jamk.fi

Visuaalisten efektien käyttö 2D-pelikehityksessä

Sampo Riikkilä

Opinnäytetyö
Toukokuu 2018
Liiketalouden ala
Tietojenkäsittelyn tutkinto-ohjelma

Jyväskylän ammattikorkeakoulu
JAMK University of Applied Sciences

jamk.fi

Kuvailulehti

Tekijä(t)	Julkaisun laji	Päivämäärä
Riikkilä, Sampo	Opinnäytetyö, AMK	Toukokuu 2018
	Sivumäärä 56	Julkaisun kieli Suomi
		Verkkojulkaisulupa myönnetty: x

Työn nimi
Visuaalisten efektien käyttö 2D-pelikehityksessä

Tutkinto-ohjelma
Tietojenkäsittelyn koulutusohjelma

Työn ohjaaja(t)
Karhulahti, Mika

Toimeksiantaja(t)
Tietojenkäsittelyn tutkinto-ohjelma, Jyväskylän ammattikorkeakoulu

Tiivistelmä

Nykypäivänä pelinkehittäjät joutuvat kilpailemaan tuotteidensa näkyvyydestä lukuisten samoilla jakelukanavilla myytävien pelien kanssa ja massasta erottuminen voi olla hankalaa. Pelin näyttävä visuaalinen ilme on yksi merkittävimmistä tavoista erottua muista kilpailijoista ja sitä voidaan tehostaa huomattavasti erilaisten visuaalisten efektien avulla. Pelattavuuteen ja tunnelmaan vaikuttavia efektejä voidaan toteuttaa monin eri tarvoin ja modernit 3D-pelimoottorit mahdollistavat entistä joustavamman tehostetyöskentelyn myös 2D-pelien kehityksessä.

Tutkimuksen tarkoituksena oli saada selville mitä visuaaliset efektit ovat, minkälaisia visuaalisia efektejä moderneissa 2D-peleissä käytetään ja kuinka niitä voidaan toteuttaa hyviä käytänteitä noudattaen Unity Enginellä toteutettavissa 2D-peleissä. Tutkimus toteutettiin kehittämistutkimuksena, jossa käytettiin kvalitatiivisia tutkimusmenetelmiä ja sen sivutuotoksena oli tarkoitus tehdä ohjeistus visuaalisten efektien toteuttamisesta. Ohjeistuksella pyrittiin kehittämään JAMK:in tietojenkäsittelyn koulutusohjelman toimintaa luomalla uudenlaista oppimateriaalia, jota JAMK:in tietojenkäsittelyn opiskelijat voisivat hyödyntää pelialan opinnoissaan.

Opinnäytetyön tulosten perusteella saatiin selville, että erilaiset partikkeliefektit, shadereilla toteutetut efektit sekä animaatioilla tehdyt efektit ovat yleisimpiä tapoja toteuttaa visuaalisia efektejä moderneissa 2D-peleissä. Lisäksi saatiin selville, kuinka näitä efektejä toteutetaan Unity Enginessä ja miten niiden optimointi tapahtuu.

Jyväskylän ammattikorkeakoulun tietojenkäsittelyn koulutusohjelmalle toteutettiin tutkimuksen sivutuotoksena käytännönläheinen ohjeistus visuaalisten efektien toteuttamisesta 2D-peleissä Unity Enginen avulla.

Avainsanat (asiasanat)

Visuaaliset efektit, peliala, pelinkehitys

Muut tiedot

jamk.fi

Description

Author(s)	Type of publication	Date
Riikkilä, Sampo	Bachelor's thesis	May 2018
		Language of publication: Finnish
	Number of pages: 56	Permission for web publication: x

Title of publication
Use of visual effects in 2D game development

Degree programme

Supervisor(s)
Karhulahti, Mika

Assigned by
Business Information Technology degree programme, JAMK University of Applied sciences

Abstract

Today, game developers are forced to compete for the visibility of their products against many other games that are being sold on the same distributing platforms; hence, standing out from the masses can be difficult. The visual appearance of a game is one of the most significant ways to stand out from competitors and it can be enhanced notably by using different visual effects. Visual effects affecting the gameplay and the mood of games can be implemented in many ways, and modern 3D game engines allow a flexible visual effects workflow also in 2D game development.

The purpose of the research was to find out what visual effects are, what kind of visual effects are used in modern 2D games and how can they can be implemented by using good practice in 2D games made in Unity Engine. The research was carried out as a development research, which used qualitative research methods and aimed to create instructions about implementing visual effects as a part of the thesis. The goal of creating the instructions was to improve the degree programme in Business Information Technology at JAMK University of Applied Sciences by creating new learning material for its students.

The results of the research led in the discovery that different kind of particle effects, shader-based effects and frame-by-frame animated effects are the most commonly used ways to implement visual effects in modern 2D games. Additionally, it was discovered how these effects can be implemented using Unity Engine and how they should be optimized.

Practical instructions about implementation of visual effects in 2D games using Unity Engine were created as a part of the thesis.

Keywords/tags (subjects)
Visual effects, game industry, game development

Miscellaneous

Sisältö

Kuviot

Käsitteet

Assetti	Assetit ovat erilaisia pelikehityksessä käytettäviä asioita, joita voidaan näyttää pelin käyttäjälle, kuten esimerkiksi grafiikka tai äänet.
Frame	Tietokoneen ruutu päivitetään useita kymmeniä kertoja sekunnissa. Yhtä tällaista päivitystä ja sen mukana ruudulle tuomaa kuvaa kutsutaan frameksi.
Mesh	Mesh tarkoittaa 3D-mallin rakennetta, joka koostuu vertekseistä muodostuvista polygoneista. 3D-meshit käyttävät X-, Y, ja Z-akseleita määrittääkseen muotoja jotka sisältävät pituuden, leveyden ja syvyyden.
Partikkeli	Pelinkehityksessä käytettävät partikkelit ovat pieniä kuvia tai meshejä, joita hallitaan partikkelijärjestelmien avulla. Ne muodostavat usein isompia kokonaisuuksia, kuten savu- tai tuliefektejä.
Pelimoottori	Pelimoottorit ovat viitekehyksiä, joiden avulla voidaan kehittää pelejä, sovelluksia tai animaatioita. Ne tarjoavat kehittäjille paljon erilaisia työtaakkaa helpottavia työkaluja.
Pikseli	Pikseli on kuvan pienin yksikkö digitaalisella näytöllä, kuten televisiolla tai tietokoneen näytöllä.
Renderöinti	Renderöinti tarkoittaa prosessia, jonka tuloksena kaksi- tai kolmiulotteinen kuva piirtyy tietokoneen näytölle.

Sprite	Sprite on kaksiulotteinen kuva, jota voidaan manipuloida tarpeen mukaan ja jonka avulla voidaan luoda suurempia kokonaisuuksia.
Sprite sheet	Sprite sheet tarkoittaa useita samaan kuvatiedostoon yhdistettyjä spritejä.
Tekstuuri	3D-grafiikassa tekstuurit ovat kuvatiedostoja, joiden avulla objekteille voidaan antaa kaksi- tai kolmiulotteisia ominaisuuksia, kuten väriä tai pinnanmuotoja.
Verteksi	Verteksi tarkoittaa virtuaalisessa tilassa sijaitsevaa pistettä, joka sisältää erilaisia attribuutteja, kuten sijainnin, värin tai tekstuurikoordinaatit.

1 Johdanto

Nykypäivän saturoituneilla videopelimarkkinoilla kilpailu on kovaa, joten pelinkehittäjät joutuvat jatkuvasti sopeutumaan kysyntään ja parantamaan tuotteidensa laatua uusien teknologioiden sekä työkalujen avulla. Hyvä tapa erottua mahdollisesti monien tuhansien samaan aikaan ja samoissa jakelukanavissa kilpailevien pelien joukosta on näyttävän visuaalisen ilmeen toteutus, jota voidaan tehostaa huomattavasti teknisillä ratkaisuilla tai efekteillä, kuten valaistuksella tai partikkeliefekteillä. Isompien peliyrityksien ja-projektien tekijöiden joukossa on tästä syystä hyvin usein teknisiä graafikkoja, efektiartisteja tai muun nimisiä henkilöitä, jotka keskittyvät pelien visuaalisen ilmeen parantamiseen pelimoottoria käyttäen. Osaaville tekijöille on selvästi tarvetta efektipuolella, sillä opinnäytetyön kirjoittamishetkellä suomalaisista peliyrityksistä mm. Nitro Games, Critical Force ja Housemarque etsivät tiimeihinsä sekä teknisiä graafikkoja että efektiartisteja. (Careers n. d.a; Careers n. d.b; Jobs in Critical Force n.d.)

Monelle aloittavalle pelinkehittäjälle foorumit, blogit, videosivustot sekä muut kanavat on hyvä tapa etsiä tietoa aiheesta, mutta yleisimpien pelimoottorien massiivisen suosion takia esimerkiksi ilmaiseksi saatavilla olevaan Unity-pelimoottoriin viittaavia ohjevideoita ilmestyy videopalvelu Youtubeen jatkuvasti käyttäjiltä ympäri maailmaa. Saatavilla olevaa tietoa on paljon, mutta sen laatu on hyvin vaihtelevaa, eivätkä monet materiaalien tuottajista ota välttämättä huomioon pelien optimoimiseen liittyviä seikkoja. Visuaalisten efektien liiallinen tai vääränlainen käyttö voi olla hyvin raskasta tietokoneen suorittimelle ja ne saattavat aiheuttaa ongelmia pelin kehityksessä ja pahimmassa tapauksessa huonontaa asiakkaan pelikokemusta. Tästä syystä opinnäytetyön toimeksiantajalle päätettiin tehdä koostettu ohjeistus visuaalisista efekteistä ja niiden toteuttamiseen liittyvistä hyvistä käytänteistä.

Tutkimuksen teoriaosuudessa selvitetään visuaalisten efektien tarjoamia mahdollisuuksia, tavallisimpien videopeleissä käytettävien efektien taustateoriaa ja käyttöä 2D-peleissä sekä tutkitaan, kuinka niitä voidaan hyödyntää hyviä käytänteitä noudattaen Unity-pelinkehitysympäristössä. Tutkimuksen pohjalta luotiin käytännönläheinen ja aloittelijaystävällinen ohjeistus visuaalisista efekteistä, jota

Jyväskylän ammattikorkeakoulun tietojenkäsittelyn koulutusohjelman opiskelijat voivat käyttää hyödyksi pelialan opinnoissaan.

2 Tutkimusasetelma

Tässä luvussa esitellään opinnäytetyön toimeksiantaja ja sen tausta, sekä käsitellään opinnäytetyössä käytetyt tutkimusmenetelmät ja tutkimuskysymykset.

2.1 Tavoitteet ja rajaukset

Jyväskylän ammattikorkeakoulu toimi opinnäytetyön toimeksiantajana. Jyväskylän ammattikorkeakoulusta valmistuu vuosittain yli 1500 opiskelijaa ja se tarjoaa yli 30 tutkintoa 8 eri alan tehtäviin. (Tutustu JAMKiin: strategia ja vahvuusalat n. d.) JAMKin tietojenkäsittelyn tutkimusohjelmassa opiskelijat kouluttautuvat moniin eri ICT-alan tehtäviin ja voivat halutessaan suuntautua pelialalle kurssivalintojensa sekä projektiopintojen kautta. (Bisneksen ja tietotekniikan osaaja n.d.)

Työn tekijän kiinnostus teknisen graafikon työtehtäviin ja visuaalisten efektien toteuttamiseen synnytti idean opinnäytetyön aiheesta, jossa käsiteltiin visuaalisten efektien toteuttamista pelikehityksessä. Lisäksi työn tekijä on opiskellut tietojenkäsittelyn koulutusohjelmassa vuoden 2014 syksystä lähtien ja opetellut visuaalisten efektien taustateoriaa ja toteuttamista omatoimisesti, sillä varsinaisten kurssiopintojen aikana niitä ei käsitelty. Näiden tekijöiden pohjalta opinnäytetyötä ohjaavan opettajan kanssa huomattiin tarve visuaalisin efekteihin liittyvälle ohjeistukselle, jota Jyväskylän ammattikorkeakoulun tietojenkäsittelyn koulutusohjelman opiskelijat voisivat hyödyntää opinnoissaan. Tavoitteena oli toteuttaa opiskelijoille käytännönläheinen ja aloittelijaystävällinen ohjeistus, josta löytyy koostettua tietoa yleisimmistä peleissä käytettävistä visuaalisista efekteistä ja niiden toteuttamisesta selkeässä muodossa.

Aloittajaystävällisyyden ja selkeyden vuoksi ohjeistus haluttiin kerätä 2D-efektien ympärille, sillä ne ovat helpommin ymmärrettävissä ja sovellettavissa myöhemmin myös 3D-pelejä varten. Esimerkkien alustaksi haluttiin pelimoottori Unity, sillä se on yksi suosituimmista pelinkehitysympäristöistä ja sitä käytetään pelien kehittämiseen myös JAMKin tietojenkäsittelyn koulutusohjelmassa. Lisäksi Unityn 3D-

ominaisuuksien avulla voidaan toteuttaa entistä näyttävämpiä visuaalisia efektejä myös 2D-peleissä. Visuaalisten efektien luonteen takia niitä on hankala havainnollistaa ilman liikkuvaa kuvaa, joten toteutettavasta tuotoksesta haluttiin monimediapaketti, joka sisältää tekstiä, kuvia ja videoita.

2.2 Tutkimusmenetelmä ja-kysymykset

Opinnäytetyön tutkimusmenetelmänä käytettiin kehittämistutkimusta. Kehittämistutkimus on monimenetelmäinen tutkimusote, mikä tarkoittaa, että se on joukko eri tutkimusmenetelmiä, joita koostetaan tutkimuksen tarpeiden mukaisesti. Siinä yhdistyvät sekä kvantitatiiviset että kvalitatiiviset tutkimusmenetelmät. (Kananen 2015, 33) Kehittämistutkimuksen tarkoituksena on saada aikaan muutos tai ongelman poistaminen. Tutkimuksellinen ote tai tutkimusosio erottaa sen tavallisesta kehittämistyöstä. (Kananen 2015, 40) Kehittämistutkimus koostuu sykleistä, joiden vaiheisiin kuuluvat suunnittelu, toiminta, havainnointi sekä seuranta. Kehittämissyklin alussa tutustutaan ilmiöön, jolloin kartoitetaan nykytila ja määritellään kehittämiskohde eli ongelma, jonka jälkeen arvioidaan keinot ongelman poistamista varten. Muutos toteutetaan ja keinojen vaikutuksia arvioidaan kriittisesti. (Kananen 2015, 41-42; Kananen 2012, 21-23) Kehittämistutkimukselle tyypillinen syklisyys toteutetaan opinnäytetyössä yhden syklin aikana. Opinnäytetyön tutkimusmenetelmää valitessa kehittämistutkimus oli melko selkeä valinta, sillä tutkimuksessa halutaan kehittää JAMKin tietojenkäsittelyn koulutusohjelman pelialan opintoja uudenlaisen oppimateriaalin avulla.

2.2.1 Mitä visuaaliset efektit ovat?

Tällä kysymyksellä selvitetään visuaalisten efektien määritelmää ja niiden merkitystä videopeleissä. Lisäksi erotellaan videopeleissä käytettävien efektien lajityyppejä ja perustellaan niiden merkitystä.

2.2.2 Minkälaisia visuaalisia efektejä moderneissa 2D-videopeleissä käytetään?

Tämän kysymyksen pohjalta tutkitaan tunnettuja moderneja 2D-pelejä ja selvitetään minkälaisia visuaalisia efektejä ne hyödyntävät. Lisäksi tutkitaan pelimoottorien

tarjoamia työkaluja 2D-pelikehitystä varten ja minkälaisia efektejä niillä on mahdollista toteuttaa.

2.2.3 Kuinka visuaalisia efektejä voidaan toteuttaa optimoidusti 2D peleihin Unity-pelinkehitysympäristössä?

Tätä kysymystä varten selvitetään yleisiä pelinkehityksessä käytettäviä efektitekniikoita, selvitetään niiden taustateoriaa ja tutkitaan, kuinka niitä voidaan toteuttaa hyviä käytänteitä noudattaen Unity-pelinkehitysympäristössä 2D-peleihin. Tämän kysymyksen pohjalta luodaan tutkimuksen lisäksi toteutettava ohjeistus.

3 Visuaaliset efektit pelikehityksessä

Tässä luvussa selvennetään visuaalisten efektien määritelmää ja niiden merkitystä pelikehityksessä. Lisäksi tutkitaan tekniikoita sekä työkaluja, joiden avulla visuaalisia efektejä voidaan toteuttaa ja kerrotaan, kuinka niitä voidaan hyödyntää 2D-pelikehityksessä.

3.1 Visuaalisten efektien määritelmä

Taiwanilainen, vuodesta 2008 asti elokuva-alalla työskennellyt visuaalisten efektien tuottaja, LightRay Visual Effects viitaa blogikirjoituksessaan alansa standardikirjallisuuteen ja täsmentää visuaalisten efektien määritelmää elokuvateollisuuden näkökulmasta. Visuaalinen efekti on termi jotka käytetään kuvailemaan visuaalista materiaalia, joka on tehty, muokattu tai parannettu elokuvaa tai muuta liikkuvaa mediaa varten ja jota ei voi toteuttaa varsinaisen kuvaamisen aikana. Nykyaikaisten digitaalisten työkalujen ansiosta visuaalisten efektien toteuttaminen kuuluu lähes jokaisen liikkuvan kuvan tekijän työnkuvaan, sillä ne mahdollistavat virheettömän sommittelun, digitaalisten lavastusten tai tietokoneella generoitujen hahmojen toteuttamisen. (What is visual effects? 2012.)

Vuodesta 2005 lähtien pelialalla työskennellyt VFX artisti Francisco García-Obledo Ordóñez kertoo haastattelussaan, minkälaisia visuaalisia efektejä VFX artistit pääasiassa toteuttavat peliprojekteissa. Ordóñezin mukaan peleissä käytetyt efektit voidaan jakaa pelattavuuteen vaikuttaviin ja ympäristöllisiin efekteihin, joiden

painoarvo vaihtelee projektikohtaisesti. Pelattavuuteen vaikuttavien efektien toteuttaminen vaatii syvää ymmärrystä pelimekaniikoista, sekä jatkuvaa yhteistyötä niiden suunnittelijoiden kanssa. Hyvänä esimerkkinä pelimekaniikkojen suunnittelijat voisivat kehittää peliin liekinheittimen, josta lähtevien liekkien paikalle asetetaan testiobjekti havainnollistamaan aluetta, jolle liekit leviävät. VFX artistin tehtävänä olisi luoda tälle alueelle liekkejä esittävä visuaalinen efekti, joka seuraa suunnittelijoiden asettamia vaatimuksia. Ympäristölliset efektit ovat luonteeltaan erilaisia, sillä usein niiden merkitys ei ole pelattavuuden kannalta merkittävä. Tällaisia ovat esimerkiksi vesiputousta, sumua, sadetta tai pölyä esittävät efektit. (Tokarev 2017)

Videopeleihin tehtävillä visuaalisilla efekteillä toteutetaan samankaltaista visuaalista materiaalia tai sen parantelua kuin muuhunkin liikkuvaan kuvaan. Erona on se, että efektit toteutetaan jo valmiiksi digitaalisen materiaalin yhteyteen.

3.2 Shaderit

3.2.1 Mitä shaderit ovat?

Videopelit ja muut graafiset ohjelmat vaativat paljon enemmän prosessointivoimaa kuin muut ohjelmat, sillä niiden graafisen sisällön takia ne joutuvat tekemään todella suuria määriä pikseli kerrallaan tapahtuvia toimintoja. Jokainen ruudulla näkyvä pikseli joudutaan käsittelemään ja 3D-peleissä myös geometriat sekä perspektiivit joudutaan laskemaan. Tietokoneen suoritin prosessoi näitä laskutoimituksia yksi kerrallaan jokaista säiettään kohden, joten pikseli kerrallaan suoritettavat operaatiot ovat todella raskaita suurissa määrin. Graafisen suorittimen avulla tämä ongelma on kierrettävissä, sillä ne koostuvat isosta määrästä rinnakkain toimivista mikroprosessoreista. Näin iso määrä pienempiä laskutoimituksia pääsee kerralla läpi prosessointivaiheesta. (Vivo & Lowe 2015.) Shaderit ovat yksinkertaisuudessaan pieniä ohjelmia, jotka toimivat suoraan tietokoneen graafisessa suorittimessa, jättäen varsinaiselle suorittimelle vähemmän prosessoitavaa. Niiden avulla voidaan toteuttaa esimerkiksi valaistukseen, värikorjaukseen tai 3D-mallien muodonmuokkaukseen liittyviä efektejä. (Game Graphics 101: Rendering 2D on GPU 2016)

3.2.2 Shaderien historia

Vuosien 2001 ja 2004 välillä graafisessa suorittimessa tapahtuvan ohjelmoitavuuden toteuttaminen tehtiin mahdolliseksi. Microsoftin vuonna 2001 julkaisema Direct3D 8.0 tuki uutta ominaisuutta jota kutsuttiin shadereiksi. Shaderit mahdollistivat paremman ohjelmoitavuuden ja toimivuuden eliminoimalla paljon tehtäviä suorittimelta, mutta ne olivat erittäin hankalia ohjelmoida niiden syntaksin takia, joka muistutti suoritinta varten tehtyä assembly-ohjelmointikieltä. Tämän lisäksi ohjelmoijat joutuivat ohjelmoimaan vähintään yhden shader-version jokaista laitteistomyyjää kohti, sillä ne eivät jakaneet samaa ohjelmointikieltä. (Luten n.d; Rodriguez 2013, 24.)

Direct3D 9.0-version julkaisu vuonna 2003 toi mukanaan merkittäviä muutoksia shadereiden käytettävyyteen, sillä Microsoft julkaisi C-ohjelmointikieleen perustuvan High-Level Shader Language-kielen, joka helpotti shaderien käyttöä. (Luten n.d) Muitakin vastaavia ohjelmointikieliä luotiin ohjelmoijien saataville, kuten NVidian kehittämä Cg. Nämä ratkaisut eivät kuitenkaan toimineet kaikilla alustoilla, sillä HLSL oli osa Direct3D:tä ja Cg:tä voitiin käyttää vain NVidian näytönohjaimien kanssa. (Rodriguez 2013, 24) Vuoden 2004 aikana jotkin yritykset tajusivat tarpeen monilla eri alustoilla toimivaan shader-ohjelmointikieleen. Siirtyessään versioon 2.0, OpenGL alkoi tukemaan shadereita ja julkaisi oman shader-ohjelmointikielensä OpenGL Shading Language. (Luten n.d; Rodriguez 2013, 24)

Nykyaikana shadereiden merkitys pelikehityksessä on erittäin merkittävä ja niitä käytetään laajalti niin isoissa kuin pienissäkin peliprojekteissa. Eri pelinkehitysympäristöt käyttävät vaihtelevia Shader-ohjelmointikieliä, mutta niiden perustason toimivuudet ovat hyvin lähellä toisiaan.

3.2.3 Shader-tyypit

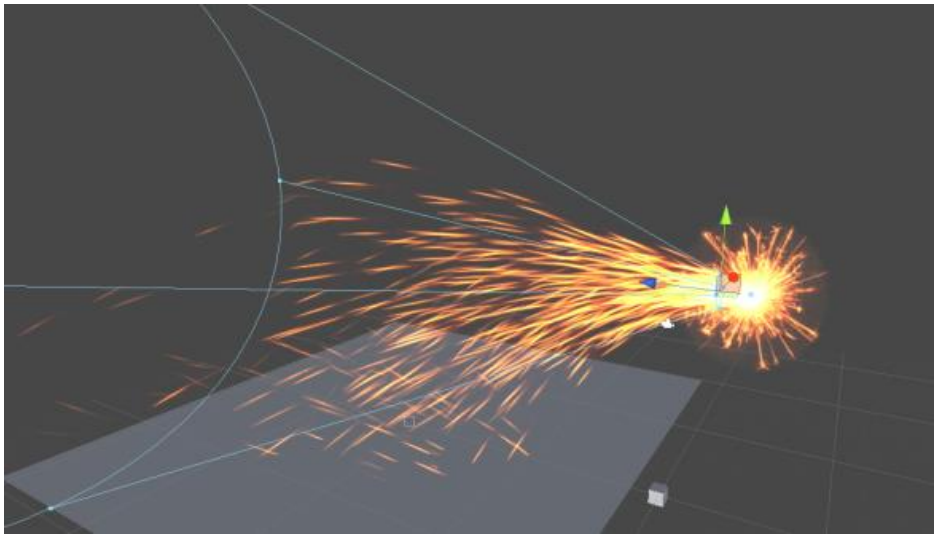
Fragment- eli Pixel shaderit ovat vastuussa jokaisen pikselin värin palauttamisesta, ja se prosessoidaan kerran jokaista näytöllä näkyvää pikseliä tai valitulla alueella sijaitsevia pikseleitä kohden. Yksinkertaisimmillaan fragment shaderit palauttavat pelkän alkuperäisen värin, mutta niiden avulla on mahdollista manipuloida värejä esimerkiksi gammakorjauksen muodossa. (Rodriguez 2013, 29; Game Graphics 101:

Rendering 2D on GPU 2016) Jacobo Rodriguez (Rodriguez 2013, 29) painottaa kirjassaan fragment shaderien optimoimisen tärkeyttä, sillä esimerkiksi liikkeen sumennukseen ja hohtoon vaikuttavat shaderit tai muut koko ruudulle renderöitävät efektit joudutaan rasteroimaan. Tämä tarkoittaa sitä, että kaikki ruudulla näkyvät pikselit joudutaan laskemaan ja todellisuudessa fragment shadereita prosessoidaan miljoonia kertoja sekunnissa. Sama pätee myös pienempiin alueisiin vaikuttaviin shadereihin, sillä laskutoimitusten määrä nousee mitä enemmän shadereita on päällekäin. (Rodriguez 2013, 29)

Vertex shadereilla voidaan vaikuttaa jokaisen prosessoitavan verteksin ominaisuuksiin ennen kuin se renderöidään näytölle. Manipuloitavia ominaisuuksia ovat esimerkiksi verteksin sijainti tai tekstuurikoordinaatit. (Sherrod & Jones 2011, 288) Vertex shaderit ovat merkittäviä myös 2D-pelikehityksessä, sillä 3D-moottorissa näytölle renderöitävät 2D-spritet muodostuvat vertekseistä. Pelissä käytettävät spritet rakentuvat usein neljästä verteksistä, jotka muodostavat suorakulmion. Näitä verteksejä muokkaamalla voidaan saada aikaan erilaisia efektejä, kuten mittojen muuttamista tai aaltoilevan liikkeen simuloimista. (Game Graphics 101: Rendering 2D on GPU 2016) Lisäksi vertex shadereilla on mahdollista vaikuttaa myös verteksien väliseen alueeseen esimerkiksi antamalla jokaiselle verteksille oma väriarvonsa ja interpoloimalla niiden väliin jäävä tila, jolloin verteksien väliin renderöityy valittua väriä.

3.3 Partikkeliefektit

Suurin osa nykyaikana käytettävistä pelimoottoreista sisältää valmiit työkalut partikkeliefektien toteuttamiseen, joita kutsutaan usein partikkelijärjestelmiksi. Partikkelijärjestelmien avulla pienistä, simppeleistä kuvista tai polygoniverkoista voidaan muodostaa komplekseja efektejä, kuten luoda vaikutelma virtaavasta nesteestä tai kaasusta (ks. Kuvio 1). Esimerkiksi savupilveä tehtäessä jokaisella partikkelilla olisi pieni savutekstuuri, joka edustaa itsessään pientä pilveä. Kun näitä pieniä pilviä yhdistetään paljon samalle alueelle, niistä syntyy kokonaisen ja isomman pilven vaikutelma. (Unity Documentation, 2017a.)



Kuvio 1 Esimerkki kipinöitä matkivasta partikkeliefektistä (Heazlewood 2013)

3.4 Efektianimaatiot

Tietokoneella generoitujen visuaalisten efektien lisäksi videopeleissä voidaan käyttää myös käsin piirrettyjä efektianimaatioita. Tällaisten efektien toteuttaminen vaatii erilaista osaamista tietokoneella generoituihin efekteihin verrattuna ja ne saattavat olla työläämpiä toteuttaa, mutta tietynlaista graafista ilmettä hakiessa ne voivat olla välttämättömiä.

Lukuisten Disneyn tuottamien animaatioelokuvien tuotannossa työskennellut Joseph Gilland on yksi efektianimaatioiden tekijöiden uranuurtajista. Gillandin LinkedIn-profiilisivun mukaan (Joseph Gilland. n.d) hänellä on yli 32-vuoden kokemus animaatiosta ja hän on työskennellyt mm. Disneyn Hercules, Lilo & Stitch ja Pocahontas-elokuvissa visuaalisten efektien animoijana. Gillandin (2009) mukaan digitaalisten design- ja animaatiotyökalujen runsauden vuoksi nykypäivänä kirjoitetaan lukemattomia kirjoja niiden käyttöliittymän opettelua varten, eikä niinkään keskitytä ilmiöihin, joita yritetään luoda uudelleen. Usein visuaalisia efektejä ohjelmoivat henkilöt joutuvat tutkimaan näitä luonnollisia ilmiöitä perusteellisesti, jotta he pystyvät simuloimaan niistä tarkkoja jäljennöksiä. Gilland (2009) pitää erityisen tärkeänä animoitavan ilmiön luonnollisen ilmentymän pitkäjänteistä tarkkailua, jotta sen fyysiikkaan liittyviä ominaisuuksia voidaan ymmärtää.

Efektianimaatioita voidaan käyttää videopeleissä sellaisenaan, tai niitä voidaan hyödyntää vaikka partikkeliefektin sisällä. Hyvänä esimerkkinä jokaiselle partikkelille

voitaisiin asettaa käsin piirretty räjähdysanimaatio ja näiden yksittäisten animaatioiden määrää sekä käyttäytymistä pystyttäisiin hallinnoimaan partikkelijärjestelmästä.

3.5 3D-pelimoottorien hyödyntäminen 2D-pelikehityksessä

Moderneissa 2D-peleissä voidaan hyödyntää normaalisti 3D-pelikehityksessä käytettäviä efektitekniikoita, sillä 3D-pelimoottorit kuten Unity- tai Unreal Engine mahdollistavat myös 2D-pelien kehittämisen. Esimerkiksi Unity Enginellä on kattava valikoima erilaisia 2D-pelikehitykseen soveltuvia työkaluja ja oma työtila 2D-työskentelyä varten. (Unity for 2D n.d) 3D-renderöinti mahdollistaa shaderien avulla esimerkiksi valaistuksen ja erilaisten jälkikäsittelyefkektien käytön, joka olisi puhtaasti 2D-pelejä varten tehdyissä pelimoottoreissa hankalaa. (Game Graphics 101: Rendering 2D on GPU 2016)

4 Visuaaliset efektit 2D peleissä

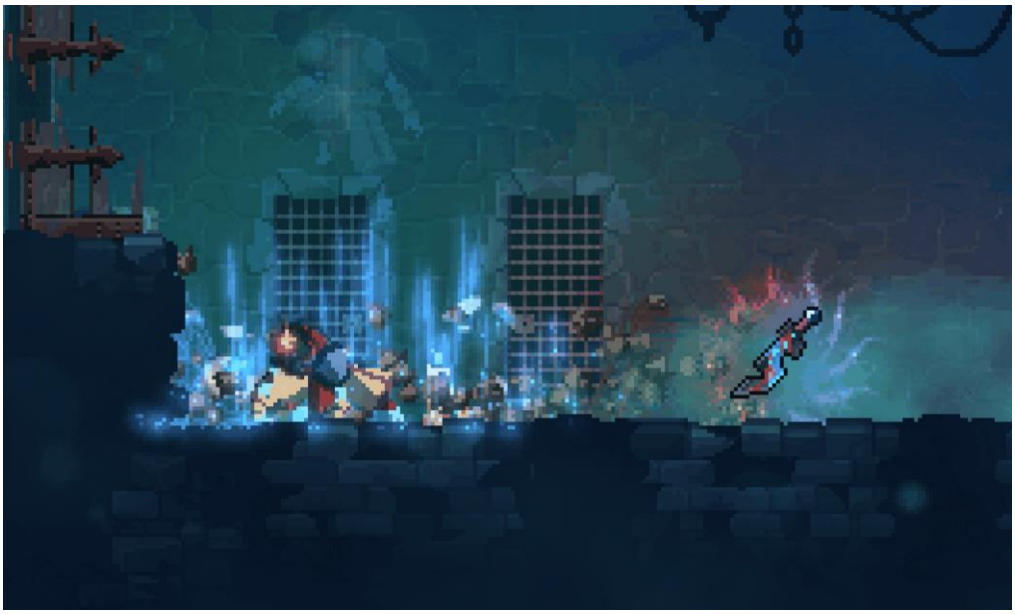
Tässä luvussa tutustutaan kolmeen moderniin ja eri tyyppiseen 2D-peliin, sekä analysoidaan niiden visuaalisten efektien käyttöä ja toteutusta. Analysoitavat pelit valittiin niiden käyttämien visuaalisten efektien erilaisuuden, julkaisuvuoden ja SteamDB-sivuston arvioitujen Steam-käyttäjämäärien perusteella. SteamDB on erilaisten ohjelmointirajapintojen kautta toimiva tietokanta, jonka avulla voidaan nähdä arvioitua tietoa Steam-pelijakelualustalla myytävistä peleistä ja niiden omistajamääristä. (Frequently Asked Questions n.d)

4.1 Dead Cells

Ranskalaisen indie-pelistudio Motion Twinin vuonna 2017 julkaisema Dead Cells kategorisoituu kehittäjiensä mukaan roguevania-nimiseen genreen. Käytännössä Dead Cells on toimintapohjainen 2D-tasoloikkapeli, joka keskittyy intensiivisiin taistelutilanteisiin ja lukemattomien erilaisten ase- ja esineyhdistelmien mahdollistamiin pelityyleihin. Pelissä kuoleminen tarkoittaa sen alusta aloittamista, joten Dead Cells nojaa vahvasti uudelleenpelattavuuteen. (Press Kit n.d.) SteamDB-

sivuston mukaan Dead Cells-pelin omistaa Steam-jakelualustalla noin 700,000 käyttäjää. (Dead Cells n.d)

Dead Cells sisältää paljon pelattavuuteen vaikuttavia ja sitä tehostavia efektejä. Lähes kaikkiin pelissä tehtäviin liikkeisiin ja hyökkäyksiin on liitetty jonkinlainen partikkeliefekti, tai sitä muistuttava tehoste, joiden avulla nähdään mille alueelle ne vaikuttavat. Lisäksi ympäristön tuhoutuessa tai vihollishahmojen kadotessa voidaan nähdä runsasta partikkeliefektien käyttöä, joka antaa pelaajalle välittömän ja selkeän visuaalisen viestin objektin tuhoutumisesta (ks. kuvio 2). Pelattavuuden kannalta vähemmän merkittävät ympäristöefektit ovat puolestaan tärkeässä roolissa Dead Cellsin tunnelman luomisessa. Eri ympäristöihin on toteutettu niihin sopivia efektejä esimerkiksi valaistuksen, sumun, heijastusten tai partikkeliefektien avulla.



Kuvio 2 Dead Cellsin partikkeliefektien käyttö.

Pelin kehittäjät kertovat blogissaan kuinka 3D-animaatioita käytetään Dead Cellsin hahmoanimaatioiden pohjana ennen kuin ne muutetaan 2D-animaatioiksi ja viedään spritesheetteinä pelimoottorin käytettäväksi. Blogin kirjoittaja kuvailee tätä ratkaisua yhden miehen bändiksi, sillä pelin lukemattomien animaatioiden muokkaaminen ja lisääminen on paljon nopeampaa käyttäen 3D-malleja. (Dead Cells – Roguevania Action Platformer 2017) 3D-pohjalta rakennettujen hahmoanimaatioiden seasta voidaan erottaa myös 2D-pohjaisia efektianimaatioita esimerkiksi pelaajahahmon hypätessä tai miekalla iskiessä (ks. kuvio 3). Dead Cells yhdistelee taitavasti molempia

edellä mainittuja tekniikoita ja tekee siten hahmoanimaatioistaan ja niihin liittyvistä efekteistä visuaalisesti mielenkiintoisia.

Kuvio 3 Esimerkki Dead Cellsin 2D-pohjaisesta efektianimaatiosta.

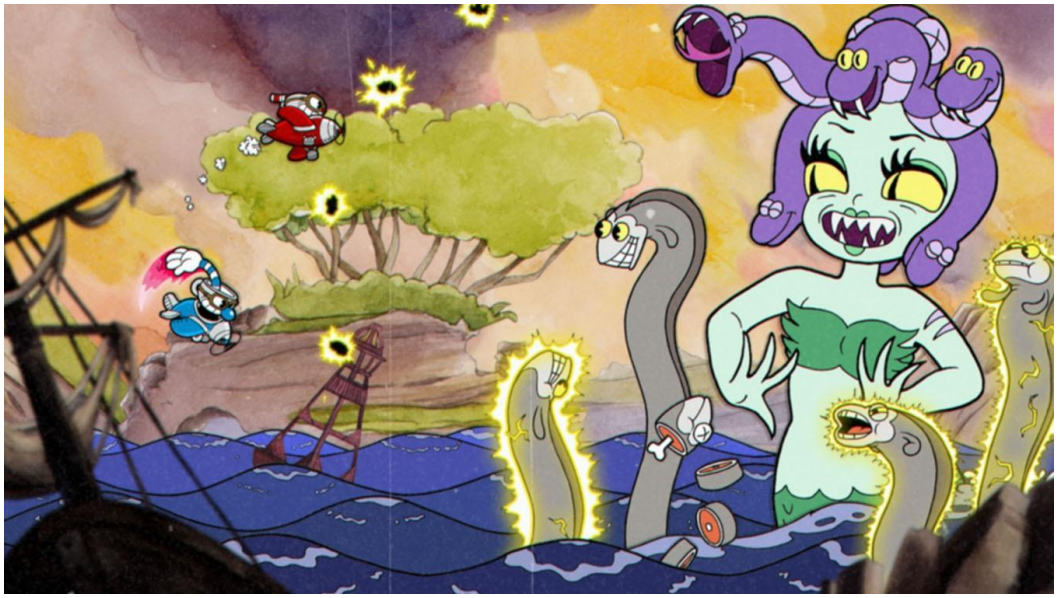
4.2 Darkwood

Vuonna 2013 onnistuneesti joukkorahoitettu ja 2017 julkaistu Darkwood sai erinomaisen vastaanoton pelaajilta sekä kriitikoilta. Darkwood keskittyy tunnelmalliseen kauhuun ja selviytymiseen, sekä sisältää roolipelimäisiä ja seikkailullisia elementtejä. Pelissä tutkitaan synkkää metsää pullollaan resursseja ja salaisuuksia, joita pelaajan täytyy hyödyntää puolustautuessaan lukemattomia vaaroja vastaan ja käyttää niitä metsästä paetakseen. (Acid Wizard Studio n. d.) SteamDB-sivuston mukaan Darkwood-pelin omistaa Steam-jakelualustalla noin 150,000 käyttäjää. (Darkwood n.d) Darkwoodin pääpelimekaniikat keskittyvät pimeyden ympärille, sillä yön tullessa ja huonosti valaistuissa sijainneissa pelaaja joutuu hyödyntämään taskulamppuaan (ks. kuvio 4), jonka avulla pelaaja näkee pimeydessä piileskelevät vaarat ja pitää osan vihollisista kokonaan loitolla. Valaistuksen toteutus ja muut visuaaliset efektit, kuten sade ja ukkosmyrskyt sekä koko ruudulle ilmestyvät tehosteet lisäävät Darkwoodin karmivaa tunnelmaa vaikuttamalla samalla suoraan sen pelattavuuteen.

Kuvio 4 Darkwoodin taskulampun valoefekti. (Acid Wizard Studio, n. d)

4.3 Cuphead

Cuphead on yhdysvaltalaisen Studio MDHR:n vuonna 2017 julkaisema, perinteinen run and gun-genreen kuuluva peli, joka on saanut inspiraationsa 1930-luvun piirretyistä. Pelin visuaalinen ilmeen luomiseen on käytetty käsin piirrettyjä animaatioita ja vesiväripiirroksia. Lisäksi pelin musiikki- ja äänisuunnittelu nojaa vahvasti 1930-luvun tyyliseen jazz-musiikkiin. (Studio MDHR n. d.) SteamDB-sivuston mukaan Cuphead-pelin omistaa Steam-jakelualustalla noin 1,450,000 käyttäjää. Luonnollisia ilmiöitä matkivat visuaaliset efektit ovat isossa osassa Cupheadin visuaalisen ilmeen rakentamisessa. Käsin piirretyt ja animoidut savupilvet, räjähdykset, virtaava vesi, sekä muut efektit noudattavat 1930-luvun piirretyille ominaisia piirteitä ja ne selkeyttävät pelattavuutta antamalla pelaajalle visuaalista tietoa mahdollisista vaaroista tai kerättävistä objekteista (ks. kuvio 5). Perinteisen animaation lisäksi pelissä käytetään jälkikäsittelyefektejä, jolla pelinäkymä on saatu muistuttamaan vanhaa filmirullalta pyörivää elokuvaa. Pelaajan on myös mahdollista muuttaa Cupheadin väriasetuksia, jolloin pelistä tulee vanhoja piirrettyjä kunnioittaen täysin mustavalkoinen.



Kuvio 5 Cupheadin käsin piirrettyä visuaalista ilmettä tehostavat erilaiset efektianimaatiot. (Cuphead n. d.)

Cupheadin kehittäjien mukaan perinteisen animaation toteuttaminen vei eniten aikaa pelin kehityskaaren aikana. Jokainen animaatioframe piirrettiin paperille ja väritettiin, ennen kuin ne tuotiin käytettäväksi pelimoottoriin. Työtaakkaa olisi voinut vähentää tekemällä animaatiot digitaalisesti, mutta pelin kehittäjät halusivat pitää Cupheadin mahdollisimman perinteisenä. Lopulta kehittäjät päätyivät värittämään piirrokset digitaalisesti, sillä ero käsin väritettyihin versioihin oli huomaamaton. (Webster 2017)

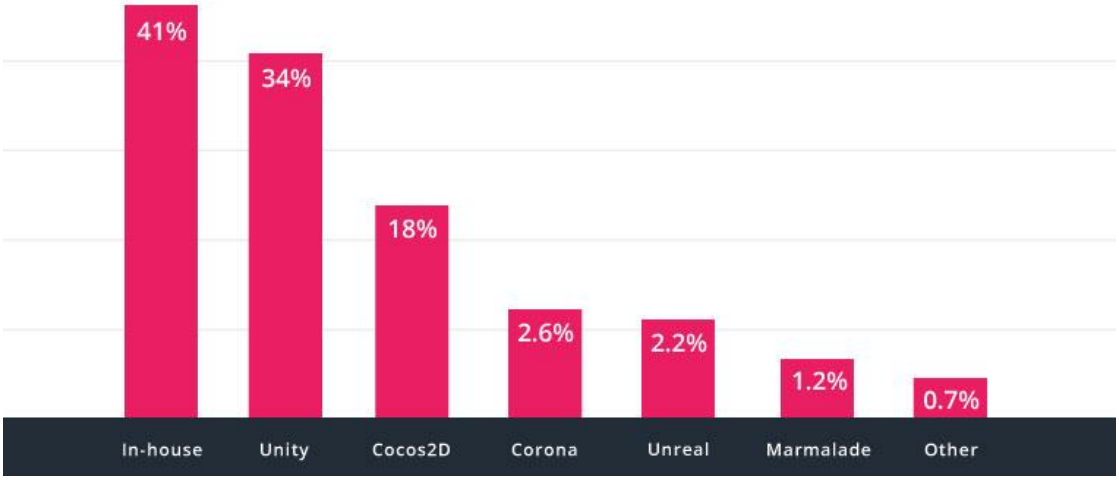
5 Unity Engine ja efektityökalut

Tässä luvussa esitellään Unity-pelimoottori, sekä sen tarjoamat tekniikat ja työkalut, joita voidaan käyttää visuaalisten efektien toteuttamiseen.

5.1 Unity

Unity Technologiesin kehittämä Unity on vuonna 2005 julkaistu pelimoottori, jolla voidaan tehdä 2D-, 3D-, VR- ja AR-pelejä sekä applikaatioita monille eri alustoille. Unity on ladattavissa ilmaiseksi, mutta suuremman liikevaihdon omaaville yrityksille tai lisäominaisuuksia haluaville on tarjolla kuukausimaksulliset Unity Plus-, Pro- ja Enterprise-versiot. (Company Facts. N. d; Unity 2017: The world-leading creation engine. N. d.) Unity on yksi suosituimmista pelimoottoreista sen monipuolisten

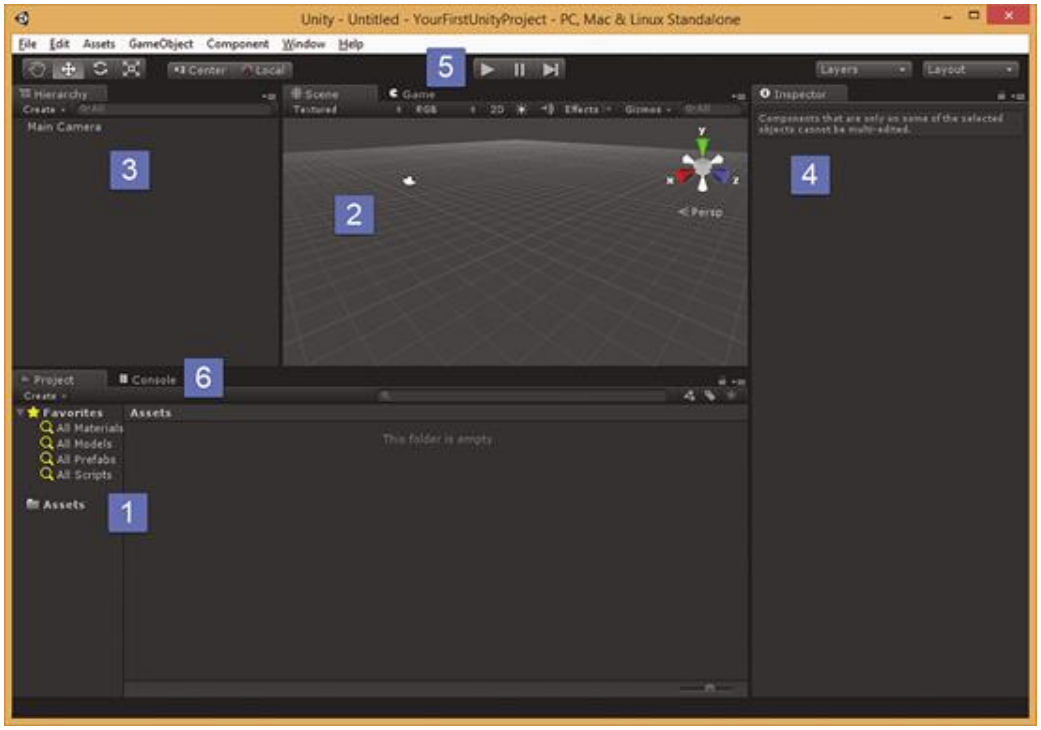
ominaisuuksien ja erilaisten maksusuunnitelmien takia, sillä sen avulla sekä harrastelija- että ammattilaiskäyttäjät voivat kehittää pelejä lähes kaikille alustoille. Laajaan ominaisuuspakettiin sisältyy varsinaisten editorityökalujen lisäksi esimerkiksi assettikauppa, integroitu versionhallinta, analytiikkatyökalut, sekä monetisaatioon ja mainostukseen tarvittavat työkalut. Varsinkin mobiilimarkkinoilla Unityn merkitys on suuri, sillä esimerkiksi vuonna 2016 34% suosituimmasta tuhannesta ilmaisesta mobiilipelistä tehtiin Unityllä (ks kuvio 6).



Kuvio 6 Vuonna 2016 34% suosituimmasta tuhannesta ilmaisesta mobiilipelistä tehtiin Unityllä. (Company Factsn. d)

5.2 Työskentely Unityssä

Unityssä jokainen projekti koostuu yhdestä tai useammasta Scene-tiedostosta, jotka sisältävät metadataa projektissa käytettävistä resursseista kyseisessä Scenessä. Scenet sisältävät GameObject-nimellä kulkevia peliobjekteja, joihin voidaan lisätä toiminnallisuuksia erilaisien komponenttien avulla. Esimerkiksi partikkeliefektiä tehtäessä luodaan ensin peliobjekti, johon kiinnitetään Particle System-komponentti. Unity perustuu natiivi C++-kieleen, ja siinä kirjoitetaan koodia joko C#, tai JavaScript-ohjelmointikielillä. Koodia editoidaan Unityssä tuplaklikkaamalla Script-tiedostoa projektinäkymässä, joka avaa vakiovaihtoehtona käytettävän Mono-Develop-editorin. Halutessaan koodieditoriksi voidaan vaihtaa myös Visual Studio. Luotuja Script-tiedostoja käytetään peliobjekteissa komponentteina. (Tuliper 2014)



Kuvio 7 Unityn oletusikkunanäkymä (Tuliper 2014)

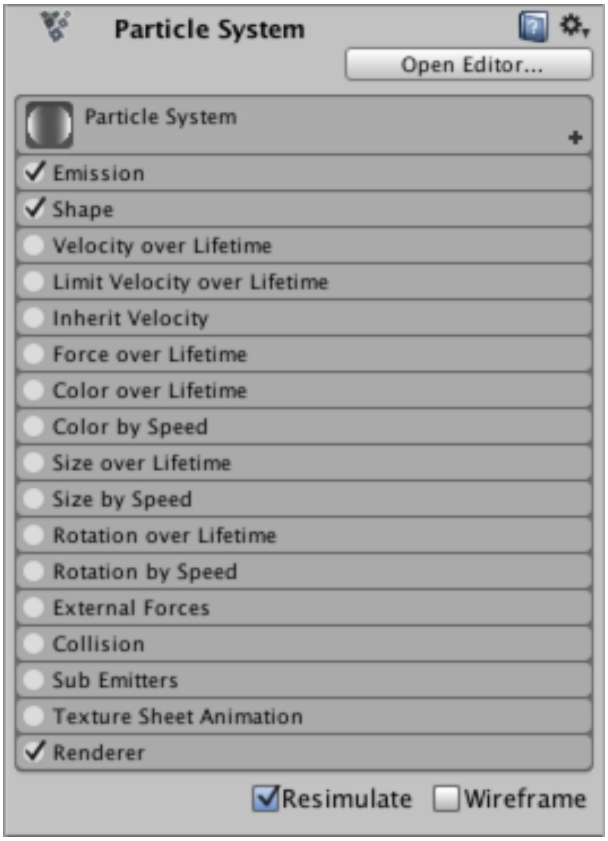
Unityn oletusikkunanäkymässä kuviossa 4 voidaan nähdä seuraavat osiot:

1. Project: Sisältää kaikki projektin tiedostot. Tänne voidaan lisätä tiedostoja tietokoneelta raahaamalle ne ikkunanäkymään.
2. Scene: Tällä hetkellä auki oleva Scene.
3. Hierarchy: Sisältää kaikki Scenessä olevat peliobjektit.
4. Inspector: Näyttää valitun peliobjektin komponentit, eli ominaisuudet.
5. Toolbar: Sisältää työkalut peliobjektien liikuttamiseen, pelin käynnistämiseen ja pysäyttämiseen, sekä Unityn ikkunanäkymän muokkaamiseen.
6. Console: Tämä ikkuna näyttää debug-ilmoitukset, virheilmoitukset, varoitukset ja muut vastaavat viestit.

5.3 Efektityökalut

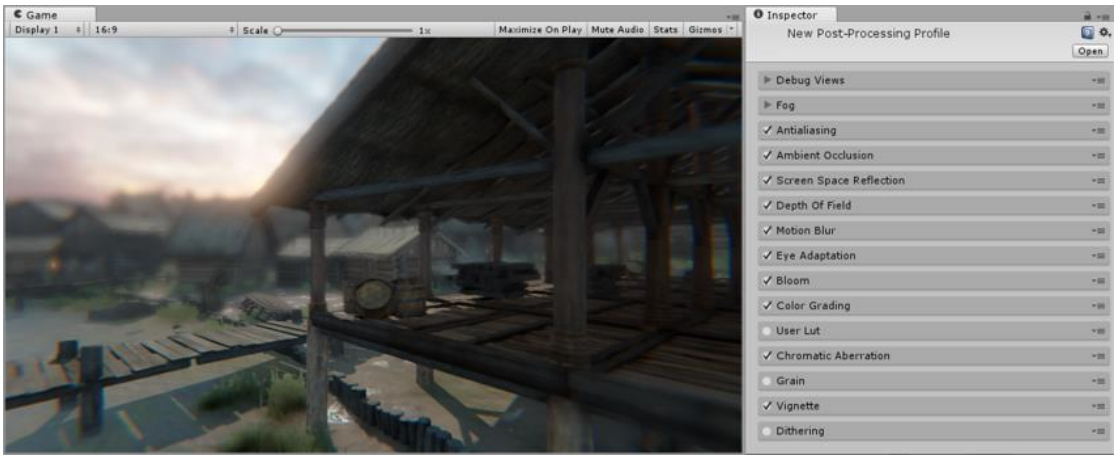
Unityn editorin kautta visuaalisia efektejä voidaan toteuttaa helposti valmiiden komponenttien ja työkalujen avulla. Niitä voidaan käyttää kameroissa, peliobjekteissa, valoissa ja muissa kehitettävän pelin elementeissä. (Unity Documentation 2017b.)

Unityn oma partikkelijärjestelmä on peliobjektiin kiinnitettävä komponentti, joka sisältää paljon partikkelien käyttäytymiseen, ulkomuotoon ja renderöitymiseen vaikuttavia ominaisuuksia. Käytännöllisyyden vuoksi kyseiset ominaisuudet on jaettu osiin, joita kutsutaan moduuleiksi. (Unity Documentation 2017c.)



Kuvio 8 Unity Enginen partikkelijärjestelmän moduulinäkymä (Unity Documentation 2017c.)

Erilaiset valmiina käytettävät jälkikäsittelyefektit kuuluvat myös Unity Enginen työkaluvalikoimaan. Niillä voidaan toteuttaa efektejä, kuten hohtoa, värikorjausta, liikkeen sumennusta tai vaikuttaa syväterävyyteen. (Unity Documentation 2017e)



Kuvio 9 Unity Enginen jälkikäsittelyefektit

5.4 Shaderit

Unityssä shadereita voidaan kirjoittaa Surface-, Vertex-, Fragment-, sekä Fixed Function Shadereilla. Riippumatta käytössä olevasta shader-tyypistä, varsinainen shader-koodi pakataan aina ShaderLab-kielen sisään, jota käytetään shaderien rakenteen järjestämiseen. (Unity Documentation 2017d)

Surface Shadereita käytetään yleisimmin, jos toteutettavan shaderin täytyy toimia valojen ja varjojen kanssa. Niiden avulla voidaan kirjoittaa helposti monimutkaisia shadereita todella kompaktilla tavalla. Surface shadereita taas kirjoitetaan yleensä vain muutamalla rivillä Cg- tai HLSL-ohjelmointikieltä, sillä suurin osa siitä generoituu automaattisesti joukoksi vertex- ja fragment shadereita. Vertex- ja fragment shadereita voidaan hyödyntää, kun shaderin ei ole tarkoitus toimia valaistuksen kanssa ja kun halutaan toteuttaa efektejä, joita surface shaderit eivät pysty käsittelemään. Tällaiset shaderit ovat joustavin tapa toteuttaa efektejä, mutta niihin joutuu kirjoittamaan manuaalisesti enemmän koodia ja niitä on vaikeampi saada toimimaan valaistuksen kanssa. Vertex- ja fragment shadereita kirjoitetaan myös joko Cg- tai HLSL-ohjelmointikielellä. Todella yksinkertaisia efektejä voidaan tehdä myös fixed function shadereilla, mutta on suositeltavampaa kirjoittaa mielummin ohjelmoitavia shadereita, sillä ne ovat paljon joustavampia vaihtoehtoja. Fixed function shaderit kirjoitetaan kokonaan ShaderLab-kielellä, mutta nekin muutetaan lopulta vertex- ja fragment shadereiksi. (Unity Documentation 2017d.)

Shadereita on mahdollista käyttää Unityssä suoraan scripteistä käsin, mutta materiaalit ovat varsinainen käyttöliittymä shadereita varten. Ne määrittävät millä tavalla kukin taso renderöidään sisältämällä viittaukset käyttämiinsä tekstuureihin, värisävymuutoksiin ja muihin shaderin vaatimiin tietoihin. (Unity Documentation 2017f.)

Unity Engineen on tulossa opinnäytetyön kirjoittamishetkellä Shader Graph-niminen työkalu, jonka avulla shadereita voidaan rakentaa visuaalisesti. Koodin kirjoittamisen sijasta shaderin osia sekä toiminnallisuuksia luodaan graafisen käyttöliittymän kautta. Vastaavia työkaluja on jo aikaisemmin ollut saatavilla maksullisena Unityn Asset Storessa ja niitä ilmenee myös ilmaisena kilpailevien pelimoottoreiden, kuten Unreal Enginen mukana. (Beta Program n.d)

6 Visuaalisten efektien optimoiminen Unityssä

Tässä luvussa perehdytään tarkemmin Unity Enginen graafisen renderöinnin prosessiin ja selvitetään minkälaisia ongelmia siinä voi ilmetä. Lisäksi tutkitaan, partikkeleilla ja shadereilla tehdyissä efekteissä tulee ottaa huomioon, sekä kuinka ongelmakohtien etsiminen tapahtuu käyttäen apuna Unity Enginen analysointityökaluja.

6.1 Graafisen renderöinnin prosessi

Renderöintiä verrataan yleensä putken malliseen linjastoon, jonka läpi prosessoitavien tehtävien on mentävä, jotta kukin ruutu voidaan renderöidä. Sekä prosessorin, että näytönohjaimen on viimeisteltävä kaikki tehtävänsä ennen ruudun renderöimistä. Epäoptimoidut ja tehottomat ratkaisut aiheuttavat niin sanottuja pullonkauloja, jolloin tehtävät vievät liikaa aikaa prosessoituessaan. Ongelmia ilmenee myös jos linjastoon yritetään työntää liian paljon dataa kerrallaan, eikä prosessointitehoa ole tarpeeksi. (Vivo & Lowe 2015; Unity Tutorials n. d.)

Unity Enginen ohjemateriaalien mukaan (Unity Tutorials n. d) jokaista renderöitävää ruutua kohden käydään läpi yksinkertaisuudessaan seuraavat vaiheet:

1. Tietokoneen prosessori selvittää mitä joudutaan renderöimään ja miten.
2. Prosessori lähettää ohjeistuksen näytönohjaimelle.
3. Näytönohjain renderöi näytölle halutun materiaalin prosessorin ohjeiden mukaisesti.

Ensimmäisessä vaiheessa tietokoneen prosessori määrittelee, mitkä scenen objektit renderöidään, sekä kerää niistä tietoa ja muuttaa tämän datan komennoiksi joita kutsutaan piirtokutsuiksi. Piirtokutsuissa on tietoa yksittäisistä mesheistä ja siitä, kuinka kyseinen mesh on tarkoitus renderöidä. Hyvänä esimerkkinä voidaan käyttää tietoa siitä, minkälaisia tekstuureita renderöitävä mesh käyttää. Tietokoneen prosessori luo jokaisesta piirtokutsusta datapaketin, jota kutsutaan nimellä batch. (Unity Tutorials n. d.)

Toisessa vaiheessa tietokoneen prosessori saattaa lähettää näytönohjaimelle SetPass-nimisen kutsun, joka kertoo sille minkälaisia asetuksia seuraavan meshin

renderöimiseen tulee käyttää. Kutsu lähetetään vain jos seuraavaksi renderöitävä mesh tarvitsee muutoksia edelliseen meshiin verrattuna. SetPass-kutsun jälkeen prosessori lähettää piirtokutsun näytönohjaimelle, jolloin piirtokutsu ohjeistaa näytönohjainta renderöimään meshin käyttäen uusimman SetPass-kutsun määrittelemiä asetuksia. (Unity Tutorials n. d.)

Kolmannessa vaiheessa näytönohjain käsittelee prosessorin sille lähettämiä tehtäviä niiden tulojärjestyksessä ja päivittää tarvittavat SetPass-kutsun määrittelemät asetukset. Suoritettavan tehtävän ollessa piirtokutsu, näytönohjain renderöi meshin ja tekee samalla shadereissa määritellyt muutokset sen vertekseihin tai yksittäisiin pikseleihin. Tämä prosessi käydään läpi niin monta kertaa, kunnes kaikki prosessorin lähettämät tehtävät ovat suoritettu näytönohjaimen toimesta. (Unity Tutorials n. d.)

6.2 Analysointityökalut

Unityn Profiler-työkalun avulla voidaan tarkastella reaaliajassa, kuinka paljon pelimoottorilla kuluu aikaa erilaisten työtehtävien suorittamiseen. Profilerilla on mahdollista analysoida näytönohjaimen, prosessorin, välimuistin, renderöinnin ja audion suorituskykyä. (Unity Tutorials n. d)

Frame Debugger-työkalulla on mahdollista nähdä kuinka jokainen ruutu renderöidään vaihe vaiheelta ja nähdään tarkkaa tietoa, kuten mitä näytölle renderöidään kunkin piirtokutsun aikana, shaderien ominaisuukset jokaisella piirtokutsulla, sekä näytönohjaimelle lähetettyjen tapahtumien järjestys. (Unity Tutorials n. d)

6.3 Optimointitekniikat

Visuaalisia efektejä optimoidessa tulee kiinnittää huomiota lähettävien datapakettien, eli batchien määrään ja varmistaa, että pikseleitä ei renderöidä näytölle enempää kuin näytönohjain niitä voi prosessoida. Jokainen batch voi sisältää dataa useita eri objekteja varten, jos ne käyttävät samaa instanssia samasta materiaalista ja kyseinen materiaali käyttää identtisiä asetuksia. Esimerkiksi monimutkaista, erilaisia tekstuureja käyttävässä partikkeliefektissä voidaan käyttää

spritesheettiä tekstuurien säilyttämistä varten, jolloin kaikki partikkeliefektien osat saadaan mahtumaan samaan datapakettiin. (Unity Tutorials n. d.)

Renderöitävien pikseleiden määrää voidaan vähentää käyttämällä vähemmän fragment shadereita ja välttäen ylitsepiirtoa mahdollisimman paljon. Läpinäkyvyyttä sisältävät, päällekäin sijaitsevat objektit joudutaan ylitsepiirtämään, eli niiden alueella sijaitsevat pikselit renderöidään useita kertoja. Ylitsepiirtoa voidaan tarkastella Unityn scenenäkymän Overdraw-näkymästä. (Unity Tutorials n. d.)

7 Ohjeistus

7.1 Tarpeen määrittely

Jyväskylän ammattikorkeakoulun tietojenkäsittelyn koulutusohjelmassa käsiteltiin pelinkehityksen efektiteknologioita melko pintapuolisesti ja oppimateriaalia visuaalisten efektien toteuttamisesta ei ollut. Omatoimiset opiskelijat pystyivät etsimään tarvittavan tiedon visuaalisten efektien toteuttamista varten, mutta aihepiiri on sen verran laaja ja kompleksi, että niihin liittyvien perusteiden omaksuminen ja oikeiden lähteiden löytäminen voi olla haastavaa. Lisäksi useissa opiskelijoiden käyttämissä ohjemateriaaleissa ei välttämättä kerrota tarkemmin esiteltävän asian toimivuudesta tai siihen liittyvistä ongelmista, kuten esimerkiksi optimoinnin tärkeydestä. Tästä syystä opiskelijoiden tuottamissa peleissä voi ilmetä suorituskykyongelmia esimerkiksi partikkeliefektejä väärinkäyttäessä, joka on iso ongelma varsinkin mobiilipelejä kehittäessä. Ongelmana oli siis helposti ymmärrettävän ja tarvittavan taustateorian kattavan ohjeistuksen puute.

7.2 Suunnittelu ja toteutus

Ohjeistuksen suunnittelu lähti liikkeelle kriteereistä, jotka ovat aloittelijaystävällisyys, käytännönläheisyys ja käytettyjen tekniikoiden perusteiden ymmärtäminen optimointi mielessä pitäen. Ohjeistuksessa käytettävien esimerkkien alustaksi haluttiin pelimoottori Unity, sillä se on yksi suosituimmista pelimoottoreista, jota käytetään myös Jyväskylän ammattikorkeakoulun tietojenkäsittelyn koulutusohjelman opinnoissa. Unityssä on nopeasti omaksuttava graafinen

käyttöliittymä ja se tarjoaa valmiita työkaluja visuaalisten efektien toteuttamista varten, sekä mahdollistaa myös omien työkalujen ja efektien toteuttamisen esimerkiksi shaderien avulla. Tämä helpottaa materiaalin toteuttamista sekä aiheeseen vasta tutustuneille, että kokeneemmillekin opiskelijoille. Ohjeistuksen esimerkit suunniteltiin rajattavaksi 2D-pelinkehitystä varten, jotta ne olisivat helpommin ymmärrettävissä. Kaksiulotteisessa peliympäristössä esimerkit voidaan pitää yksinkertaisempina ja opittuja tekniikoita voidaan hyödyntää myöhemmin myös 3D-pelinkehityksessä.

Ohjeistuksen sisällön haluttiin koostuvan Unityn valmiiden efektikomponenttien esittelystä esimerkkeineen, jälkikäsittelytyökalujen läpikäymisestä, sekä shaderien toiminnallisuudesta ja niiden käyttämän ohjelmointikielen esittelystä. Partikkeliefektit ja shaderit haluttiin pitää ohjeistuksen ytimenä. Visuaaliset efektit ovat luonteeltaan toimivia vain liikkuvassa kuvassa, eikä niitä voi havainnollistaa toimivina esimerkkeinä pelkkien valokuvien tai kuvankaappausten avulla. Tästä syystä toteutettavasta ohjeistuksesta haluttiin monimediapaketti, joka sisältää tekstiä, kuvia, sekä videoita.

Ohjeistusta alettiin suunnitella kirjallisena Word-dokumenttina ja sille toteutettiin alustava sisältöluettelo (ks. kuvio 10). Koska partikkeliefektit ja shaderit haluttiin pitää paketin keskipisteenä, niille varattiin omat kappaleensa ja muut tekniikat päätettiin sisällyttää yhden otsikon alle. Partikkeliefekteille ja shadereille suunniteltiin kappalerakenne joka sisälti perusteet, optimointiin liittyvät tekijät sekä käytännön esimerkit.

Kuvio 10 Ohjeistuksen alustava sisältöluettelo

Ohjeistusta varten toteutettiin esimerkit moniosaisesta partikkeliefektistä ja yksinkertaisesta shaderista, joita tehdessä kiinnitettiin huomiota ylitsepiirron välttämiseen ja oikeaoppiseen sprite sheettien käyttöön. Efektien toteuttamisen prosessi kirjattiin ohjeistukseen vaihe vaiheelta mahdollisimman yksinkertaistettuna, jotta ohjeistuksen lukija pystyy helposti tekemään esimerkkien efektit pelkän tekstin avulla. Tekstin yhteyteen lisättiin myös havainnollistavia kuvia auttamaan ohjeistuksen ymmärtämistä. Esimerkkien jälkeen kirjoitettiin lyhyet kappaleet kunkin efektityypin optimoimiseen liittyvistä huomioista.

8 Pohdinta

Tässä luvussa kerrotaan opinnäytetyön tuotoksena tehdyn ohjeistuksen lopputuloksesta. Lisäksi selvennetään tutkimuksen lopputuloksia ja johtopäätöksiä vastaamalla opinnäytetyön tutkimuskysymyksiin sekä pohditaan tutkimuksen luotettavuutta ja jatkokehitysmahdollisuuksia.

8.1 Työn tulos ja johtopäätökset

Opinnäytetyön tuloksena kehitettiin käytännönläheinen ohjeistus visuaalisten efektien toteuttamisesta 2D-peleihin Unity Enginessä, jota Jyväskylän

ammattikorkeakoulun tietojenkäsittelyn tutkinto-ohjelman opiskelijat voivat hyödyntää opinnoissaan. Alkuperäisen suunnitelman mukaan ohjeistuksesta haluttiin monimediapaketti, joka sisältää tekstiä, kuvia sekä videoita. Videon tuottaminen olisi ollut kuitenkin liian työlästä ja aikataulullisista syistä ohjeistuksesta tehtiin helposti seurattava tekstidokumentti, joka sisältää vaihe vaiheelta selitetyt esimerkit havainnollistavien kuvien tehostamana.

Ohjeistuksen laajuutta pohdittiin sen toteutusprosessin aikana useaan otteeseen. Lopulta päädyttiin ratkaisuun sisällyttää ohjeistukseen vain yleisimmät 2D-peleissä käytetyt efektit, joita voidaan toteuttaa Unity Enginellä; partikkeliefektit ja shaderit. Vaikka frame animaatiot ovat yleinen tapa toteuttaa visuaalisia efektejä 2D-peleissä, niitä ei haluttu sisällyttää ohjeistukseen, sillä Unity Enginen käyttö haluttiin pitää etusijalla kuvankäsittelyohjelmalla tehtävien animaatioiden sijaan. Ohjeistuksen laajuuden pienentämisestä huolimatta sen lukijaa haluttiin ohjata oikeaan suuntaan sisällyttämällä linkkejä sivustoille, joista saa lisätietoa kustakin efektitekniikasta.

Tutkimuksen tulokset ovat osittain yleistettävissä, sillä tutkittujen teknologioiden ja efektitekniikoiden käyttö on pelialalla yleistä. Opinnäytetyön tuloksena toteutettu ohjeistus olettaa sen lukijan osaavan vain Unity Enginen perusteet, joka mahdollistaa sen seuraamisen myös kokemattomien opiskelijoiden toimesta. Ohjeistus on kuitenkin tehty Jyväskylän ammattikorkeakoulun tietojenkäsittelyn koulutusohjelmaa varten ja siinä käsitellyt tekniikat on tehty huomioiden kyseisen koulutusohjelman tarpeita. Niitä varten kehitettiin vaihtoehtoinen toimintatapa.

Mitä visuaaliset efektit ovat?

Tutkimuksen perusteella saatiin selville, että visuaaliset efektit ovat kaikenlaisessa liikkuvassa mediassa käytettävää visuaalista materiaalia, joilla pyritään luomaan vaikutelma jostain luonnollisesta ilmiöstä tai joilla parannellaan visuaalisen materiaalin tehokkuutta erilaisten jälkikäsittelytekniikoiden avulla. Lisäksi saatiin selville, että digitaalisissa peleissä visuaaliset efektit voidaan jakaa pelattavuuteen vaikuttaviin- tai ympäristöllisiin efekteihin, joiden painoarvo vaihtelee projektikohtaisesti.

Minkälaisia visuaalisia efektejä moderneissa 2D-videopeleissä käytetään?

Kolmen erityyppisen ja suositun 2D-videopelin analysoinnin jälkeen tultiin siihen tulokseen, että modernit 2D-videopelit käyttävät paljon eri tyyppisiä visuaalisia efektejä riippuen niiden graafisen ilmeen tyylistä ja pelattavuuden tehostuksen tarpeista. Erilaiset partikkeliefektit, shadereilla toteutetut efektit, sekä frame animaatioilla piirretyt efektit ovat yleisimpiä tapoja toteuttaa visuaalisia efektejä. Edellä mainittuja efektejä käytetään joko sellaisenaan, tai yhdistellään muiden efektien kanssa optimaalisen tehosteen toteuttamiseksi.

Kuinka visuaalisia efektejä voidaan toteuttaa optimoidusti 2D peleihin Unity-pelinkehitysympäristössä?

Visuaalisia efektejä toteuttaessa Unity Enginellä tulee kiinnittää huomiota tietokoneen prosessorin ja näytönohjaimen käsittelemien datapakettien sekä pikseleiden määrään. Unityllä toteutetun pelin visuaalisten efektien suorituskykyä voidaan seurata siihen rakennetuilla työkaluilla, jolloin ongelmakohdat ovat helposti löydettävissä. Yleisimpiä ratkaisuja visuaalisten efektien optimoimiseen ovat peliin ladattavien tekstuurien vähentäminen ja ylitsepiirron välttäminen, jolloin näytönohjaimelle lähetettävät datapaketit ja sen käsittelemät pikselimäärät vähenevät.

8.2 Luotettavuus

Työn tekijällä on takanaan 3,5 vuotta kestänyt pelialan koulutus, sekä reilun kolmen vuoden kokemus digitaalisten pelien toteuttamisesta ja tietokonegrafiikasta. Lisäksi opinnäytetyön kirjoittamisen aikana tekijä työllistyi pelialan yritykseen tekemään teknisen graafikon työtehtäviä. Edellä mainittu kokemus on antanut työn tekijälle vahvaa tietämystä ja laajempaa näkökulmaa pelien kehittämisestä, sekä auttanut ohjeistuksen toteutuksen mahdollistamisessa. Tutkimuksessa analysoituja pelejä tarkisteltiin kerätyn taustateorian, pelinkehittäjien kirjoituksien ja työn tekijän pelialan kokemuksen perusteella. Analysoidut pelit valittiin taustateorian pohjalta kerätyn tiedon, vuosiluvun ja SteamDB-sivuston arvioitujen käyttäjämäärien perusteella, sillä pelien suosio ja modernius haluttiin varmistaa.

Pedagogista taustateoriaa ohjeistuksen toteutuksen osalta ei ole, joka saattaa vaikuttaa sen pätevyyteen. Ohjeistuksesta kuitenkin pyrittiin tekemään

mahdollisimman selkeä, jottei se jättäisi mitään tulkinnanvaraiseksi. Lisäksi ohjeistukseen sisällytetyt linkit jatko-opiskelua varten varmistavat, että sen lukija osaa hakea lisätietoa oikeasta paikasta. Valmista ohjeistusta olisi myös voinut testata tietojenkäsittelyn koulutusohjelman opiskelijoilla ja haastatella heitä ohjeistuksen laadun varmistamiseksi, mutta aikataulullisista syistä ohjeistus toteutettiin pelkästään toimeksiantajan toiveiden perusteella.

Tutkimuksessa käytettyjä lähteitä kerättiin runsaasti ja esimerkiksi shadereista kirjoittaessa pyrittiin ottamaan huomioon eri laitevalmistajien sekä ohjelmointikielien näkökulmat. Suurin osa tutkimuksen lähteistä on verkkoartikkeleita, joka voidaan perustella pelialan kirjallisuuden pienestä määrästä. Kirjalliset lähteet olivat kuitenkin jossain luvuissa välttämättömiä, sillä esimerkiksi syvemmälle perehdyttävää taustateoriaa visuaalisten efektien toteuttamisesta efektianimaatioina ei Gillandin (2009) lisäksi juuri ollut. Blogikirjoitukset ja haastattelut pyrittiin valitsemaan tunnetuilta pelialan verkkosivuilta tai pelinkehittäjien omilta sivustoilta ja lisäksi haastatteluissa esiintyviltä henkilöillä haluttiin olevan paljon kokemusta pelialalla työskentelemisestä.

8.3 Jatkokehitys

Tutkimuksessa on jatkokehityspontentiaalia, sillä kukin opinnäytetyössä mainittu efektityyppi sisältää paljon erilaisia tekniikoita ja toimintatapoja. Syvällinen perehtyminen jokaiseen efektitekniikkaan on todella laaja aihealue ja niiden toteuttamisesta saa tehtyä kuhunkin aihealueeseen syventyviä tutkimuksia. Esimerkiksi shaderien toteuttamisesta voitaisiin tehdä syventävä tutkimus, jonka perusteella tehtäisiin useita ohjeistuksen esimerkkitapauksia ja joiden pohjalta opeteltaisiin tarkemmin shaderien käyttämän ohjelmointikielen syntaksia.

Lähteet

Acid Wizard Studio. N. d. Darkwood-pelin lehdistötiedote. Viitattu 14.4.2018. http://www.darkwoodgame.com/presskit/.

Beta Program. N. d. What's included in 2018.1b. Unity Enginen betaversion esittelysivu. Viitattu 14.4.2018. https://unity3d.com/unity/beta/2018.1b.

Bisneksen ja tietotekniikan osaaja. N. d. JAMKin tietojenkäsittelyn koulutusohjelman esittelysivu. Viitattu 14.4.2018. https://www.jamk.fi/fi/Koulutus/Liiketalouden-ala/tradenomi-tietojenkasittely/.

Careers. N. d.a. Housemarque rekrytointisivu. Viitattu 14.4.2018. http://www.housemarque.com/company/careers/.

Careers. N. d.b. Nitro Games rekrytointisivu. Viitattu 14.4.2018. https://www.nitrogames.com/careers/.

Company Facts. N. d. Faktoja Unity Technologies-yrityksestä. Viitattu 14.4.2018. https://unity3d.com/public-relations.

Cuphead. N. d.a. Made with Unity. Verkkojulkaisu Unity Enginellä tehdystä Cuphead-pelistä. Viitattu 19.4.2018. https://unity.com/madewith/cuphead.

Cuphead. N. d.b. Charts & Graphs. SteamDB-sivuston datasivu Cupheadista. Viitattu 2.5.2018. https://steamdb.info/app/268910/graphs/.

Darkwood. N. d. Charts & Graphs. SteamDB-sivuston datasivu Darkwoodista. Viitattu 2.5.2018. https://steamdb.info/app/274520/graphs/.

Dead Cells – Roguevania Action Platformer. 2017. Motion Twin foorumipostaus. Viitattu 14.4.2018. https://forums.tigsource.com/index.php?topic=59381.0.

Dead Cells. N.d. Charts & Graphs. SteamDB-sivuston datasivu Dead Cellsistä. Viitattu 2.5.2018. https://steamdb.info/app/588650/graphs/.

Frequently Asked Questions. N.d. SteamDB-sivuston usein kysytyt kysymykset-osio. Viitattu 2.5.2018. https://steamdb.info/faq/.

Game Graphics 101: Rendering 2D on GPU. 2016. Kirjan testijulkaisu. Viitattu 14.4.2018. http://ithare.com/game-graphics-101-rendering-2d-on-gpu-shaders/.

Gilland, J. 2009. Elemental Magic. The Classical Art of Hand-Drawn Special Effects Animation. E-kirja. Oxford: Elsevier.

Heazlewood, J. 2013. Unity Sparks System. Blogikirjoitus. Viitattu 14.4.2018. https://seagullcity.wordpress.com/2013/10/13/unity-sparks-system/.

Jobs in Critical Force. N. d. Critical Force rekrytointisivu. Viitattu 14.4.2018. http://criticalforce.fi/vacancy/.

Joseph Gilland. N. d. Visual Effects Director, Autor. Joseph Gillandin LinkedIn-profiilisivu. Viitattu 19.4.2018. https://www.linkedin.com/in/joegilland/.

Kananen, J. 2012. Kehittämistutkimus opinnäytetyönä. Kehittämistutkimuksen kirjoittamisen käytännön opas. Jyväskylä: Jyväskylän ammattikorkeakoulu.

Kananen, J. 2015. Kehittämistutkimuksen kirjoittamisen käytännön opas. Miten kirjoitan kehittämistutkimuksen vaihe vaiheelta. Jyväskylä: Jyväskylän ammattikorkeakoulu.

Luten, E. N. d. Preface: What is OpenGL? Verkkojulkaisu. Viitattu 14.4.2018. http://openglbook.com/chapter-0-preface-what-is-opengl.html.

Press Kit. N. d. Dead Cells-pelin lehdistötiedote. Viitattu 14.4.2018. https://motion-twin.com/presskit/81.

Rodriguez, J. 2013. GLSL Essentials. Enrich your 3D scenes with the power of GLSL! Birmingham: Packt Publishing.

Sherrod, A. & Jones, W. 2011. Beginning DirectX 11 Game Programming. E-kirja peliohjelmoinnista. Boston: Course Technology.

Studio MDHR. N. d. Cuphead-pelin lehdistötiedote. Viitattu 19.4.2018. http://www.studiomdhr.com/press/.

Tokarev, K. 2017. VFX for Games Explained. Haastattelu artikkelimuodossa. Viitattu 14.4.2018. https://80.lv/articles/vfx-for-games-explained/.

Tuliper, A. 2014. Unity: Developing Your First Game with Unity and C#. Viitattu 14.4.2018. https://msdn.microsoft.com/en-us/magazine/dn759441.aspx.

Tutustu JAMKiin: strategia ja vahvuusalat. N. d. JAMKin esittelysivu. Viitattu 14.4.2018. https://www.jamk.fi/fi/Tietoa-JAMKista/Tutustu-JAMKiin/.

Unity 2017: The world-leading creation engine. N. d. Unity Enginen esittelysivu. Viitattu 14.4.2018. https://unity3d.com/unity.

Unity Documentation. 2017a. What is a Particle System? Unity Enginen dokumentaatiosivusto. Viitattu 14.4.2018. https://docs.unity3d.com/Manual/PartSysWhatIs.html.

Unity Documentation. 2017b. Visual Effects Reference. Unity Enginen dokumentaatiosivusto. Viitattu 14.4.2018. https://docs.unity3d.com/Manual/comp-Effects.html.

Unity Documentation. 2017c. Particle System. Unity Enginen dokumentaatiosivusto. Viitattu 14.4.2018. https://docs.unity3d.com/Manual/class-ParticleSystem.html.

Unity Documentation. 2017d. Writing Shaders. Unity Enginen dokumentaatiosivusto. Viitattu 14.4.2018. https://docs.unity3d.com/Manual/ShadersOverview.html.

Unity Documentation. 2017e. Post-processing overview. Unity Enginen dokumentaatiosivusto. Viitattu 19.4.2018. https://docs.unity3d.com/Manual/PostProcessingOverview.html.

Unity Documentation. 2017f. Materials, Shaders & Textures. Unity Enginen dokumentaatiosivusto. Viitattu 19.4.2018. https://docs.unity3d.com/Manual/Shaders.html.

Unity for 2D. N. d. Unity Enginen 2D-ominaisuuksien esittelysivu. Viitattu 17.4.2018. https://unity3d.com/solutions/2d.

Unity Tutorials. N. d. Optimizing graphics rendering in Unity Games. Unity Enginen ohjeistussivu. Viitattu 18.4.2018. https://unity3d.com/learn/tutorials/temas/performance-optimization/optimizing-graphics-rendering-unity-games.

Webster, A. 2017. Cuphead: Creating a Game That Looks Like a 1930s Cartoon. Verkkojulkaisu. Viitattu 19.4.2018. https://www.theverge.com/2017/9/28/16378364/cuphead-art-design-1930s-animation.

What is visual effects? 2012. LightRay Visual Effects blogisivu. Viitattu 14.4.2018. https://vfxforfilm.wordpress.com/2012/09/12/whats-visual-effects/.

Vivo, P. & Lowe, J. 2015. The Book of Shaders. Verkkojulkaistu kirja. Viitattu 14.4.2018. https://thebookofshaders.com/01/.

Liitteet

Liite 1. Visuaalisten efektien toteuttaminen Unity Enginessä

Visuaalisten efektien toteuttaminen Unity Enginessä

Sisältö

9 Johdanto

Tämän ohjeistuksen tarkoitus on tutustuttaa sen lukija partikkelijärjestelmien ja shaderien avulla tehtävien visuaalisten efektien toteuttamiseen Unity Enginellä. Ohjeistuksessa käydään läpi peleissä yleisimmin käytettyjä tekniikoita ja toteutetaan efektejä vaihe vaiheelta selitettyjen esimerkkien avulla. Esimerkit ovat rajattu 2D-peleihin käytettäväksi, mutta opeteltuja tekniikoita voidaan hyödyntää myös 3D -pelikehityksessä. Oletuksena ohjeistuksen lukija osaa Unityn käyttöliittymän perusteet. Aikaisempaa kokemusta visuaalisten efektien toteuttamisesta ei tarvita. Esimerkeissä käytetään Unity-pelimoottoria.

Visuaalisia efektejä voidaan toteuttaa myös muilla valmiilla efektikomponenteilla ja työkaluilla, joiden läpikäyminen ei sisälly tämän ohjeistuksen laajuuteen. Katso https://docs.unity3d.com/Manual/comp-Effects.html & https://docs.unity3d.com/Manual/PostProcessingOverview.html tutustuaksesi Unity Enginen valmiisiin efektityökaluihin.

Unity Enginen käyttöön liittyvää tarkempaa tietoa tarvitessa lukijaa suositellaan tutustumaan Unityn manuaaliin. Linkit oikeille manuaalin sivuille on lisätty tarvittaviin kappaleisiin.

10 Partikkeliefektit

10.1 Mitä partikkeliefektit ovat?

Suurin osa nykyaikana käytettävistä pelimoottoreista sisältää valmiit työkalut partikkeliefektien toteuttamiseen, joita kutsutaan usein partikkelijärjestelmiksi. Partikkelijärjestelmien avulla pienistä, simppeleistä kuvista tai mesheistä voidaan muodostaa komplekseja efektejä, kuten luoda vaikutelma virtaavasta nesteestä tai kaasusta. Esimerkiksi savupilveä tehtäessä jokaisella partikkelilla olisi pieni savutekstuuri, joka edustaa itsessään pientä pilveä. Kun näitä pieniä pilviä yhdistetään paljon samalle alueelle, niistä syntyy kokonaisen ja isomman pilven vaikutelma. Näyttävämmät partikkeliefektit muodostuvat yleensä useammasta

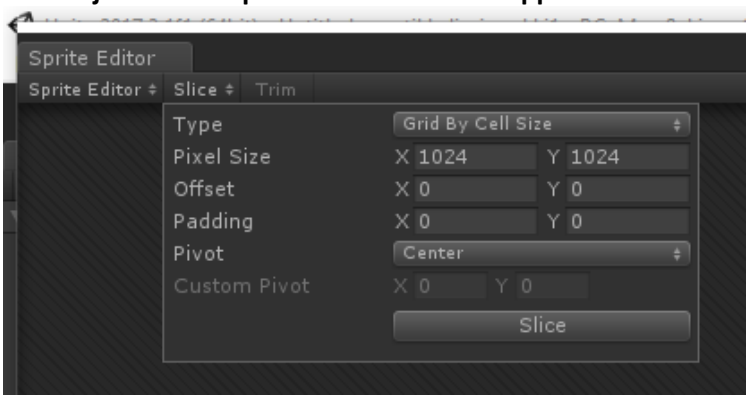
päällekkäisestä efektistä. Esimerkiksi räjähdysefektissä voitaisiin käyttää yhtä partikkelijärjestelmää tuottamaan, savua, toinen shokkiaaltoa ja kolmas kipinöitä.

10.2 Esimerkki – Moniosainen partikkeliefekti

Tässä kappaleessa toteutetaan neljästä osasta koostuva partikkeliefekti. Prosessi käydään vaihe vaiheelta läpi.

10.2.1 Valmistelu

1. **Lataa tekstuuritiedosto osoitteesta https://drive.google.com/file/d/1E3XYd-Wqopj8cPBxQPUs7rFS9KZRcZWj/view?usp=sharing**
2. **Luo uusi Unity-projekti valitsemallasi nimellä ja valitse työtilaksi 2D.**
3. **Vedä aikaisemmin lataamasi particlesheet-tekstuuritiedosto Unityn Assets-näkymään.**
4. **Napsauta particlesheet-tekstuuritiedostoa, Unityn Inspectoriin tulee näkyviin tekstuurin Import-asetukset. Muuta Sprite Mode Multipleksi ja paina alaoikealla näkyvää Apply-näppäintä ja jatka painamalla Sprite Editor -näppäintä.**
5. **Sprite Editor-näkymässä, paina ylävasemmalla näkyvää Slice-näppäintä ja muuta Type-kohtaan Grid By Cell Size. Sen jälkeen muuta Pixel Size X- ja Y-arvoiksi 1024. Tämän jälkeen voit painaa valikon Slice-näppäintä.**



Kuvio 11 Muuta Slice-asetukset seuraavanlaisiksi ja paina Slice-näppäintä

6. **Sprite Editor-ikkunassa, paina yläoikealla näkyvää Apply-näppäintä. Voit nyt sulkea Sprite Editorin.**

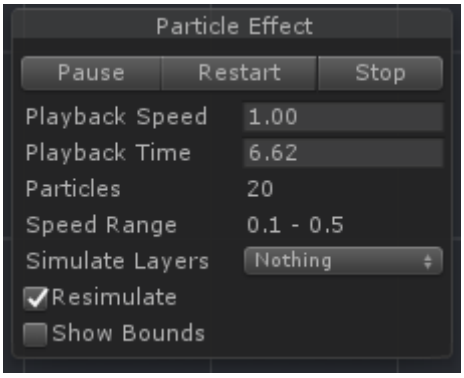
7. **Valitse ylävalikosta Assets -> Create -> Material. Assets-kansioon ilmestyy uusi Material-tiedosto, jonka voit nimetä haluamallasi tavalla.**

8. **Napsauta luomaasi Materiaalia. Inspector-ikkunassa, valitse Shader -> Particles -> Additive. Vedä aikaisemmin luomasi particlesheet-tekstuuritiedosto Inspectorissa näkyvään Particle Texture-laatikkoon.**

9. **Nyt sinulla on yhtä tekstuuria käyttävä materiaali, jota voit käyttää useassa eri partikkeliefektissä!**

10.2.2 Ensimmäinen osa - Savu

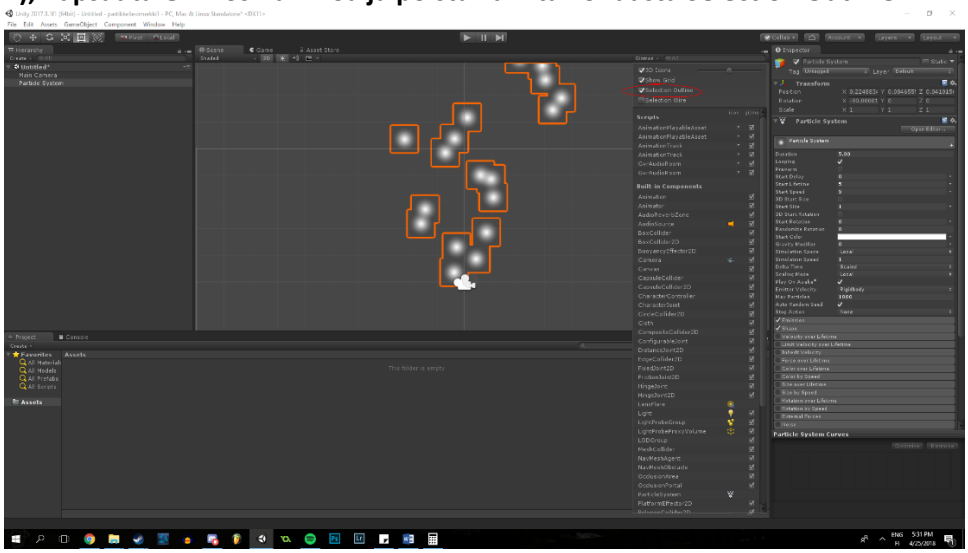
Partikkeliefektin ensimmäinen osa on pieni muotoaan muuttava savupilvi. Savu toistuu efektin taustalla ja nousee hieman ylöspäin.



Kuvio 12 Jos partikkeliefektisi ei jostain syystä päivity, tai se käyttäytyy muuten oudolla tavalla, napsauta partikkeliefektiästi Hierarchy-ikkunassa ja paina Restart-nappulaa Scene-näkymään ilmestyvästä Particle Effect-ikkunasta.

1. **Kun olet tyhjässä scenenäkymässä, valitse ylävalikosta GameObject -> Effects -> Particle System. Scenessä pitäisi näkyä nyt partikkeleja, jotka tulevat partikkelijärjestelmästä. Jos partikkelien ympärillä näkyy oranssi laatikko (ks. kuvio**

1), napsauta Gizmos-valikkoa ja poista valinta kohdasta Selection Outline.



Kuvio 13 Gizmos-valikon näkymä

2. **Napsauta Particle System-objektia Hierarchy-näkymässä. Inspectoriin tulee näkyviin partikkelijärjestelmän ominaisuudet moduuleina.**

3. **Napsauta alimmaisena Inspectorin listassa näkyvää Renderer-moduulia ja vedä aikaisemmin luomasi partikkelimateriaali Material-valikon päälle. Default-Particle-materiaalin tilalla pitäisi näkyä nyt partikkelimateriaali.**

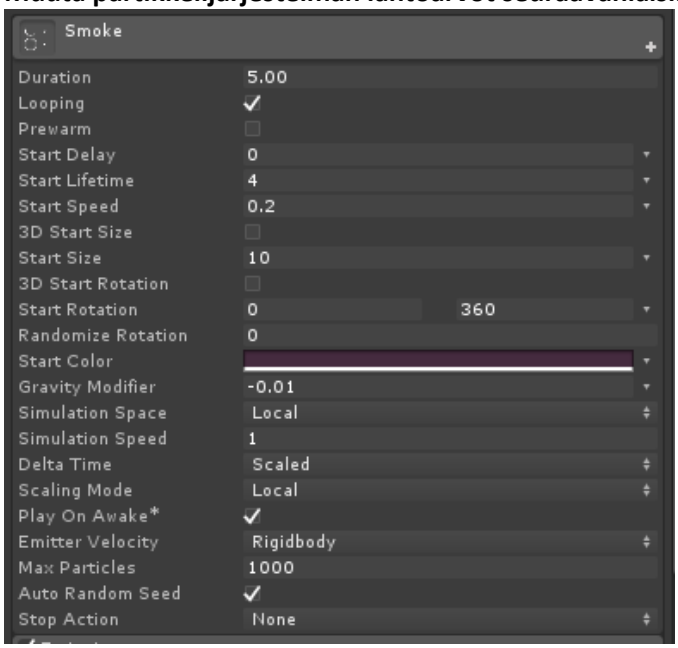
4. **Napsauta Texture Sheet Animation-moduulin vasemmassa reunassa olevaa harmaata laatikkoa aktivoidaksesi sen. Sen jälkeen napsauta Texture Sheet Animation-moduuli auki.**

5. **Muuta Texture Sheet Animation-moduulin Tiles-arvot lukuihin 2 ja 2.**

6. **Muuta Texture Sheet Animation-moduulin Frame over Time-arvo Constantiksi painamalla sen oikealla puolella sijaitsevaa harmaata nuolta. Lukuarvoa**

muuttamalla 0-3 välillä voimme valita mitä tekstuuria partikkelit käyttävät. Muuta arvoksi 1. Partikkelit käyttävät nyt savupilven tekstuuria.

7. **Muuta partikkelijärjestelmän lähtöarvot seuraavanlaisiksi:**



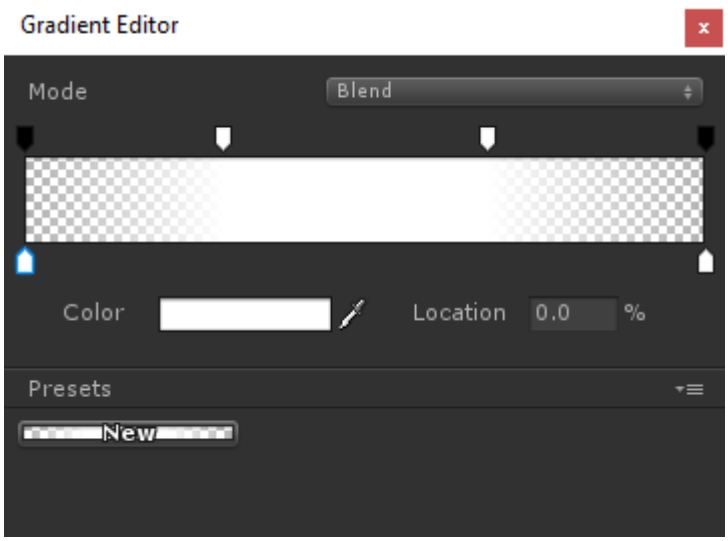
Kuvio 14 Lähtöarvoilla savupilvelle annetaan pientä liikettä ja väriä, Start Color voi olla mikä tahansa haluamasi väri.

8. **Muuta Emission-moduulin Rate over Time-arvoksi 5.**

9. **Muuta Shape-moduulin muoto Sphereksi ja anna sille Radius-arvoksi 0.3.**

10. **Aktivoi Color over Lifetime-moduuli. Napsauta Color-tekstin vieressä olevaa valkeaa laatikkoa. Näkyviin ilmestyy Gradient Editor.**

11. **Muuta Gradient Editorin arvoja seuraavasti:**

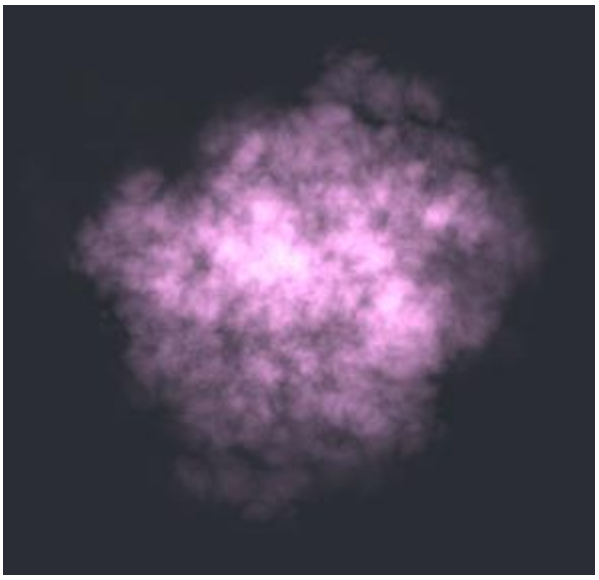


Kuvio 15 Napsauttamalla väripalkin yläreunan päältä tehdäksesi kaksi uutta Gradient-pistettä kuvassa näkyviin kohtiin. Aseta vasemmalla ja oikealla näkyvien mustien Gradient-pisteiden Alpha-arvoksi 0 ja keskimmäisten pisteiden Alpha-arvoksi 255.

12. **Aktivoi Rotation over Lifetime-moduuli ja muuta Angular Velocity tilaan Random Between Two Constants. Muuta arvoiksi 0 ja 5.**

13. **Muuta Renderer-moduulin Max Particle Size vastaamaan Start Size-lähtöarvoamme, joka on 10.**

14. **Efektin savuosuus on nyt valmis!**



10.2.3 Toinen osa - Pyörivä savu

Tässä osuudessa olemassa olevan savupilven reunoille toteutetaan pyörivää liikettä tekevä savuvana.

1. **Napsauta Hierarchy-ikkunassa edellisessä osuudessa tehtyä partikkelijärjestelmää oikealla hiiren painikkeella. Valitse Effects -> Particle System.**

2. **Edellisen partikkelijärjestelmän alaisuudessa on nyt toinen partikkelijärjestelmä. Nimeä se haluamallasi tavalla.**

3. **Napsauta Particle System-objektia Hierarchy-näkymässä. Inspectoriin tulee näkyviin partikkelijärjestelmän ominaisuudet moduuleina.**

4. **Napsauta alimmaisena Inspectorin listassa näkyvää Renderer-moduulia ja vedä aikaisemmin luomasi partikkelimateriaali Material-valikon päälle. Default-Particle-materiaalin tilalla pitäisi näkyä nyt partikkelimateriaali.**

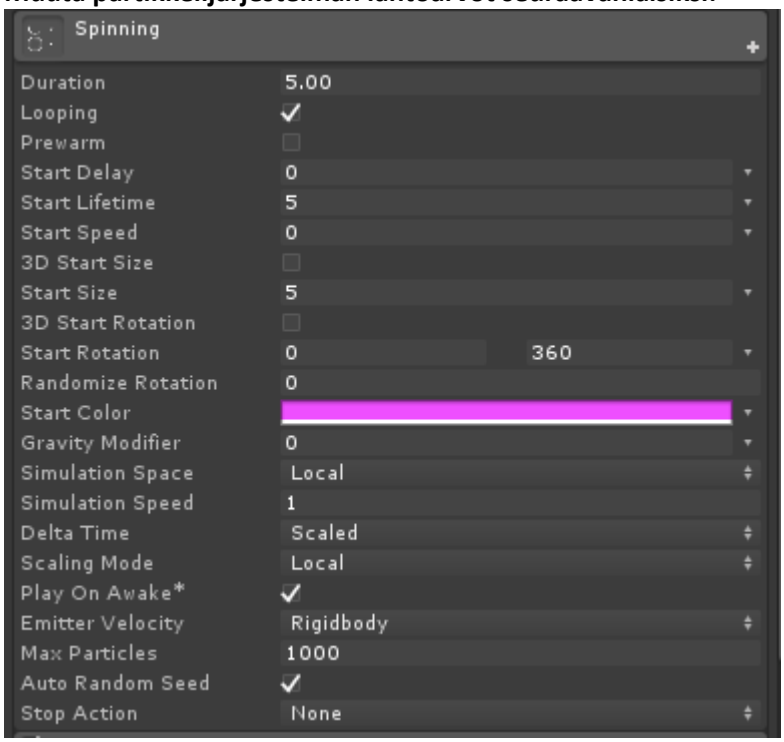
5. **Napsauta Texture Sheet Animation-moduulin vasemmassa reunassa olevaa harmaata laatikkoa aktivoidaksesi sen. Sen jälkeen napsauta Texture Sheet Animation-moduuli auki.**

6. **Muuta Texture Sheet Animation-moduulin Tiles-arvot lukuihin 2 ja 2.**

7. **Muuta Texture Sheet Animation-moduulin Frame over Time-arvo Constantiksi painamalla sen oikealla puolella sijaitsevaa harmaata nuolta. Lukuarvoa**

muuttamalla 0-3 välillä voimme valita mitä tekstuuria partikkelit käyttävät. Muuta arvoksi 0. Partikkelit käyttävät nyt pyörivän savun tekstuuria.

8. **Muuta partikkelijärjestelmän lähtöarvot seuraavanlaisiksi:**



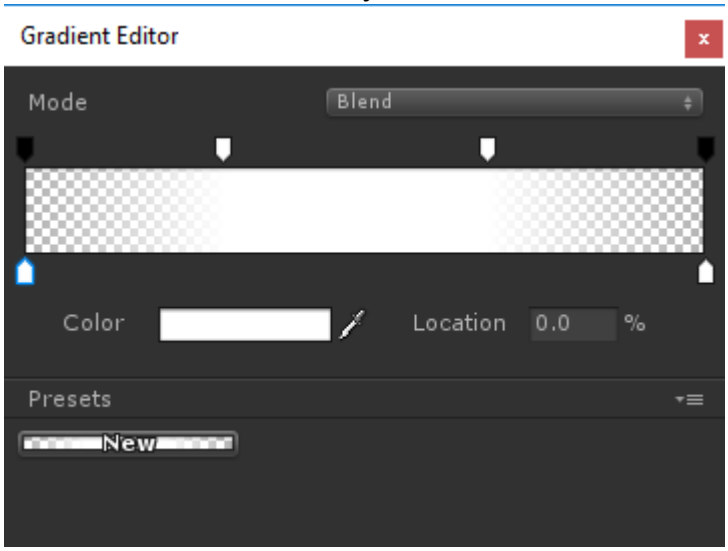
Kuvio 16 Lähtöarvoilla partikkelit pysäytetään kokonaan ja niille annetaan väriä. Start Color voi olla mikä tahansa haluamasi väri.

9. **Muuta Emission-moduulin Rate over Time-arvoksi 2.**

10. **Muuta Shape-moduulin muodoksi Box ja anna sen X-Scale-arvoksi 0.3 ja Y-Scale-arvoksi 0.3.**

11. **Aktivoi Color over Lifetime-moduuli. Napsauta Color-tekstin vieressä olevaa valkeaa laatikkoa. Näkyviin ilmestyy Gradient Editor.**

12. **Muuta Gradient Editorin arvoja seuraavasti:**

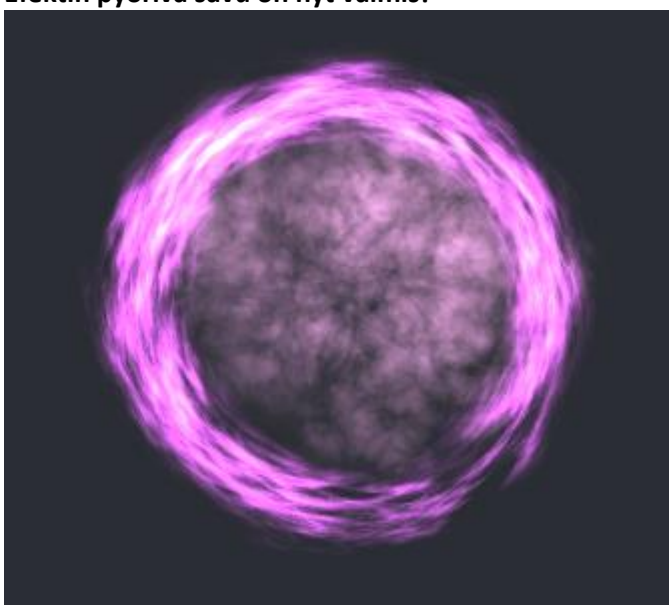


Kuvio 17 Napsauttamalla väripalkin yläreunan päältä tehdäksesi kaksi uutta Gradient-pistettä kuvassa näkyviin kohtiin. Aseta vasemmalla ja oikealla näkyvien mustien Gradient-pisteiden Alpha-arvoksi 0 ja keskimmäisten pisteiden Alpha-arvoksi 255.

13. **Aktivoi Rotation over Lifetime-moduuli ja muuta Angular Velocity tilaan Random Between Two Constants. Muuta arvoiksi -500 ja 500.**

14. **Muuta Renderer-moduulin Max Particle Size vastaamaan Start Size-lähtöarvoamme, joka on 5.**

15. **Efektin pyörivä savu on nyt valmis!**



10.2.4 Kolmas osa - Ympyrät

Tässä osuudessa efektin keskiosaan lisätään ympyrän muotoisia partikkeleja.

1. **Napsauta Hierarchy-ikkunassa ensimmäisessä osuudessa tehtyä partikkelijärjestelmää oikealla hiiren painikkeella. Valitse Effects -> Particle System.**

2. **Ednsimmäisen partikkelijärjestelmän alaisuudessa on nyt uusi partikkelijärjestelmä. Nimeä se haluamallasi tavalla.**

3. **Napsauta Particle System-objektia Hierarchy-näkymässä. Inspectoriin tulee näkyviin partikkelijärjestelmän ominaisuudet moduuleina.**

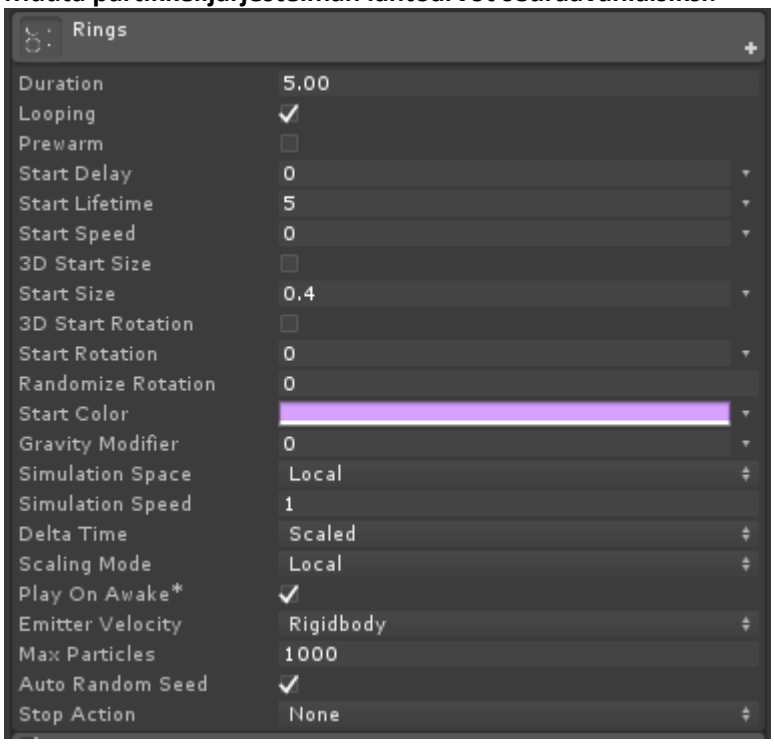
4. **Napsauta alimmaisena Inspectorin listassa näkyvää Renderer-moduulia ja vedä aikaisemmin luomasi partikkelimateriaali Material-valikon päälle. Default-Particle-materiaalin tilalla pitäisi näkyä nyt partikkelimateriaali.**

5. **Napsauta Texture Sheet Animation-moduulin vasemmassa reunassa olevaa harmaata laatikkoa aktivoidaksesi sen. Sen jälkeen napsauta Texture Sheet Animation-moduuli auki.**

6. **Muuta Texture Sheet Animation-moduulin Tiles-arvot lukuihin 2 ja 2.**

7. **Muuta Texture Sheet Animation-moduulin Frame over Time-arvo Constantiksi painamalla sen oikealla puolella sijaitsevaa harmaata nuolta. Lukuarvoa muuttamalla 0-3 välillä voimme valita mitä tekstuuria partikkelit käyttävät. Muuta arvoksi 2. Partikkelit käyttävät nyt pyörivän savun tekstuuria.**

8. **Muuta partikkelijärjestelmän lähtöarvot seuraavanlaisiksi:**



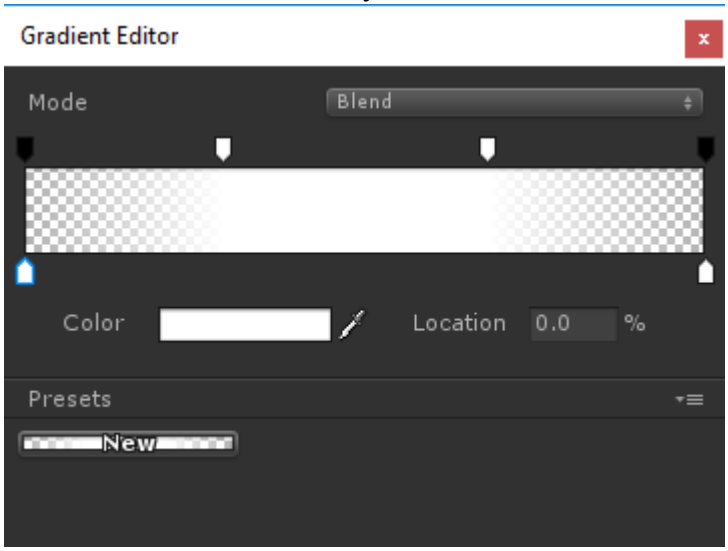
Kuvio 18 Lähtöarvoilla partikkelit pysäytetään kokonaan ja niille annetaan väriä. Start Color voi olla mikä tahansa haluamasi väri.

9. **Muuta Emission-moduulin Rate over Time-arvoksi 7.**

10. **Muuta Shape-moduulin muodoksi Sphere ja anna sille Radius-arvoksi 0.01.**

11. **Aktivoi Color over Lifetime-moduuli. Napsauta Color-tekstin vieressä olevaa valkeaa laatikkoa. Näkyviin ilmestyy Gradient Editor.**

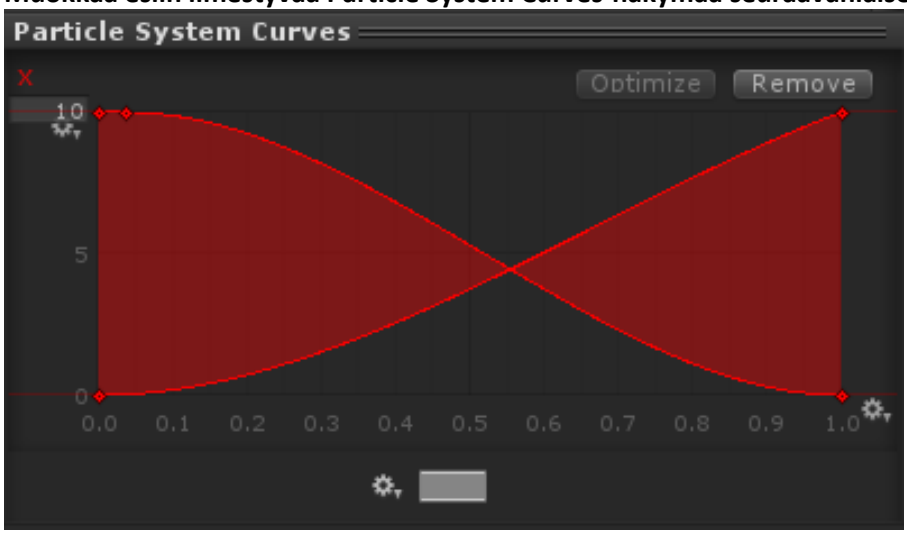
12. **Muuta Gradient Editorin arvoja seuraavasti:**



Kuvio 19 Napsauttamalla väripalkin yläreunan päältä tehdäksesi kaksi uutta Gradient-pistettä kuvassa näkyviin kohtiin. Aseta vasemmalla ja oikealla näkyvien mustien Gradient-pisteiden Alpha-arvoksi 0 ja keskimmäisten pisteiden Alpha-arvoksi 255.

13. **Aktivoi Size over Lifetime-moduuli ja muuta Size-tyypiksi Random Between Two Curves.**

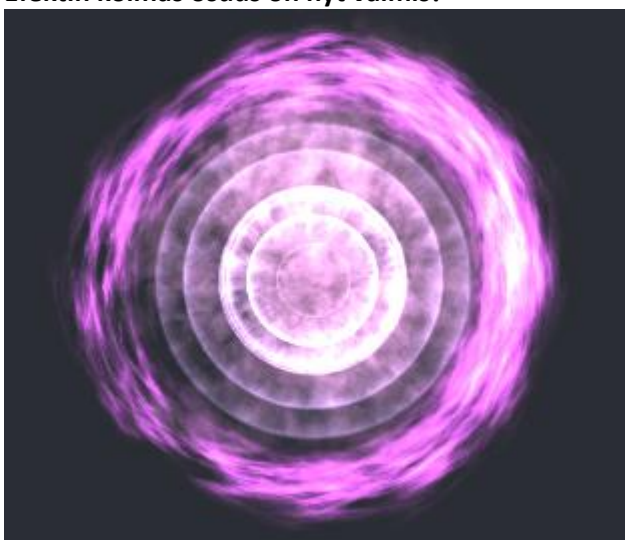
14. **Muokkaa esiin ilmestyvää Particle System Curves-näkymää seuraavanlaisesti:**



Kuvio 20 Muokkaa kurvit kuvassa näkyvällä tavalla. Vasemmalla ylälaidassa näkyvä numero on koon maksimiarvo. Kirjoita siihen 10.

15. **Muuta Renderer-moduulin Max Particle Size vastaamaan Start Size-lähtöarvoamme, joka on 0.4.**

16. **Efektin kolmas osuus on nyt valmis!**



10.2.5 Neljäs osa – Kipinät

Tässä osuudessa partikkeliefekti viimeistellään lisäämällä siihen sen keskeltä lentäviä kipinöitä.

1. **Napsauta Hierarchy-ikkunassa ensimmäisessä osuudessa tehtyä partikkelijärjestelmää oikealla hiiren painikkeella. Valitse Effects -> Particle System.**

2. **Ednsimmäisen partikkelijärjestelmän alaisuudessa on nyt uusi partikkelijärjestelmä. Nimeä se haluamallasi tavalla.**

3. **Napsauta Particle System-objektia Hierarchy-näkymässä. Inspectoriin tulee näkyviin partikkelijärjestelmän ominaisuudet moduuleina.**

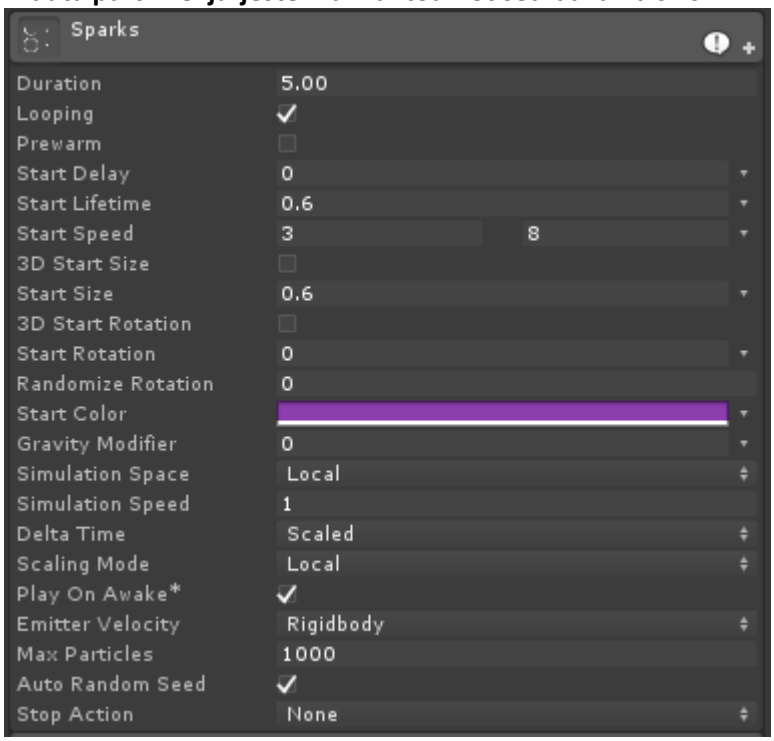
4. **Napsauta alimmaisena Inspectorin listassa näkyvää Renderer-moduulia ja vedä aikaisemmin luomasi partikkelimateriaali Material-valikon päälle. Default-Particle-materiaalin tilalla pitäisi näkyä nyt partikkelimateriaali.**

5. **Napsauta Texture Sheet Animation-moduulin vasemmassa reunassa olevaa harmaata laatikkoa aktivoidaksesi sen. Sen jälkeen napsauta Texture Sheet Animation-moduuli auki.**

6. **Muuta Texture Sheet Animation-moduulin Tiles-arvot lukuihin 2 ja 2.**

7. **Muuta Texture Sheet Animation-moduulin Frame over Time-arvo Constantiksi painamalla sen oikealla puolella sijaitsevaa harmaata nuolta. Lukuarvoa muuttamalla 0-3 välillä voimme valita mitä tekstuuria partikkelit käyttävät. Muuta arvoksi 3. Partikkelit käyttävät nyt kipinöiden tekstuuria.**

8. **Muuta partikkelijärjestelmän lähtöarvot seuraavanlaisiksi:**



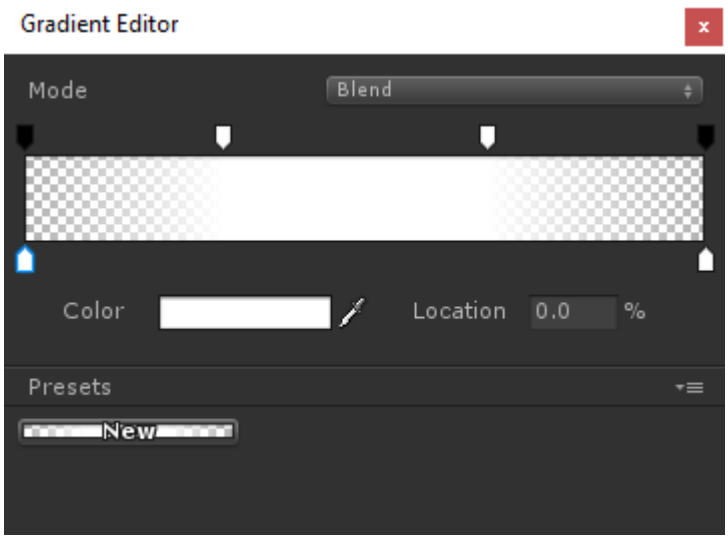
Kuvio 21 Lähtöarvoilla partikkeleille annetaan liikettä ja väriä. Start Color voi olla mikä tahansa haluamasi väri.

9. **Muuta Emission-moduulin Rate over Time-arvoksi 25.**

10. **Muuta Shape-moduulin muodoksi Sphere ja anna sen Radius-arvoksi 2.2.**

11. **Aktivoi Color over Lifetime-moduuli. Napsauta Color-tekstin vieressä olevaa valkeaa laatikkoa. Näkyviin ilmestyy Gradient Editor.**

12. **Muuta Gradient Editorin arvoja seuraavasti:**



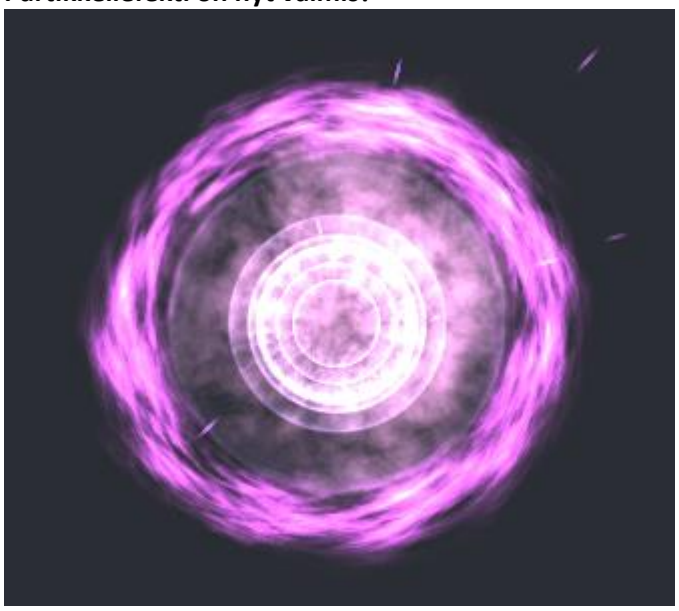
Kuvio 22 Napsauttamalla väripalkin yläreunan päältä tehdäksesi kaksi uutta Gradient-pistettä kuvassa näkyviin kohtiin. Aseta vasemmalla ja oikealla näkyvien mustien Gradient-pisteiden Alpha-arvoksi 0 ja keskimmäisten pisteiden Alpha-arvoksi 255.

13. **Aktivoi Noise-moduuli ja aseta sen Strength-arvoksi 0.2 ja Frequency-arvoksi 0.5. Partikkelit saavat lentorataansa pientä liikettä.**

14. **Muuta Renderer-moduulin Render Modeksi Stretched Billboard ja anna sille seuraavat arvot: Camera Scale: 0, Speed Scale: 0.2, Length Scale: 1. Partikkelit venyvät nyt riippuen niiden nopeudesta.**

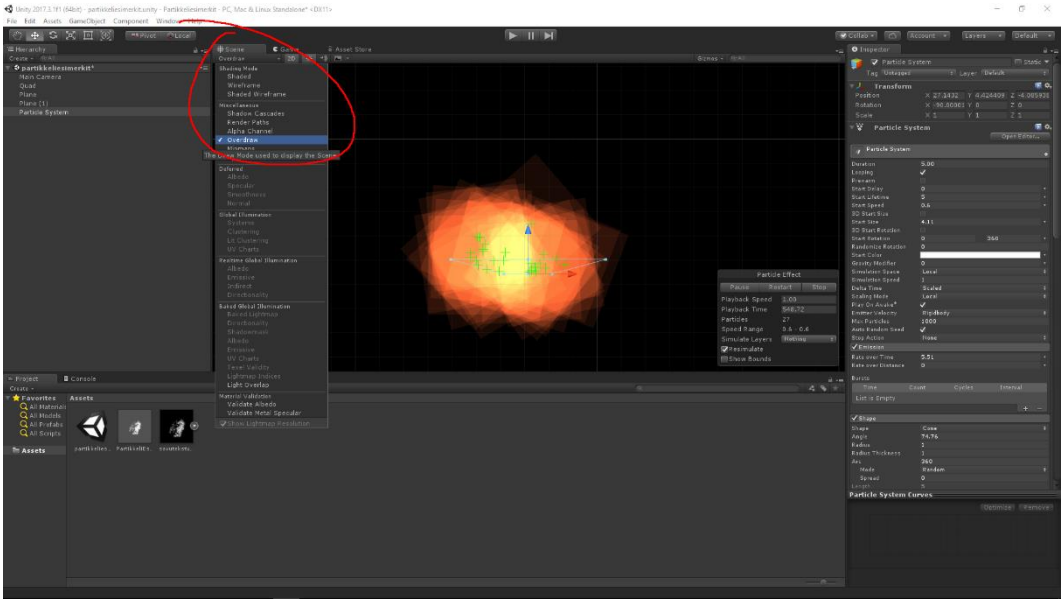
15. **Muuta Renderer-moduulin Max Particle Size vastaamaan Start Size-lähtöarvoamme, joka on 0.6.**

16. **Partikkeliefekti on nyt valmis!**



10.3 Partikkeliefektien optimointi

Väärin käytettynä, partikkeliefektit voivat olla raskaita ja viedä paljon prosessointivoimaa laitteelta, jolla niitä pyöritetään. Varsinkin mobiilikehityksessä on tärkeää optimoida toteutetut partikkeliefektit, sillä ne vaikuttavat suoraan esimerkiksi puhelimen akun kulutukseen.



Kuvio 23 Unityn Overdraw -näkymä

Yksi merkittävistä partikkeliefektien suorituskykyyn vaikuttavista tekijöistä on ylitsepiirto. Tämä tarkoittaa sitä, miten partikkelit piirtyvät näytölle kun niitä on paljon päällekäin. Läpinäkyvyyttä sisältävät partikkelit joudutaan piirtämään aina uudestaan, jos niiden alla on lisää partikkeleita tai muita objekteja. Unityn Scene -ikkunan Overdraw -näkymästä voi tarkastella, kuinka paljon ylitsepiirtoa scenessä esiintyy. Kirkkaammat alueet yllä olevan esimerkin efektissä ovat kalliimpia renderöidä, kuin reunalla näkyvät tummemmat alueet. Läpinäkyvyyttä sisältäviä partikkeleja käytettäessä on siis hyvä asettaa mahdollisimman vähän partikkeleita päällekäin.

Partikkeliefekteissä käytettävät tekstuurit kannattaa asetella sprite sheettiin, joka on kokoelma useita tekstuureita. Näin tietokone joutuu lataamaan vähemmän tekstuureita partikkeliefektiä renderöidessä, ja on täten tehokkaampi. Katso oheinen linkki sprite sheettien toteuttamisesta:

https://www.codeandweb.com/texturepacker/tutorials/how-to-create-a-sprite-sheet.

Gabriel Aguiar kertoo videollaan lisää hyviä vinkkejä partikkeliefektien optimoimiseen: https://youtu.be/JqWvJK1uFn8

11 Shaderit

Tässä luvussa esitellään shaderit ja kerrotaan lyhyesti kuinka niitä voidaan käyttää Unityssä.

11.1 Mitä shaderit ovat?

Shaderit ovat yksinkertaisuudessaan pieniä ohjelmia, jotka toimivat suoraan tietokoneen näytönohjaimessa, jättäen suorittimelle vähemmän prosessoitavaa. Shaderien avulla voidaan manipuloida näytölle piirtyvää kuvaa esimerkiksi vaihtamalla pikseleitä eri värisiksi tai muuttamalla verteksien sisältämää tietoa. Shaderien avulla voidaan tehdä todella monimutkaisia efektejä ja niiden kirjoittaminen Unityssä voi olla melko hankalaa, sillä niitä kirjoitetaan C# sijaan Cg/HLSL kielillä, jotka ovat hyvin lähellä C -kieltä. Unityn tulevassa 2018.1 -versiossa shadereita on mahdollista tehdä visuaalisella Shader Graph -nimisellä node-editorilla, jonka kaltaisia työkaluja on saatavilla Unityn Asset Storesta ja kilpailevien pelimoottoreiden työkaluvalikoimista.

Kuvio 24 Esimerkki shaderin avulla tehdystä efektistä (Lähde: https://wpfwonderland.wordpress.com/2008/10/06/wpf-shader-effects-library-posted/)

11.2 Esimerkki

Tässä luvussa tehdään vaihe vaiheelta ruutua punaisena vilkuttava jälkikäsittelyshader. Esimerkkitehtävän avulla ymmärrät kuinka yksinkertainen shader tehdään js miten shadereita käytetään materiaalien avulla. Katso oheiset Unity-manuaalin sivut: https://unity3d.com/learn/tutorials/topics/graphics/gentle-introduction-shaders & https://docs.unity3d.com/Manual/SL-VertexFragmentShaderExamples.html.

1. **Luo uusi Unityprojekti ja nimeä se haluamallasi tavalla. Aseta työtilaksi 2D.**
2. **Haluamme tehdä koko ruutua välkyttävän efektin ja liittää sen skenessä valmiiksi olevaan kameraan. Koska kameraan itsessään ei voi asettaa materiaalia, teemme scriptin, jonka avulla voimme asettaa shaderin toimimaan koko ruudun alueella.**
3. **Luodaan uusi C# -script napsauttamalla ylävalikosta Assets -> Create -> C# Script. Anna sille nimeksi SimpleBlit.**
4. **Avaa SimpleBlit-script MonoDevelop-editorissa.**

5. **Editoi script seuraavanlaiseksi:**

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

public class SimpleBlit : MonoBehaviour {

    // Alustetaan materiaali, jonka voimme asettaa Unityn Inspector-näkymässä.
    public Material mat;

    // OnRenderImagella voimme muokata näytölle renderöitävää kuvaa, ennen kuin se piirtyy näytölle.
    void OnRenderImage(RenderTexture src, RenderTexture dest) {

        // src on normaalisti näytölle suoraan lähetettävä renderöity kuva.
        // dest on Render Texture, johon lopullinen muokattu kuva renderöidään.

        // Graphics.Blitillä kopioidaan source-tekstuuri ja lähetetään se destination-tekstuurille shaderin kanssa.
        Graphics.Blit(src, dest, mat);

    }

}

```

Kuvio 25 SimpleBlit-scripti kommentoituna.

6. **Tämän scriptin avulla voimme antaa kameralle tiedon shaderista, jolla haluamme muokata ruudulle renderöityvää kuvaa.**

7. **Tallenna tekemäsi script.**

8. **Napsauta Main Camera-objektia Unityn Hierarchy-ikkunassa.**

9. **Inspector-ikkunassa, lisää uusi komponentti napsauttamalla Add Component-painiketta.**

10. **Kirjoita hakukenttään tekemäsi scriptin nimi ”SimpleBlit”.**

11. **Napsauta hakuun ilmestyvää SimpleBlit-scriptiä.**

12. **Script on nyt kiinni kamerassa. Älä anna sille vielä materiaalia.**

13. **Luo uusi Image Effect Shader napsauttamalla ylävalikosta Assets -> Create -> Shader -> Image Effect Shader. Nimeä se haluamallasi tavalla. Image Effect Shader on Unityn valmis pohja jälkikäsittelyshadereita varten.**

14. **Avaa luomasi shader MonoDevelop-editorissa.**

15. **Tässä koko shader kommentoituna:**

```
// Tekemämme Shader ei näy materiaalien Shader-valinnassa, jos se on nimetty Hidden-kategoriaan.
// Muutetaan Hiddenin tilalle Custom, jotta se tulee näkyviin käyttämässämme materiaalissa.
Shader "Custom/TestShader"
{
    // Properties sisältää Shaderin ominaisuudet, jotka tulevat näkyviin käytettävän materiaalin Inspector-näkymässä.
    // Image Effect Shader -pohja sisältää vakiona vain tekstuurin.
    Properties
    {
        _MainTex ("Texture", 2D) = "white" {}
    }
    SubShader
    {
        // No culling or depth
        Cull Off ZWrite Off ZTest Always

        Pass
        {
            CGPROGRAM
            #pragma vertex vert
            #pragma fragment frag

            #include "UnityCG.cginc"

            struct appdata
            {
                float4 vertex : POSITION;
                float2 uv : TEXCOORD0;
            };

            struct v2f
            {
                float2 uv : TEXCOORD0;
                float4 vertex : SV_POSITION;
            };

            // Tämä on shaderin vertex-osuus!
            // Vertex shaderia kutsutaan kerran yhden framen aikana jokaista verteksiä kohden, johon tämä shader vaikuttaa.
            // Esimerkiksi Unityn Cube-komponentissa vertex shaderia kutsuttaisiin 8 kertaa jokaista framea kohden.
            // Vertex shaderin avulla voimme vaikuttaa verteksien ominaisuuksiin, kuten liikuttaa niitä halutulla tavalla.
            v2f vert (appdata v)
            {
                v2f o;
                o.vertex = UnityObjectToClipPos(v.vertex);
                o.uv = v.uv;
                return o;
            }

            // _MainTex merkitsee tässä shaderissa koko ruutuamme.
            sampler2D _MainTex;

            // Tämä on shaderin fragment-osuus!
            // Fragment shaderia kutsutaan kerran yhden framen aikana jokaista pikseliä kohden, jolla alueella tämä shader vaikuttaa.
            // Esimerkiksi 1920x1080 resoluutiolla koko ruudun peittävä fragment shader kutsuttaisiin 2,073,600 kertaa yhden framen aikana.
            // Monien koko ruudun peittävien efektien yhtäaikainen käyttö voi olla raskasta!
            // Fragment shaderin avulla voimme vaikuttaa jokaiseen sen alueella sijaitsevaan pikseliin.
            fixed4 frag (v2f i) : SV_Target
            {
                // Tällä rivillä _MainTex -tekstuurilta kysytään, minkälainen pikseli on missäkin koordinaatissa ja tietoa voidaan muokata.
                fixed4 col = tex2D(_MainTex, i.uv);
                // just invert the colors
                col.rgb = 1 - col.rgb;
                return col;
            }
            ENDCG
        }
    }
```

Kuvio 26 Unityssä shaderit kirjoitetaan HLSL-kielen variaatiolla, jota kutsutaan Cg:ksi.

16. **Ensimmäisellä rivillä, muuta Shaderin kategoria Hiddenistä Customiksi, tai muuhun haluamaasi kategoriaan:**

```
// Tekemämme Shader ei näy materiaalien Shader-valinnassa, jos se on nimetty Hidden-kategoriaan.
// Muutetaan Hiddenin tilalle Custom, jotta se tulee näkyviin käyttämässämme materiaalissa.
Shader "Custom/TestShader"
```

17. **Image Effect Shader muuttaa oletuksena ruudun värit käänteiseksi fragment shaderissa. Poista tämä rivi shaderista:**

```
42          fixed4 frag (v2f i) : SV_Target
43          {
44              fixed4 col = tex2D(_MainTex, i.uv);
45              // just invert the colors
46              col.rgb = 1 - col.rgb;
47              return col;
48          }
```

Kuvio 27 Poista sinisellä yliviivattu osuus Image Effect Shaderista.

18. **Muokkaamalla shaderin fragment-osuutta, voit vaikuttaa ruudulle renderöityviin pikseleihin. Lisätään ruudun punaisuutta ja lisätään siihen sin-funktio, joka muuttuu ajan myötä.**

```
            fixed4 frag (v2f i) : SV_Target
            {
                // Tällä rivillä _MainTex -tekstuurilta kysytään, minkälainen pikseli on missäkin koordinaatissa ja tietoa voidaan muokata.
                fixed4 col = tex2D(_MainTex, i.uv);

                // Ruudun punaista väriä lisätään ja muutetaan värin voimakkuutta sin-funktion avulla. Ruutu vilkkuu nyt punaisena!
                col.r = 1 + sin(_Time[3]);

                return col;
            }
            ENDCG
        }
    }
}
```

Kuvio 28 Valmis, ruutua vilkuttava fragment shader.

19. **Tallenna shader.**

20. **Tekemäsi shader ei tee vielä mitään, sillä sitä pitää käyttää materiaalin kautta ja antaa se tekemällemme SimpleBlit-scriptille.**

21. **Palaa Unityyn ja luo uusi materiaali napsauttamalla ylävalikosta Assets -> Create -> Material. Nimeä se haluamallasi tavalla.**

22. **Napsauta juuri luomaasi materiaalia Unityn Assets-kansiossa.**

23. **Inspector-ikkunaan ilmestyvässä materiaalissa, valitse Shader-listasta tekemäsi shader. Sen pitäisi löytyä shaderissa luomasi Custom-kategorian alta.**

24. **Napsauta Main Camera-objektia Hierarchy-ikkunassa.**

25. **Anna Inspector-ikkunassa näkyvän Main Camera-objektin Simple Blit -scriptille tekemäsi materiaali kohtaan Mat.**

26. **Napsauta Unityn Play-painiketta.**

27. **Ruutu vilkkuu nyt punaisena!**

11.3 Optimointi

Fragment shadereita kirjoittaessa tulee kiinnittää huomiota ylitsepiirtoon ja suoritettujen prosessien määrään, sillä se kutsutaan kerran yhden framen aikana jokaista näytöllä näkyvää pikseliä kohden. 1920x1080 resoluutioisella näytöllä tämä tarkoittaa siis 2 073 600 laskutoimitusta framessa ja 124 416 000 laskutoimitusta sekunnissa. Monet yhtä aikaa koko ruudulle piirrettävät ovat raskaita, ja niitä tulee käyttää harkiten.

Lisää artikkeleita shadereiden optimoinnista:

https://unity3d.com/how-to/shader-profiling-and-optimization-tips

https://docs.unity3d.com/Manual/SL-ShaderPerformance.html