin ja tehdä joissain tapauksissa omia ratkaisuja, jotta ne sai toimimaan halutulla tavalla. Kurja Demoon tekemäni asiat kartuttivat osaamistani huomattavasti ja tunnen osaavani käyttää Unitya jo kehittyneellä tasolla.

7.5 VR-pelit

VR-peleissä pitää harkita paljon, miten pelaajahahmo liikkuu pelimaailmassa, koska matkapahoinvointi on yleinen ongelma, etenkin tottumattomille pelaajille ja itse huomasin, että oma sietokykyni kasvoi käyttötuntien kasvaessa. Nykyisin en koe pahoinvointia, vaikka liikkuisin VR-maailmassa nopealla vauhdilla. Iso haaste VR-peleissä on myös se, mitä pelaaja milloinkin katsoo ja osaako pelaaja tulkita asiat oikein. Pelin tekijä saattaa käyttää paljon aikaa joihinkin yksityiskohtiin vain huomatakseen, että suurin osa testaajista ei sitä huomaa ollenkaan. Esimerkiksi toteuttamani ison patsaan räjäytys palasiksi jäi joiltakin kokonaan huomaamatta, koska heidän huomionsa oli muualla, joten päädyimme lisäämään vihjaavaa ääntä, joka toivottavasti saa pelaajat kiinnittämään huomiota tapahtumaan paremmin. Tällaisissa asioissa tarvitsen lisää kokemusta, jotta ymmärrän paremmin, mikä on tärkeää pelejä tehdessä. Omasta mielestäni molemmissa peleissä diegeettinen käyttöliittymä lisäsi pelin immersiota huomattavasti, tehden pelattavuudesta nautittavampaa. Tästä

syystä aion jatkossakin suosia diegeettisiä käyttöliittymiä peleissä. Kuitenkin pelaajille pitäisi opettaa käyttöliittymän ominaisuudet pelin edetessä ja tässä tulee ottaa mukaan tarinan kerronta ja kenttäsuunnittelu, jotta käyttöliittymä on ymmärrettävissä muillekin kuin vain sen tekijöille.

7.6 Partikkeliefektien käyttäminen osana käyttöliittymää

Minulle tuli selväksi Kurja Demoa tehdessä, että partikkeleita voi käyttää tehokkaasti viestimään pelaajalle tietoa osana käyttöliittymää. Minusta tuntuukin, että ratkaisuni elin- ja taikavoiman viestimiseen palvelevat tarkoitustaan hyvin. Ne yksinkertaisesti viestivät pelaajalle, missä mennään asteikolla 0–100, mikä tässä tapauksessa on se mitä tarvittiinkin. Täytyy pitää mielessä, että partikkeliefektit ovat tässä tapauksessa yhdistettyinä kolmiulotteisiin objekteihin käyttöliittymässä. Muilla objekteilla onkin kätevää toteuttaa partikkeliefektille rajaus ja yhteys pelimaailmaan. Yksinään ilmassa leijuva partikkeliefekti pitäisi suunnitella todella hyvin, jotta sillä voisi antaa täsmällistä tietoa pelaajalle ja varmaankin vaatisi paljon testausta onko se edes helposti ymmärrettävissä. Kuitenkin partikkeliefektien parametrien tarkalla säätelyllä ne saadaan tuottamaan tarkkoja muotoja, joita voidaan käyttää käyttöliittymän osina. En näkisi mitään estettä sille, etteikö partikkeleilla voisi toteuttaa mitä vain mittaria, minkä arvoa pitäisi pelaajalle viestiä osana käyttöliittymää, mutta kannattaa pitää mielessä, että se ei ole välttämättä ole ainoa tai paras keino tehdä käyttöliittymän osia. Tämän opinnäytetyön tekeminen opetti minulle, että partikkeliefektit ovat toimiva työkalu käyttöliittymiä toteutettaessa.

7.7 Työn tekemisen aikana opittuja asioita

Peliin toteutetut asiat kannattaa testauttaa muilla ja käyttää siitä saatu palaute hyödyksi. Palautetta ei tule ottaa henkilökohtaisesti, vaan tulee yrittää selvittää, miksi testaaja koki asian tietyllä tavalla, jotta siitä voi oppia.

Pelin käyttöliittymästä tulisi tehdä ymmärrettävä ilman jatkuvaa päättelyä, jotta pelaaja osaa käyttää sitä helposti. Kuitenkin monesti asioiden selittäminen pelaajalle voi olla tarpeen, vaikka immersio kärsii hieman siitä.

Diegeettinen käyttöliittymä on parempi immersion kannalta, mutta pelaajat hyväksyvät helposti neljännen seinän rikkovaa tietoa, jos siitä on heille hyötyä. Esimerkiksi opaste kertoo pelaajalle, miten ase ladataan pelissä ja opaste häviää sen jälkeen, kun pelaaja lataa aseen, minkä jälkeen voidaan olettaa, että pelaaja osaa ladata aseen jatkossakin.

Pelaajan ei välttämättä ole tarpeellista tietää tarkkaa arvoa jostakin asiasta, kuten elinvoimasta, vaan arvosta voi riittää summittainen viesti, jonka saa selville nopealla vilkaisulla. Lisäksi tulisi miettiä, sopiiko käyttöliittymän osa pelin maailmaan. Esimerkiksi, jos tavoillaan diegeettistä käyttöliittymää fantasiapeliin, numeronäyttö pelaajan elinvoimasta ei välttämättä sovellu järkevästi siihen ympäristöön.

Partikkeliefektit toimivat tiedon välittämiseen pelaajalle osana käyttöliittymää. Silti kannattaa miettiä, ovatko ne tilanteeseen sopivia vai onko asia ratkaistavissa muulla tavalla, joka voisi olla parempi tapa toteuttaa se. Partikkeliefektit ovat vain yksi tapa muiden joukossa.

Lähteet

Andrews, M. 2010. Game UI Discoveries: What Players Want. Gamasutra https://www.gamasutra.com/view/feature/4286/game_ui_discoveries_what_players_.php 17.1.2022.

Blender. 2022. Blender Foundation. https://blender.org/about/ . 18.2.2022.

Ermi, L. & Mäyrä, F. 2005. Fundamental components of the gameplay experience: Analyzing immersion. http://www.digra.org/digital-library/publications/fundamental-components-of-the-gameplay-experience-analysing-immersion/ . 9.2.2022.

FBX. 2022. Blender Foundation. FBX – Blender Manual. https://docs.blender.org/manual/en/2.80/addons/io_scene_fbx.html 10.1.2022.

GIMP. 2022. https://www.gimp.org/docs/userfaq.html . 6.4.2022.

GNU. 2007. https://www.gnu.org/licenses/gpl-3.0.en.html . 6.4.2022.

Half-Life: Alyx. 2020. Valve.

Iacovides, I. Cox, A. Kennedy, R. Cairns, P. Cairns, J. & Jennet, C. 2015. Removing the HUD: The Impact of Non-Diegetic Game Elements and Expertise on Player Involvement. https://oro.open.ac.uk/46759/1/Diegesis%20and%20immersion%20-%20final.pdf . 19.1.2022.

Kurja Demo. 2021. Dreal. https://www.dreal.fi/kurja/

Lander, J. 1998. The Ocean Spray in Your Face. https://www.lri.fr/~mbl/ENS/IG2/devoir2/files/docs/particles.pdf . 2.3.2022

Marxent. 2020. Everything You Need to Know About Using FBX Files https://www.marxentlabs.com/fbx-files/ . 17.4.2022.

Metacritic. 2022. Half-Life: Alyx. https://www.metacritic.com/game/pc/half-life-alyx . 2022.

Microsoft. 2022. Find code changes and other history with CodeLens. https://docs.microsoft.com/en-us/visualstudio/ide/find-code-changes-and-other-history-with-codelens?view=vs-2019 . 6.4.2022.

Russel, D. 2011. Video game user interface design: Diegesis theory. Dev.Mag http://devmag.org.za/2011/02/02/video-game-user-interface-design-diegesis-theory/ 2.2.2011.

Unity. 2022a. Unity technologies. https://unity.com/ . 24.1.2022.

Unity. 2022b. Plans and pricing. https://store.unity.com/#plans-business . 24.1.2022.

Unity. 2022c. Coding in C# in Unity for beginners. https://unity.com/how-to/learning-c-sharp-unity-beginners . 24.1.2022.

Unity. 2022d. Writing Surface Shaders. https://docs.unity3d.com/Manual/SL-SurfaceShaders.html . 9.2.2022.