Osaamista
ja oivallusta
tulevaisuuden
tekemiseen

Tanja Lamberg

Partikkeli- ja elementtitehosteet seikkailupelissä

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tieto- ja viestintätekniikka

Insinöörityö

16.4.2019

Tekijä Otsikko Sivumäärä Aika	Tanja Lamberg Partikkeli- ja elementtitehosteet seikkailupelissä 47 sivua 16.4.2019
Tutkinto	Insinööri (AMK)
Tutkinto-ohjelma	Tieto- ja viestintätekniikka
Ammatillinen pääaine	Pelisovellukset
Ohjaaja	Lehtori Antti Laiho

Insinöörityön tarkoituksena oli toteuttaa erilaisia partikkeli- ja elementtitehosteita Frog Shaman -nimiseen 3D-seikkailupeliin. Tavoitteena oli toteuttaa mahdollisimman moni pelissä käytettävästä 20 loitsusta. Tehosteet toteutettiin Unity-pelimoottorin partikkelijärjestelmällä ja varjostimilla.

Insinöörityössä perehdyttiin syvemmin partikkelijärjestelmän ja varjostimien historiaan sekä niiden käyttöön Unity-pelimoottorissa. Samalla tutustuttiin myös viiteen eri elementtiin, joita pelissä käytetään loitsujen muodostamiseen. Näitä elementtejä olivat tuli, ilma, jää, maa ja sähkö. Jokainen toteutettu loitsutehoste kuvasti jollain tapaa elementtejä, joita siihen yhdistettiin.

Työtä tehdessä löydettiin myös hyviä käytänteitä, joita tehosteiden toteutuksessa tulisi ottaa huomioon. Tehosteiden pitäisi olla ennen kaikkea sellaisia, että ne kommunikoivat pelin tapahtumia pelaajalle riittävän selkeästi, jotta pelaajat ymmärtävät ne oikein. Tehosteiden olisi myös hyvä olla viihdyttäviä, jotta pelaajat jaksavat katsoa niitä.

Insinöörityön aikana ehdittiin toteuttaa yhteensä 12 eri loitsutehostetta. Tehosteet toteutettiin suunnitelmien mukaan sellaisiksi, että niitä voidaan sellaisenaan pelissä käyttää. Tehosteiden toteutuksessa käytettiin monia erilaisia partikkelitehosteita ja muutamia elementtitehosteita. Tehosteisiin lisättiin lisäksi toiminnallisuuksia, joita pelissä tulevaisuudessa tarvitaan.

Useimmat loitsuista saatiin valmiiksi asti, mutta osasta jäi vielä uupumaan ominaisuuksia, jotka joudutaan tulevaisuudessa toteuttamaan. Tehosteet toteutettiin kuitenkin niin, että pelin ulkoasun kehittyessä tehosteiden ulkonäköä pystytään helposti muuttamaan.

Avainsanat	partikkeli, varjostin, tehoste, Unity

Author	Tanja Lamberg
Title	Particle and Shader Effects in an Adventure Game
Number of Pages	47 pages
Date	16 April 2019
Degree	Bachelor of Engineering
Degree Programme	Information and Communication Technology
Professional Major	Game Applications
Instructor	Antti Laiho, Senior Lecturer

The goal of the thesis was to develop particle and shader effects for a 3D adventure game called Frog Shaman. The idea was to develop as many as possible of the game's 20 spell effects. The effects were developed with shaders and the particle system which exists in the Unity game engine.

The thesis studies the history of particle systems and shaders and investigates how these are used in the Unity game engine. It also contains research about the five elements that are used in the game for creating the spells: fire, air, ice, earth and electricity. Each of the developed spell effects somehow reflected the elements that were used in their creation in the game.

Good practices were also found regarding the creation of the effects. The effects should be communicating their purpose in the game clear enough to the player so there are no misunderstandings. They should also be interesting enough so players want to use them in the game.

Altogether twelve spell effects were developed during the thesis. The effects were developed according to the design and in a way that they can be used in the game as they are. Many kinds of particle effects and some shaders were used in the development of spell effects. Various features, which will be needed in the game in the future, were also added to the effects.

Most of the spell effects were finished, but some still lack features that will have to be added in the future. All the effects were still created in a way that their appearance can easily be altered when the visual appearance of the game becomes clear.

Keywords	particle, shader, effect, Unity

Sisällys

Lyhenteet

CPU	*Central Processing Unit* eli suomeksi suoritin. Sen tehtävänä on suorittaa ohjelmien konekielisiä käskyjä.
GPU	*Graphics Processing Unit.* Grafiikkasuoritin, jonka tehtävänä on hoitaa grafiikan renderöintiin liittyvät tehtävät.
UV	Pisteen koordinaatti tekstuurin vaaka- ja pystyakselilla. Vaaka-akselia nimetään kirjaimella U ja pystyakselia kirjaimella V.

1 Johdanto

Partikkeli- ja elementtitehosteilla tarkoitetaan tehosteita, joiden tarkoituksena on elävöittää peliä ja visualisoida pelaajalle pelissä tapahtuvia asioita, jotka ilman tehosteita saattaisivat jäädä epäselviksi. Tällaisia asioita voivat olla esimerkiksi keräilytehosteet tai räjähdykset.

Partikkelitehosteiden tarkoituksena on visualisoida asioita, jotka muilla keinoilla toteutettuina olisivat hankalia tai jopa mahdottomia. Tällaisia asioita ovat esimerkiksi tuli, vesi ja savu. Kaikki ovat luonnossa esiintyviä ilmiöitä, jotka muuttavat muotoaan ja liikkuvat ajan kuluessa. Tällaisia epäsäännöllisiä ja monimutkaisia muotoja olisi vaikea kuvata tavanomaisilla 3D-malleilla, jotka ovat parempia kiinteiden asioiden visualisointiin. Apuna partikkelitehosteiden luomisessa voidaan käyttää partikkelijärjestelmää (engl. Particle system). [1.]

Elementtitehosteet voivat puolestaan olla mitä tahansa tehosteita, jotka tehostavat jotakin pelin elementtiä. Tällaisia asioita voivat olla esimerkiksi animaatiot, äänet ja varjostimet, mutta tässä insinöörityössä keskitytään ainoastaan varjostimiin (engl. Shader). Varjostimet ovat ohjelmia, jotka suoritetaan GPU:lla jokaiselle pikselille erikseen. Tämän takia niiden täytyy olla mahdollisimman tehokkaita. Niiden tehtävänä on määrittää jokaiselle pikselille väriarvo erilaisten lähtötietojen perusteella.

Insinöörityön tarkoituksena on luoda erilaisia loitsutehosteita Frog Shaman -nimistä 3D-seikkailupeliä varten. Pelissä pelataan shamaani-hahmolla, jolla on hallussaan viisi eri elementtiä: tuli, ilma, jää, maa ja sähkö. Elementtejä yhdistelemällä voidaan luoda yhteensä 20 erilaista loitsua. Loitsuja käytetään hyväksi erilaisten ympäristössä olevien arvoitusten ratkaisemisessa sekä vihollisia vastaan taistelussa. Useimpien loitsujen toteutuksissa hyödynnetään partikkeli- ja elementtitehosteita. Tehosteet toteutetaan Unity-pelimoottorissa.

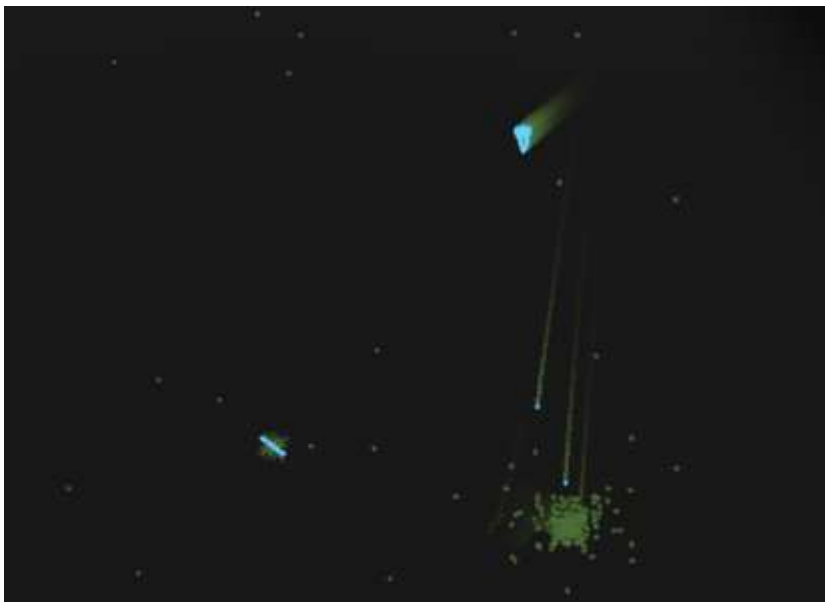
Aihe valittiin, koska se on hyvin suuressa osassa kyseisen pelin kehityksessä ja tehosteiden kehittäminen varhaisessa vaiheessa nähtiin hyödyllisenä. Kiinnostus partikkelitehosteisiin ja varjostimiin innoitti myös valitsemaan tämän aiheen.

2 Partikkeli- ja elementtitehosteiden käyttö peleissä

Erikoistehosteet ovat hyvin suosittuja nykyajan peleissä, mutta niitä on käytetty jo tietokonepelien alkuajoista lähtien. Niitä käytetään myös paljon elokuvatuotannossa. Erona peleihin täytyy kuitenkin huomata, että pelien täytyy suorittaa tehosteet reaaliajassa, kun taas elokuvat voivat kuluttaa tehosteiden renderöintiin monia tunteja.

2.1 Tehosteiden historia

Tehosteiden käyttö peleissä alkoi jo niinkin varhaisessa vaiheessa kuin vuonna 1962, jolloin SpaceWars!-niminen peli julkaistiin. SpaceWars! on kahden pelaajan avaruustaistelupeli, jossa pelaajat yrittävät ampua toisiaan torpedoilla ja väistää samalla kentän keskellä olevaa tähteä, joka vetää aluksia puoleensa. Pelin kehitti Steve Russel yhdessä muiden Study Group on Space Warfare -jäsenten kanssa vuonna 1961. Se julkaistiin keväällä 1962, jolloin se oli yksi ensimmäisistä, ellei jopa ensimmäinen tietokoneelle tehty peli. Se oli myös ensimmäinen tietokonepeli, jonka voidaan nähdä käyttäneen hyväksi partikkelitehosteita. Niitä ilmestyi näytölle, kun pelaajan alus tuhoutui, mikä näkyy kuvan 1 alareunassa. [2.]



Kuva 1. Alus tuhotaan torpedolla SpaceWars!-pelissä [2].

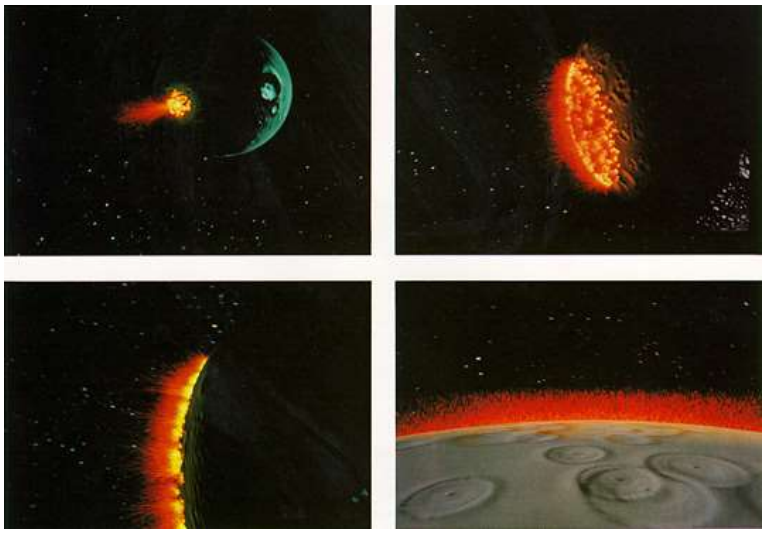
Partikkelitehosteet ovat siis olleet mukana pelinkehityksessä jo tietokonepelien alkuajoista lähtien. Toinen esimerkki pelien alkuajoilta on Asteroids, jonka Atari julkaisi vuonna 1979 (kuva 2). Pelissä ohjataan avaruusalusta, jonka tehtävänä on tuhota asteroideja ja ufoja. Kun asteroideja ammutaan, ne hajoavat pienemmiksi palasiksi, jotka täytyy myös tuhota. Asteroideihin osuessaan myös pelaajan ohjaama avaruusalus tuhoutuu useaksi osaksi. Nämä ovat esimerkkejä pelissä käytetyistä tehosteista. [3.]

Kuva 2. Asteroids-pelin arkadikoneen aloitusruutu [3].

Vuonna 1983 William Reeves julkaisi ensimmäisen partikkelijärjestelmää koskevan artikkelin, jossa hän käsitteli partikkelijärjestelmän luomista ja sen käyttöä ”Genesis”-efektin luomisessa, joka nähtiin vuoden 1982 elokuvassa Star Trek II: Khanin viha (kuva 3). Efektin oli tarkoitus esittää Kuun kaltaisen planeetan muuntautumista Maan kaltaiseksi planeetaksi Genesis-pommiksi kutsutun kokeilun avulla. Pommin osuessa planeettaan leviävä tuliseinämä liikkuu planeetan halki ”puhdistaen” sen. Efektissä käytettiin Reevesin mukaan yli 800:aa partikkelijärjestelmää. [4.]

Artikkelissaan Reeves kuvailee partikkelijärjestelmää kokoelmaksi pieniä partikkeleita, jotka yhdessä muodostavat epäselvän objektin. Ajan kuluessa partikkelit luodaan järjestelmään, ne liikkuvat ja muuttuvat järjestelmässä ja lopulta kuolevat järjestelmästä pois. Hän antaa myös esimerkkejä eduista, joita partikkelijärjestelmän käytöllä on. Esimerkiksi yhden partikkelin liikuttaminen ja muokkaaminen on paljon yksinkertaisempaa kuin esimerkiksi polygonin, josta tavalliset 3D-mallit koostuvat, sillä partikkeli voidaan ajatella vain yhdeksi pisteeksi. Reevesin partikkelijärjestelmässä partikkelijärjestelmälle

annettavia arvoja olivat paikka koordinaatit, nopeus ja suunta, koko, väri, muoto ja elinaika, ja näitä muuttamalla hän sai aikaiseksi edellä mainitun ”Genesis”-efektin. [4.]



Kuva 3. ”Genesis”-efekti [5].

”Genesis”-efektin jälkeen partikkelijärjestelmiä alettiin käyttää laajemmin partikkeleiden ohjaamiseen. Reeves loi partikkelijärjestelmälle hyvän pohjan, jota on voitu käyttää vuosien saatossa hyväksi uudempien partikkelijärjestelmien kehittämisessä. Esimerkiksi Unity-pelimoottorin partikkelijärjestelmä on hyvin samankaltainen, mutta ajatusta on kehitetty pidemmälle.

Varjostimien historia puolestaan alkoi 1980-luvulla, kun Lucasfilm palkkasi grafiikkaohjelmoijia tietokoneistamaan spesiaalitehosteiden teollisuudenalan. Myöhemmin samalla vuosikymmenellä yksiköstä tuli Pixar, yksi tunnetuimmista animaatioelokuvien tuottajista. Se on tehnyt muun muassa Toy Story -elokuvasarjan. Työn tuloksena syntyi lopulta Renderman-renderöintiohjelma, jota kehitetään yhä edelleen elokuvatuotannon tarpeisiin. [6; 7.]

Alun perin sanaa varjostin käytettiin ainoastaan puhuttaessa pikselivarjostimista, mutta myöhemmin sen tarkoitus laajeni tarkoittamaan myös kärkipiste- ja geometriavarjostimia. Vielä 1980-luvulla varjostimien kehittäminen oli vaikeaa, sillä eri alustat tarvitsivat erilaisia ohjelmia toimiakseen. Tämän helpottamiseksi syntyivät OpenGL ja DirectX. Ne

tarjoavat helppokäyttöisemmän ja abstraktin rajapinnan varjostimien kirjoittamiseen, jolloin ei tarvitse miettiä yhtä paljon sitä, mille alustalle varjostimia ollaan kirjoittamassa. [6.]

Kun ensimmäiset näytönohjaimet julkaistiin vuonna 1995, oli vihdoin mahdollista renderöidä grafiikkaa reaaliajassa ja kehittää 3D-pelejä. Varjostimia ei kuitenkaan vielä kyetty käyttämään, sillä käytössä oli FFP-renderöintikanava (engl. fixed-function pipeline). FFP tarjosi erilaisia muutosmahdollisuuksia renderöintiprosessiin, mutta mahdollisuudet olivat ennalta määriteltyjä ja siksi rajallisia. Se ei esimerkiksi hyväksynyt omia algoritmeja. PP-renderöintikanava (engl. programmable pipeline) tuli käyttöön vuonna 2002, jolloin varjostimien käyttö tuli vihdoin mahdolliseksi. Aiemmat määrätyt funktiot korvattiin yleisillä vaiheilla, joissa pystyi muuntamaan dataa haluamallaan tavalla. Ohjelmoijalla oli vihdoin täysi hallinta jokaiseen näytöllä näkyvään kärkipisteeseen ja pikseliin. Oli siis mahdollista tehdä mitä tahansa, jos vain osasi ohjelmoida sen. [6.]

Alun perin varjostimia kirjoitettiin Assembly-ohjelmointikielellä, mutta sen kirjoittaminen ja lukeminen oli hankalaa, joten oli tarve kehittää korkeammantasoisia varjostinohjelmointikieliä. Vuonna 2004 syntyi GLSL, jota käytettiin OpenGL-ohjelmien kirjoittamiseen [8]. Vastaavasti HLSL-ohjelmointikieli syntyi DirectX-ohjelmien kirjoittamiseen. Syntyi myös CG-ohjelmointikieli, jolla pystyi kirjoittamaan ohjelmia molemmille rajapinnoille. [6.] CG-kielen tuki lopetettiin vuonna 2012, mutta esimerkiksi Unity-pelimoottori käyttää sitä edelleen [9].

2.2 Tehosteiden käyttö samankaltaisissa peleissä

Partikkelitehosteita käytetään paljon loitsuefektejä sisältävissä peleissä, sillä ne ovat yleensä paras tapa taikojen epäselvän ja mystisen muodon kuvaamiseen. Myös elementtitehosteita on jokaisessa markkinoilla olevassa 3D-pelissä, sillä ilman niitä olisi mahdotonta kuvata objektien ulkonäköä. Partikkelitehosteita on käytetty hyväksi taikatehosteiden tekemiseen esimerkiksi Mages of Mystralia- ja Magicka-peleissä.

Mages of Mystralia -pelissä pelataan Zia-nimisellä maagilla, joka on juuri löytänyt voimansa. Pelissä on tarkoituksena luoda erilaisia loitsuja yhdistelemällä monenlaisia loitsukomponentteja ja käyttämällä luotuja loitsuja taistelussa vihollisia vastaan. Pelissä on myös arvoituksia, jotka vaativat tietynlaisen loitsuyhdistelmän ratketakseen. Käytössä on

neljä erilaista peruselementtiä: tuli, jää, maa ja sähkö. Elementtien perusloitsut ovat näkyvissä kuvassa 4. Loitsujen luonti ei kuitenkaan perustu itse elementteihin, vaan elementteihin yhdistettäviin riimuihin, jotka ohjaavat loitsun toimintaa. Esimerkiksi liike-riimun avulla loitsu lentää eteenpäin ja kun siihen lisää vasen-riimun, se liikkuu vasemmalle. Näiden avulla pystytään luomaan satoja erilaisia loitsuyhdistelmiä. [10.]

Kuva 4. Mages of Mystralia -pelin neljä eri elementtiä [10].

Magicka-peli perustuu sekin loitsujen luomiseen. Magicka on toimintaseikkailupeli, joka sijoittuu satiiriseen fantasiamaailmaan. Sitä pystyy pelaamaan yksin tai yhdessä kolmen ystävän kanssa. Pelaajien tehtävänä on pelata pyhään järjestöön kuuluvilla velhoilla, joiden tehtävänä on pysäyttää paha velho, joka on ajanut maailman kaaokseen. Pelissä taistellaan vihollisia vastaan loitsujen avulla. [11; 12.]

Magicka-pelin loitsujen luonti perustuu elementtien yhdistämiseen. Elementtejä on oikeastaan kahdeksan, mutta yhdistämällä muita elementtejä voidaan luoda kaksi elementtiä lisää. Magickassa on käytössä tuli-, vesi- ja maaelementit, jotka kuuluvat antiikin aikaisiin klassisiin elementteihin, mutta niiden lisäksi käytössä on vielä monta muuta elementtiä: kylmä, kilpi, mystiikka, elämä, salama, höyry ja jää. Näitä elementtejä pystytään yhdistämään enintään viiden mittaisiksi yhdistelmiksi. Lisäksi loitsua voidaan käyttää neljällä eri tavalla: se voidaan loitsia velhoon, velhon käyttämään miekkaan tai kahdella erilaisella hyökkäystavalla. Erilaisia yhdistelmiä on siis yli tuhat. Jokaisella elementillä on kuitenkin jokin vastaelementti, jonka kanssa sitä on mahdotonta yhdistää, mikä rajoittaa hieman mahdollisia loitsuyhdistelmiä. [12.]

Kummassakin esimerkkipelissä on toisin sanoen hyvin monta erilaista partikkelijärjestelmää, jotka on pitänyt toteuttaa. Todennäköisesti esimerkiksi Mages of Mystralia -pelin taustalla partikkelijärjestelmän tai loitsun toimintaa ohjataan edellä mainittujen riimujen avulla ja saadaan siten tehosteet näyttämään erilaisilta riimujen vaihtojen vuoksi. Jotain samankaltaista saatetaan käyttää myös Magickassa, sillä on hyvin epätodennäköistä, että kehittäjät ovat kehittäneet itse jokaiselle loitsulle tehosteen, kun yhdistelmiä voi olla jopa tuhat.

Monissa loitsuissa on selvästi näkyvillä loitsun eri vaiheet: ilmestymisefekti, itse efekti ja jonkinlainen päättyminen. Näistä esimerkkinä on Mages of Mystralia -pelin kiviloitsu, jonka vaiheet näkyvät kuvasta 5. Ilmestymisefektinä toimii taikasauvan kärkeen ilmestyvä pölyhiukkasista ja lehdistä koostuva pölypilvi, jonka keskeltä kivi ilmestyy. Loitsun puolivälissä kivi lentää kaaressa ja sen vanaan piirtyy liikeviivoja, lehtiä ja pölyhiukkasia sekä pieniä kiviä. Lopulta, kun kivi osuu maahan, kivi korvataan pienemmillä kivenpalasilla, jotka jatkavat kiven liikerataa. Osumapaikalle ilmestyy myös valkoisia räjähdystä kuvaavia partikkeleita sekä pölyä.

Kuva 5. Mages of Mystralia -pelin kiviloitsun eri vaiheet [10].

Lisäksi Mages of Mystralia -peli on tehty Unity-pelimoottorilla, jota myös tässä insinöörityössä käytetään. Tässä nähdään esimerkki siitä, millaisia efektejä Unityn partikkelijärjestelmällä voidaan saada aikaan. Loitsuefekteissä on myös todennäköisesti käytetty erilaisia elementtitehosteita, mutta niitä on vaikeampi erottaa, sillä ei voi olla varma, onko tehosteet tehty varjostimilla vai lisätty materiaalin käyttämään tekstuuriin.

Useimmat käytetyistä loitsutehosteista sisältävät todennäköisesti myös useampia kuin yhden partikkelijärjestelmän. Tämä näkyy hyvin edellä mainitussa kiviloitsussa, mutta myös Magicka-pelin tuliloitsussa, jota kuva 6 esittää. Loitsussa on nähtävästi käytetty

ainakin kolmea eri partikkelijärjestelmää. Yksi partikkelijärjestelmä kuvaa itse tulta. Tulen väri muuttuu elinaikansa aikana sinertävän valkoisesta oranssiksi, mikä kuvaa tulen käyttäytymistä todellisuudessa. Tehosteessa käytetään myös savua, joka näkyy tummana tehosteen kärjessä. Savu saattaa myös toimia osumatehosteena, jolloin se liikutetaan liekin päätyyn tai kohtaan, jossa se osuu esteeseen. Kolmas partikkelijärjestelmä näyttäisi olevan maahan piirtyvä tumma jälki, joka on saatettu toteuttaa tarrojen (engl. decal) avulla, joita pystytään toteuttamaan myös partikkelijärjestelmää käyttäen. Tarrat ovat kuvia, jotka piirretään olemassa olevan pinnan päälle.

Kuva 6. Magicka-pelin tuliloitsu [12].

2.3 Elementtien määritelmät

Frog Shaman -pelissä erittäin isossa osassa on elementtien yhdistäminen. Näitä elementtejä ovat tuli, ilma, jää, maa ja sähkö. Tämä poikkeaa hieman klassisesta neljästä elementistä, joista jo antiikin Kreikan filosofit puhuivat 450-luvulla eaa. Klassisia elementtejä olivat tuli, vesi, ilma ja maa, joista kaiken materian uskottiin koostuvan. Myöhemmin on tietysti todettu maailman koostuvan useammasta alkuaineesta, mutta esimerkiksi fantasiavideopeleissä on tavallista hyödyntää näitä klassisia elementtejä tai niiden muunnelmia. Elementtejä käytetään usein loitsujen luomiseen, ja ne järjestetään usein vastakkain jonkin toisen elementin kanssa, jotta pelin voimasuhteita saataisiin tasoitettua. Esimerkkinä voisivat olla tuli ja vesi, jotka voidaan nähdä toisensa kumoavina vastavoimina. Myös ilma ja maa voidaan nähdä vastaelementteinä, sillä ilma on kevyttä ja maa raskasta. [13.]

Tuli

Tuli on yksi klassisista elementeistä, mutta muista poiketen se ei oikeastaan ole ainetta vaan näkyvä sivuvaikutus siitä, kun aine vaihtaa muotoaan. Tuli syntyy, kun happi ja jokin polttoaine reagoivat polttoaineen syttymislämpötilassa. Palamisesta aiheutuvat reaktiot voivat palavasta aineesta riippuen olla erilaisia. Esimerkiksi poltettaessa puuta palamisen tuotteena syntyy savua, hiiltä ja tuhkaa, mutta poltettaessa hiiltä ei savua synny ollenkaan, sillä savut ovat jo puun polton yhteydessä poltettu pois. [14.]

Palaminen synnyttää liekkejä, jotka voivat olla polttoaineesta ja kuumuudesta riippuen hyvinkin erivärisiä ja -kokoisia. Liekin väri on yleensä kuumimmassa kohdassa sininen ja viileämmissä kodissa ylempänä keltainen tai punainen. Liekit aiheuttavat ympärilleen lämpöä ja valoa. Lämmön tuottamisen takia liekki pystyy jatkamaan palamista niin kauan, kuin ympärillä on happea ja polttoainetta. Tämän takia se on melko vaarallinen ja herkkä leviämään. Liekit myös kapenevat aina ylöspäin mentäessä, sillä liekin kaasut ovat paljon kuumempia ja vähemmän tiheitä kuin ympäröivä ilma, minkä takia ne kohoavat ylöspäin kohti alempaa painetta. Jos liekki sytytettäisi painovoimattomassa tilassa, se olisi pyöreä. [14.]

Peleissä tuli nähdään usein tuhoavana elementtinä, jolla on tarkoitus aiheuttaa vahinkoa vihollisille. Monissa peleissä tuli myös leviää, jolloin sen vaikutusta on vaikea arvioida etukäteen. Tällöin pelaaja saattaa päätyä itsekin tulen uhriksi, jos ajautuu erehdyksessä liian lähelle sitä. Tulen avulla voidaan myös polttaa tiellä olevia esteitä, jotka palaessaan tuhoutuvat ja vapauttavat tien.

Ilma

Ilma on läpinäkyvää ja koostuu kaasumaisista aineista, joten se ei näytä itsessään juuri miltään. Tämän takia sitä on hieman hankalampaa kuvata visuaalisesti kuin muita klassisia elementtejä. Sen tavallisin olomuoto peleissä on kuitenkin tuuli. Tuuli tunnetusti liikuttaa asioita mukanaan, joten sitä voidaan kuvata esimerkiksi pilvien liikkeellä tai tuulen mukana liikkuvan pölyn tai lehtien avulla. Myös erilaiset tuulen liikuttamat esineet, kuten liput tai puut, voivat kuvata tuulen voimakkuutta. Yksi selkeimmistä tavoista on myös

käyttää viivoja, jotka visualisoivat ilman liikettä, mistä kuva 7 on hyvä esimerkki. Tällä tavalla voidaan helposti kuvata tuulen muodostamia pyörteitä.

Kuva 7. Esimerkki ilman visualisoinnista [15].

Tuulen lisäksi ilma voi ilmentyä peleissä esimerkiksi pyörremyrskynä. Pyörremyrsky on hyvin voimakas ilmiö, jonka tuulennopeus voi oikeassa elämässä kohota 512 kilometrin tuntinopeuteen, ja se voi olla kilometrien levyinen. Isot tornadot kykenevät hajottamaan taloja ja lennättämään jopa autoja. Pyörremyrskyt syntyvät usein ukkosmyrskyjen yhteydessä, jolloin on otolliset olosuhteet pyörremyrskyn syntymiseen. Lämmin matalapaineinen ilma vetää puoleensa kylmempää korkeapaineista ilmaa, eli syntyy tuuli, joka suuntaa kohti matalapainetta. Tuuli työntää matalapaineista ilmaa ylöspäin, mutta lämpenee itsekin ja alkaa kohota. Koska ilma virtaa paikalle joka suunnasta, alkaa matalapaineen paikalle muodostua pyörre. Tornadojen ilmanpaine voi olla jopa 10 prosenttia matalampi kuin ympäröivä ilma, jolloin ympärillä oleva ilma syöksyy mukaan vielä nopeammin. Pyörteet ovat muutenkin yleisiä ilmiöitä. Niitä voidaan nähdä esimerkiksi aavikoilla, kun tuuli kulkee kuuman hiekan yli ja aiheuttaa pölypyörteen. Myös maastopalot voivat aiheuttaa kohoavia tuli- ja tuhkapyörteitä. [16.]

Jää

Jää on yksi Frog Shaman -pelissä käytettävistä elementeistä, mutta se ei kuulu klassisiin elementteihin, vaikka onkin yksi veden olomuodoista. Sillä voidaan nähdä olevan paljon samanlaisia ominaisuuksia kuin vedellä. Esimerkiksi sekä jää että vesi sammuttavat molemmat tulen ja synnyttävät vesihöyryä haihtuessaan. Monissa fantasiapeleissä on kuitenkin erotettu jää ja vesi omiksi elementeikseen. Erona elementeissä on usein se, että vesi kastelee vihollisia ja jää jäädyttää. Frog Shaman -pelissä on kuitenkin päädytty

siihen, ettei veden ominaisuuksia välttämättä tarvita, sillä jää toimii monissa tilanteissa vettä monipuolisemmin.

Jään kylmyyden takia sen loogisia visualisointitapoja ovat lumihiutaleet tai jääkiteet, joita käytetäänkin monissa peleissä. On myös tavallista, että jää kykenee jäädyttämään kastuneita vihollisia ja vesialueita. Jää on myös jonkin verran läpinäkyvää ja heijastavaa, minkä vuoksi sitä voidaan käyttää hyväksi esimerkiksi valon heijastamisessa, niin kuin esimerkiksi Mages of Mystralia -pelissä tehdään. Pelissä on mahdollista luoda hahmon eteen jääkilpi, joka sitten valoon osuessaan heijastaa säteet eteenpäin. Tällainen ominaisuus on todennäköisesti tulossa myös Frog Shaman -peliin.

Maa

Myös maa on yksi neljästä klassisesta elementistä. Se on käsitteenä muita elementtejä paljon laajempi, sillä maa voi olla muodoltaan pehmeää kasvurikasta maata tai se voi olla kovaa kiveä. Siksi sillä on peleissä monenlaisia käyttökohteita ja erilaisia tapoja visualisoida sitä. Monissa peleissä, kuten esimerkiksi Mages of Mystraliassa ja Magickassa, on kiviloitsu. Se on yksinkertainen loitsu, jossa heitetään kivi ja osuessaan se aiheuttaa vahinkoa. Toisaalta maa voi olla myös parantava elementti, sillä kasvattaahan se kasvejakin, joten elämän luominen sopii tähän elementtiin hyvin.

Sähkö

Sähkö ei myöskään ole klassisia elementtejä, mutta se on silti monessa pelissä, jossa käytetään elementtejä loitsujen luontiin. Sähköllä on usein tarkoitus satuttaa vihollisia, ja esimerkiksi Magicka-pelissä on mahdollista aiheuttaa lisää vahinkoa, mikäli kohde on kastunut.

Sähköä visualisoidaan yleisesti salamoilla. Todellisuudessa salamat ovat sähköisiä purkauksia, jotka johtuvat sähköisestä epätasapainosta pilvien ja maan välillä, tai jopa pilvien itsensä. Useimmat salamat nimittäin purkautuvat pilven sisällä. Salamat ovat erittäin kuumia: ne voivat lämmittää ympäröivän ilman jopa viisi kertaa auringon pintaa kuumemmaksi. Koska lämpö saa ilman laajentumaan nopeasti, kuuluu salamaniskusta aina ukkosenjyrähdykseksi kutsuttu ääni. Pienemmistä kipinöistä, esimerkiksi hankaussähköstä

johtuvat purkaukset kuulostavat pieniltä räsähdyksiltä. Niitä voi kuulla esimerkiksi villapaitaa riisuessaan. Salaman kuumuus saattaa myös räjäyttää puun siihen osuessaan, sillä kuumuus höyrystää puussa olevan veden, joka sitten purkautuu räjähdyksenä. [17.]

3 Tehosteiden suunnittelu

3.1 Tehosteiden suunnittelun neljä pilaria

Tehosteiden suunnitteluun on todennäköisesti paljon eri lähestymistapoja, mutta esimerkiksi Julian Love kertoo neljästä pilarista, joiden mukaan hän suunnitteli tehosteita Diablo-peliin. Ensimmäisenä ja todennäköisesti tärkeimpänä on viestiä pelattavuutta. Pelaaja voi kokea pelin huonoksi, mikäli hän huomaa, että iso räjähdys ei teekään yhtään sen enempää vahinkoa kuin pelissä oleva pienempi räjähdys. Toisin sanoen tehoste ja pelin design eivät vastaa toisiaan ja tehoste aiheuttaa tässä tapauksessa turhia odotuksia pelaajalle. Tärkeää on siis kulkea pelattavuus edellä ja kertoa pelaajalle, mitä hänen kuuluu tietää. [18.]

Toisena pilarina Love mainitsee samaistuttavuuden, sillä ihmiset tunnetusti ymmärtävät helpommin asioita, joita ovat nähneet aiemmin. Hän esimerkiksi suosittelee käyttämään hyväksi pelkoja. Pelkojen avulla pelaaja tietää vältellä asioita, jotka todellisuudessa aiheuttaisivat hänelle vahinkoa, esimerkiksi tuli ymmärretään karkottavana elementtinä. Hän myös mainitsee, kuinka tärkeää on ajatella tehosteiden tempoa. Jos tempo on hitaampaa kuin ihmisen sydämensyke, on tehoste usein kutsuvampi pelaajille kuin nopeampi tempo. Jos tallennuspaikkaa haluttaisiin kuvastaa tehosteella, on järkevä laittaa sen liike melko hitaaksi, jottei pelaajalle tule oloa, että hänen pitäisi välttää sitä. Vastaavasti räjähtävän pommin voi laittaa sykkimään nopeasti, jolloin pelaaja pyrkii todennäköisesti kauemmas siitä. [18.]

Kolmantena pilarina kielletään kokemuksen pilaaminen. Tällä Love todennäköisesti tarkoittaa sitä, että kaiken näytöllä näkyvän pitää olla selkeää ja ymmärrettävää pelaajalle. Esimerkkinä hän mainitsee yhtiön käyttävän paljon varjostimia, jotka käyttävät sekoitustapana (engl. blending mode) sekoitussummaa (engl. blend-add). Sen avulla saadaan tehosteet näyttämään hyvältä kaikkia taustoja vastaan. Sekoitustavalla tarkoitetaan

tapaa, jolla varjostimen antama väriarvo pitäisi sekoittaa näytöllä jo olemassa olevaan väriarvoon, jolloin pikselin lopullinen väriarvo määräytyy sekoitustavan perusteella. Tavallisessa summaustavassa summataan jo näytöllä oleva arvo varjostimesta tulevan väriarvon kanssa yhteen, jolloin syntyy yleensä paljon valkoista, mikä näyttää vaaleaa taustaa vasten hyvin sekavalta. Sekoitustavassa puolestaan hyödynnetään tekstuurin alfa-arvoa määrittämään sitä, mihin tekstuuri pitäisi piirtää. Tämän avulla saadaan aikaiseksi melko hyvä tulos, mutta siitä puuttuu hehkua, joka esimerkiksi salaman kuvaamisessa olisi hyödyllistä. Sekoitussummatavassa siis yhdistetään nämä kaksi tapaa. Jos alfakanavasta tuleva arvo on musta, summataan, ja jos valkoinen, niin sekoitetaan. Näin saadaan aikaiseksi hehkua, joka näyttää hyvältä kaikkia taustoja vasten. [18.] Kuva 8 hahmottaa paremmin näitä eri sekoitustapoja.

Kuva 8. Julian Loven havainnollistus sekoitussumma-sekoitustavasta [18].

Viimeisenä pilarina Love mainitsee, että tehosteen pitää olla viihdyttävä. Jokaisen tehosteen pitäisi jo itsessään tarjota pelaajalle hieman viihdettä, mikä tietenkin vaikuttaa siihen, paljonko pelaaja tehostetta käyttää. Hän mainitsee keinoista, joilla Blizzardilla on toteutettu tällaisia mieleenpainuvia tehosteita. On esimerkiksi käytetty partikkeleissa varjostimia, jotka liikuttavat tekstuureiden UV-koordinaatteja ja skaalaavat tekstuureita erikokoisiksi. Tämä saa usein aikaiseksi melko monimutkaista liikettä, josta katsojan on

vaikeampi nähdä toistuvia kuvioita. Usein haluttua monimutkaisuutta saadaan aikaiseksi vain yhdenlaista tekstuuria käyttämällä. [18.]

3.2 Halutut tehosteet

Frog Shaman -peliin on suunniteltu yhteensä 20 loitsutehostetta, joista mahdollisimman moni pyritään toteuttamaan tämän insinöörityön aikana. Pelissä käytetään viittä eri elementtiä loitsujen muodostamiseen yhden tai kahden elementin yhdistelmiksi. Elementit ovat tuli, ilma, jää, maa ja sähkö. Elementtien yhdistelmiä mietittäessä on pohdittu eri elementtien ominaisuuksia sekä sitä, mikä voisi olla pelattavuuden kannalta hauskin toiminnallisuus. Tämän lisäksi osan loitsuista halutaan toimivan yhdessä. Esimerkiksi pyöremyrskyloitsun halutaan kasvattavan kokoaan, mikäli siihen puhalletaan tuuliloitsulla.

Käytettävistä elementeistä näkyy hyvin, että vaikutteita on otettu klassisista elementeistä, mutta myös muista samankaltaisista peleistä. Tyypillisesti tällaisissa peleissä on käytetty elementtinä myös vettä, mutta loitsujen määrästä ja veden ominaisuuksista johtuen todettiin, ettei se pelin kannalta olisi kovin tärkeä elementti ja jää toimii monessa tilanteessa vettä monimuotoisemmin. Elementtien ominaisuudet ovat olleet hyvin suuressa osassa elementtiyhdistelmiä mietittäessä, sillä yhdistelmistä haluttiin saada mahdollisimman helposti ymmärrettäviä. Esimerkiksi on loogista, että tulta ja jäätä yhdistämällä saadaan aikaiseksi sumua. Voidaan ajatella, että jää toimii tässä hyvin samalla lailla kuin vesi. Se vain ensin sulaa vedeksi ja höyrystyy sitten vesihöyryksi eli sumuksi.

Tärkeimmät loitsut sisältävät vain yhtä elementtiä ja heijastavat siksi eniten kyseisen elementin ominaisuuksia. Näiden loitsujen on ajateltu olevan suurimmassa käytössä peliä pelattaessa, ja siksi ne pitäisi toteuttaa ensimmäisenä. Yhden elementin loitsut vievät yleisesti vähemmän pelissä käytettävää loitsuenergiaa kuin vahvemmat kahden elementin loitsut.

Yhden elementin tuliloitsu on eräänlainen lämpösuihku, jonka tarkoituksena on polttaa erilaisia esteitä ja satuttaa vihollisia. Loitsun kesto riippuu siitä, kuinka kauan loitsupainiketta pidetään pohjassa. Kahden elementin tuliloitsu sen sijaan on tulipallo, joka tähtää lähimpänä olevaan viholliseen. Sen tarkoituksena on aiheuttaa vahinkoa, mutta se kykenee periaatteessa tekemään saman kuin lämpösuihkukin. Kumpikin aiheuttaa

palamista, mikäli osuvat viholliseen tai palavaan esteeseen. Tällöin niihin syntyy liekkejä, jotka pystyvät leviämään muihin vihollisiin ja pelaajaan, jos tämä tulee liian lähelle. Tarkoitus olisi myös, että liekit voisivat levitä maassakin. Palavissa esteissä pitäisi puolestaan olla elementtitehoste, joka muuttaisi esineen palavaksi ja häivyttäisi sen pois näkyvistä, kun esine on palanut loppuun.

Ilmaloitsut liittyvät puolestaan vahvasti asioiden liikuttamiseen. Ensimmäinen ilmaloitsu on eräänlainen puhallus. Sen tarkoituksena on liikutella esineitä, kuten laatikoita tai purjeita. Sillä pitäisi pystyä myös levittämään tulta, mikäli liekkeihin puhalletaan, sekä kasvattamaan tornadoloitsun kokoa. Kahden elementin loitsu onkin siis tornadoloitsu, joka on puhallusloitsua voimakkaampi. Sillä voi esimerkiksi lennättää vihollisia pois tieltä, ja sen päälle on mahdollista hypätä, jolloin pelaaja saa lisää nopeutta ja korkeutta. Tornadon pitäisi myös syttyä palamaan, jos se menee liekkien yli. Sen lisäksi sen partikkeleiden pitäisi muuttua hieman sen perusteella, millä alueella ollaan. Esimerkiksi ruohikkoisella alueella partikkeleina olisivat vihreät lehdet, lumisella puolestaan lumihiutaleet.

Jääloitsut ovat osittain tuliloitsujen vastaloitsuja. Yhden elementin loitsu on jääsuihku, eli hyvin samankaltainen kuin tuliloitsun lämpösuihku, mutta se sisältää tulen sijasta jäätä. Jääsuihkulla pitäisi pystyä jäädyttämään maata ja vihollisia sekä joitakin esineitä. Jääsuihkulla pystyy myös sammuttamaan tulesta syntyneitä liekkejä, mutta vastaavasti tulisuihku pystyy sulattamaan jääsuihkun aiheuttamaa jäätä. Kahden elementin loitsuna on jääaita, joka on tarkoitus satuttaa, jos siihen osuu. Sen lisäksi se rajaa aluetta ja pakottaa viholliset kiertämään sen. Jääaidan pystyy tuhoamaan ainoastaan tulisuihkulla tai vahvalla iskulla.

Maaloitsuja ovat puolestaan kivi- ja liaaniloitsu. Kiviloitsun tarkoituksena on satuttaa vihollisia, ja sillä pystyy painamaan vaikeasti saavutettavia nappeja. Liaaniloitsua taas tuskin toteutetaan vielä tämän insinöörityön aikana, sillä sen toteutus perustuu todennäköisesti muihin menetelmiin kuin partikkeleihin.

Yhden elementin sähköloitsu on salama, jota pystytään käyttämään taistelussa, mutta sillä voidaan myös esimerkiksi katkoa puita ja käynnistää sähkölaatikoita. Sen on myös tarkoitus aiheuttaa vihollisille sähköiskuja, jolloin ne jähmettyvät hetkeksi ja

vahingoittuvat. Kahden elementin loitsun on tarkoitus olla hyvin samankaltainen, mutta se aiheuttaisi vahinkoa laajemmalle alueelle.

Yhdistelmäloitsuja ovat puolestaan esimerkiksi jään ja tulen yhdistelmästä syntyvä sumuloitsu, jonka tarkoituksena on kätkeä pelaaja vihollisilta. Sitä siis voidaan käyttää vihollisten ohi hiipimiseen. Insinöörityössä toteutetaan myös lumimyrsky loitsu, joka syntyy jäätä ja ilmaa yhdistämällä. Sen on tarkoitus toimia hieman samalla tavalla kuin jääsuihkun, mutta laajemmalla alueella. Pommiloitsu syntyy puolestaan maata ja tulta yhdistämällä. Sen on tarkoitus esittää laavapommia, joka räjähtäisi tietyn ajan kuluessa, ja sillä pystyttäisiin rikkomaan erilaisia esteitä. Myös muita yhdistelmiä toteutetaan, mikäli aika riittää. Kaikki suunnitellut loitsut voidaan nähdä taulukosta 1.

Taulukko 1. Kaikki peliä varten tehtäväksi suunnitellut loitsut.

Elementtiyhdistelmä	Loitsu	Ominaisuudet
tuli	lämpösuihku	Aiheuttaa vahinkoa ja palamista. Sulattaa jäätä.
tuli + tuli	tulipallo	Aiheuttaa vahinkoa ja palamista.
ilma	tuulipuhallus	Liikuttaa asioita.
ilma + ilma	tornado	Lennättää asioita, ja pelaaja pystyy liikkumaan sen avulla.
jää	jääsuihku	Jäädyttää vihollisia ja maata. Sammuttaa liekkejä.
jää + jää	jääaita	Aiheuttaa vahinkoa ja rajaa aluetta.
maa	kivi	Aiheuttaa vahinkoa ja painaa nappeja.
maa + maa	liaani	Mahdollistaa asioihin tarttumisen.
sähkö	salama	Aiheuttaa vahinkoa ja aktivoi sähkölaatikoita.
sähkö + sähkö	iso salama	Aiheuttaa vahinkoa laajemmalle alueelle.
tuli + ilma	tulispiraali	Aiheuttaa vahinkoa ja palamista.
tuli + jää	sumu	Kätkee pelaajan.
tuli + maa	pommi	Aiheuttaa vahinkoa ja räjäyttää esteitä.
tuli + sähkö	laser	Aiheuttaa vahinkoa.
ilma + jää	lumimyrsky	Aiheuttaa vahinkoa ja jäätymistä laajalla alueella.
ilma + maa	leijutus	Mahdollistaa asioiden leijuttamisen.
ilma + sähkö	ukkospilvi	Salamoi ympärilleen ja aiheuttaa vahinkoa.
jää + maa	jääkilpi	Suojaa pelaajaa ja heijastaa valoa.
jää + sähkö	valopallo	Valaisee pimeää ja häikäisee vihollisia.
maa + sähkö	ansa	Aiheuttaa vahinkoa siihen osuttaessa.

Kaikkien loitsujen olisi tarkoitus olla visuaaliselta ilmeeltään samantyyppisiä. Loitsujen pitäisi myös olla riittävän mielenkiintoisia, että pelaajat jaksaisivat katsella niitä, sillä ne ovat suuressa osassa pelin ulkoasun ja pelattavuuden toteutuksessa. Niiden pitäisi myös

toimia pelissä hyvin ja olla hidastamatta pelin päivitysnopeutta liikaa. Useimpia loitsutehosteita voi olla näytöllä samanaikaisesti useita, minkä vuoksi onkin hyvin tärkeää pitää huolta niiden tehoista. Onneksi projekti kuitenkin toteutetaan tietokoneille, jolloin tehorajoitteet eivät ole yhtä kovat kuin esimerkiksi mobiilialustoille kehitettäessä. Tehojen lisäksi monien loitsujen kanssa täytyy ottaa huomioon sekin, miten tehosteet piirretään näytölle. Jos loitsut joutuvat taistelemaan samasta piirtosyvyydestä, saattaa vahingossa käydä niin, että loitsut näyttävät vilkkuvilta, mikä ei ole haluttua.

3.3 Käytettävät työkalut

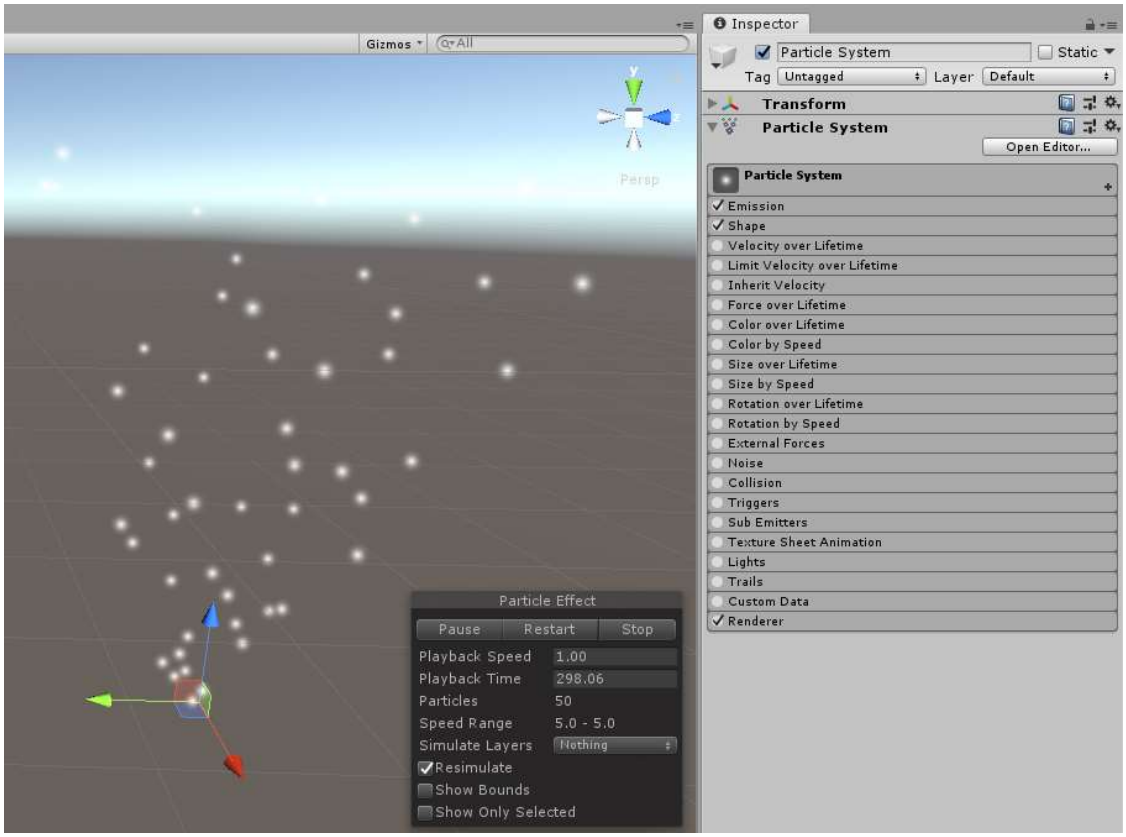
Frog Shaman -peliä tehdään Unity-pelimoottorilla, joten se toimii pääasiallisena kehitysalustana. Unity on alustariippumaton pelimoottori, jota kehittää Unity Technologies. Sen ensimmäinen versio julkaistiin kesäkuussa vuonna 2005, ja sitä on siitä lähtien päivitetty säännöllisesti. Tällä hetkellä se tukee 27:ää eri alustaa. [19.]

Unity-pelimoottorilla pystytään tekemään 2D- ja 3D-pelejä. Pääasiallisena kehityskielenä on C#, mutta itse pelimoottorin kielenä on käytetty C++:aa, joka on C#:a usein hieman tehokkaampi. C# on kuitenkin aloittelijoille hieman helpommin omaksuttava kieli kuin C++, sillä esimerkiksi muistinhallintaan ei tarvitse kiinnittää niin paljon huomiota. Unity onkin suuressa suosiossa erityisesti harrastelijoiden ja pienempien yritysten keskuudessa, mikä on toisaalta myös hieman huonontanut sen mainetta. Unity on tehnyt pelien tekemisen mahdolliseksi kaikille, joten markkinoille päätyy vääjäämättäkin nopeasti kehitettyjä pelejä, joita tehdessä ei ole nähty paljon vaivaa. Tähän vaikuttaa sekin, että Unityn käytön aloittaminen on ilmaista, ja vasta, kun yrityksen tulot nousevat yli 100 000 euroon, on pelimoottorista alettava maksaa. [19.]

Unityssa peli kootaan GameObject-luokkien avulla. Jokainen pelissä oleva asia on GameObject-luokan ilmentymä. GameObject-luokat eivät kuitenkaan itsessään ole juuri mitään, vaan niihin pitää lisätä komponentteja, jotka antavat objektille ominaisuuksia. Esimerkiksi luotaessa kameraa pitää objektiin liittää kamerakomponentti ja valoa luotaessa valokomponentti. Myös kehittäjien luomat skriptit ovat komponentteja, joita voidaan lisätä objekteihin. [19.]

Unity-pelimoottorin partikkelit

Unity on toteuttanut partikkelijärjestelmänsä komponentin avulla. Sen voi lisätä Unityn editoriin valmiin objektin mukana tai olemassa olevaan objektiin komponenttivalikosta. Kuvasta 9 voi nähdä partikkelijärjestelmäkomponentin ja sen luoman partikkelijärjestelmän, jonka se luo automaattisesti. Partikkelijärjestelmäkomponentti on melko monimutkainen, ja siksi se on jaettu erillisiin alaluokkiin eli moduuleihin. Moduulit sisältävät aihealueeseensa liittyviä ominaisuuksia, joita voidaan säätää tarpeen mukaan tehosteeseen sopiviksi. [20.]



Kuva 9. Partikkelijärjestelmäkomponentti ja oletuksena luotu partikkelijärjestelmä.

Oletuksena luodussa partikkelijärjestelmässä on käytössä moduulit, joita käytetään useimmissa partikkelitehosteissa. Nämä moduulit ovat Main-, Emission-, Shape- ja Renderer-moduulit. Main-moduulissa ovat kaikki tärkeimmät koko järjestelmää koskevat valinnat, kuten partikkeleiden elinaika, aloitusväri ja nopeus. Emission-moduuli puolestaan säätää sitä, kuinka paljon ja milloin partikkeleita luodaan. Shape-moduuli määrittää

partikkelijärjestelmän muodon, johon partikkelit syntyvät. Erilaisia muotoja on useita, mutta oletuksena muodoksi on valittu kartio. Renderer-moduuli taas määrittelee, mitä materiaalia partikkelit käyttävät. Lisäksi moduulissa määritellään partikkeleiden suuntaus ja se, miten partikkelit renderöidään. Ne voidaan renderöidä esimerkiksi 2D-kuvana tai 3D-mallina. Jo näiden neljän moduulin avulla voidaan saada aikaiseksi monenlaisia tehosteita.

Partikkelijärjestelmän moduulit eivät kuitenkaan lopu tähän. Monissa tehosteissa halutaan esimerkiksi partikkeleiden muuttavan väriä tai kokoa niiden elinajan aikana. Tällaisiin tapauksiin on tehty moduulit, joiden nimissä on sana ”Lifetime”. Niiden avulla voidaan värin ja koon lisäksi vaikuttaa esimerkiksi partikkeleiden liikkeeseen ja nopeuteen. Joskus halutaan partikkeleiden kiihdyttävän vauhtiaan syntymänsä jälkeen, jolloin voitaisiin käyttää Velocity over Lifetime -moduulia. Samalla periaatteella Lifetime-moduulien kanssa toimivat moduulit, joiden nimessä on sana ”Speed”. Ne muuttavat partikkeleiden ominaisuuksia niiden nopeuden perusteella elinajan sijaan.

Lisäksi partikkelijärjestelmässä on Trigger- ja Collision-moduulit, joiden avulla voidaan ottaa huomioon törmäyksiä. Partikkelit voidaan törmäysmoduulien avulla laittaa esimerkiksi katoamaan tai pomppimaan niiden törmätessä lattiaan. Partikkelijärjestelmässä on myös Lights- ja Trails-moduulit, joiden avulla voidaan laittaa partikkelit valaisemaan ympäristöään tai piirtää vana niiden kulkemaa reittiä pitkin. Partikkelijärjestelmillä voi olla myös alijärjestelmiä, jotka voidaan käynnistää esimerkiksi partikkeleiden kuollessa. Tällainen tehoste voi olla tarpeen esimerkiksi räjähtävien ammusten kanssa, jolloin räjähdystehoste on ammuksen alitehosteena ja laitetaan räjähtämään ammuksen osuessa johonkin. Partikkelijärjestelmä mahdollistaa myös partikkeleiden animoimisen Texture Sheet Animation -moduulin avulla.

Varjostimet Unity-pelimoottorissa

Unityssa voidaan kirjoittaa varjostinohjelmia kahdella eri tavalla. Ensimmäinen tapa on kirjoittaa niitä Unityn omalla pintavarjostin-tavalla (engl. Surface Shader). Tämä tapa on erityisen hyödyllinen, jos valon ja varjojen halutaan vaikuttavan varjostimeen. Pintavarjostin on Unityn oma varjostintyyppi, eikä sitä siksi tueta Unityn ulkopuolella. Se tarjoaa

yksinkertaisemman tavan tehdä monimutkaisia varjostimia, sillä muutamalla yksinkertaisella komennolla Unity luo automaattisesti tarvitsemansa koodin. [22.]

Toinen tapa on tehdä pikseli- ja kärkipistevarjostin (engl. fragment and vertex shader). Niitä on pintavarjostimiin verrattuna vaikeampi kirjoittaa, sillä koodia pitää kirjoittaa enemmän, mikäli haluaa varjostimen toimivan valojen kanssa. Yleisesti on siis suositeltavaa käyttää niitä vain, mikäli varjostimen ei tarvitse toimia valon kanssa tai varjostin on niin monimutkainen, ettei sitä pystyttäisi tekemään pintavarjostimella. Yleisesti ottaen pikseli- ja kärkipistevarjostimet ovat tehokkaampia ja ne tarjoavat paljon suuremman päätösvallan siitä, mitä ohjelmassa tapahtuu. Itse asiassa myös pintavarjostimet muunnetaan Unityn sisällä pikseli- ja kärkipistevarjostin muotoon. [22.]

Varjostimien kirjoittamiseen käytetään Unityn omaa ShaderLab-kieltä ja Cg/HLSL-kieltä. ShaderLab-kieltä käytetään varjostimen rakenteen organisointiin [20]. Kielellä määritellään esimerkiksi muuttujat, jotka tulevat näkyviin materiaali-ikkunaan, sekä varjostimen yleisiä ominaisuuksia, kuten esimerkiksi, mitä sekoitustapaa sen pitäisi käyttää. Itse varjostinkoodi kirjoitetaan cgprogram-sanan jälkeen Cg/HLSL-kielellä. [23.]

Muut ohjelmat

Tehosteiden luomiseen käytetään myös hieman Blender (www.blender.org) -nimistä ohjelmaa. Blender on ilmainen ja avoimeen lähdekoodiin perustuva 3D-kehitystyökalu. Sillä pystyy tekemään kaikkea, mitä 3D-kehityskanavassa voi kaivata, kuten mallintamaan ja animoimaan. Blender sopii hyvin pienille yrityksille ja yksittäisille henkilöille, jotka hyötyvät siitä, että kaikki on yhdessä sovelluksessa. Blenderiä käytetään, mikäli tehosteet vaativat 3D-mallin luomista. [24.]

Tehosteiden 2D-tekstuurien piirtämiseen käytetään Krita (www.krita.org) -nimistä sovellusta. Krita on ilmainen ja avoimeen lähdekoodiin perustuva piirtosovellus. Sillä pystyy tekemään hyvin konseptitaidetta, tekstuureja ja paljon muutakin. Siinä on esimerkiksi monia erilaisia siveltimiä, joilla kyvytönkin taitelija saa aikaiseksi vaikka mitä. Kritaa kehitetään yhdessä käyttäjien kanssa, minkä takia se vastaa käyttäjien tarpeita hyvin. [25.]

3.4 Tehosteiden käyttö Frog Shaman -pelissä

Loitsut ovat hyvin tärkeässä osassa Frog Shaman -pelin pelattavuutta. Niillä on tarkoitus taistella vihollisia vastaan ja ratkoa erilaisia arvoituksia, jotka vaativat tietyn elementtiyhdistelmän käyttöä. Taistelutapahtumassa myös viholliset pystyvät käyttämään samankaltaisia loitsuja kuin pelaaja, joten loitsut kykenevät satuttamaan niin pelaajaa kuin vihollistakin.

Loitsuja pitää myös kyetä tähtäämään jollain tapaa, sillä on haluttu, että esimerkiksi kiviloitsu kykenisi painamaan seinillä olevia nappeja. Tällöin kivi pitää tietenkin saada osumaan kyseiseen nappiin. Avuksi tähtäämiseen on mietitty tähtäyspistettä, joka sijaitsee ruudun keskellä. Tällöin pelaaja tietää, että loitsu tulee osumaan pisteen määrittelemään kohtaan, mikäli loitsun pituus riittää.

Pelissä on tärkeässä osassa loitsuyhdistelmien hyödyntäminen pelissä etenemiseen. Peliin on suunniteltu, että esimerkiksi isoja laatikoita voisi siirtää ainoastaan puhallusloitsun avulla. Lisäksi osaa laatikoista voisi siirtää vain, mikäli alusta olisi liukas. Se vaatisi, että maa olisi ensin jäädytetty. Lisäksi peli sisältäisi erilaisia poltettavia ja räjäytettäviä esteitä, mihin tarvittaisiin tuli- tai pommiloitsua.

Pelissä elementti valitaan tällä hetkellä valintanappia painamalla ja ohjaamalla ohjaimen tatti osoittamaan haluttua elementtiä. Valittu elementti ilmestyy käyttöliittymän alareunassa olevaan ruutuun osoittamaan, että se on nyt yhdistetty loitsuksi. Lisäksi käyttöliittymästä löytyy kuva, joka kertoo tällä hetkellä muodostetun loitsun.

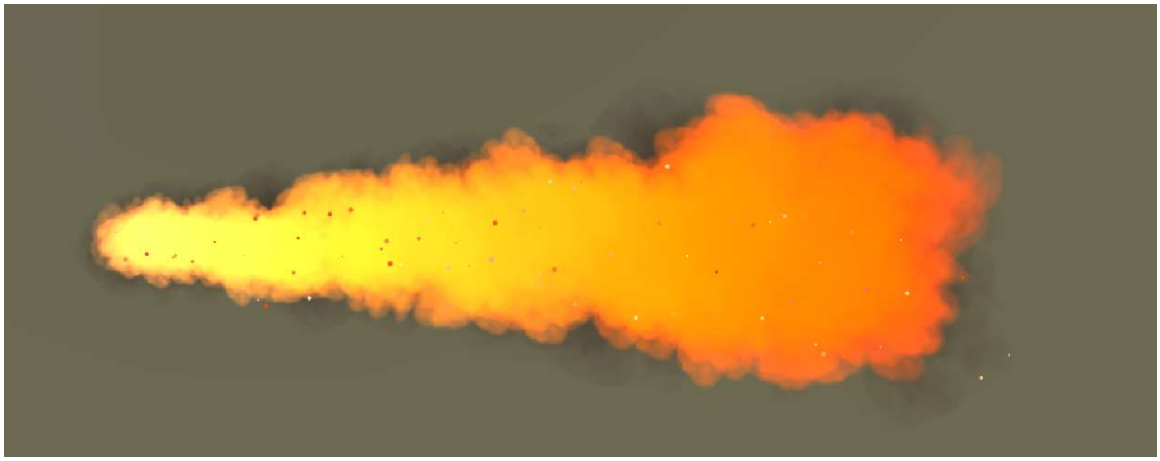
Peliin on myös suunniteltu loitsuenergia-mittari, joka näyttää paljonko loitsuenergiaa on tällä hetkellä käytössä. Jokainen loitsu kuluttaa aina tietyn määrän loitsuenergiaa, ja jos loitsuenergia on päässyt loppumaan, ei loitsua välttämättä pysty loitsimaan ollenkaan. Tässä tilanteessa täytyy odottaa loitsuenergian palautumista tai etsiä sitä palauttavia esineitä. Loitsuenergian käyttämiseen päädyttiin, sillä se estää pelaajia käyttämästä loitsuja loputtomasti. Tällä tavoin pelaaja pakotetaan pohtimaan hieman tarkemmin, mitä loitsuja missäkin tilanteessa kannattaa käyttää.

4 Partikkeli- ja elementtitehosteiden toteutus

4.1 Partikkelitehosteiden toteutus

Tulitehosteet

Tulitehosteiden toteutus aloitettiin hyvin varhaisessa vaiheessa insinöörityötä, sillä niistä löytyi paljon referenssimateriaalia. Lämpösuihkuloitsu toteutettiin käyttäen kolmea eri partikkelijärjestelmää, ja se näkyy kuvassa 10.



Kuva 10. Lämpösuihkuloitsu.

Näkyvimpänä järjestelmänä on itse lämpösuihku, jonka luomisessa hyödynnettiin Color over Lifetime- ja Size over Lifetime -moduuleja. Väri säädettiin muuttumaan kellertävästä punertavaksi partikkeleiden elinajan aikana. Myös alfa-arvo on alussa ja lopussa läpinäkyvä, jolloin partikkelit ilmestyvät ja häviävät sulavammin. Koko puolestaan muuttuu kasvavan käyrän mukaan pienestä suuremmaksi. Lisäksi järjestelmässä on käytetty valkoisia partikkeleita, joiden materiaaliin on lisätty punaista hehkua (engl. emission), joka saa partikkelin haaleammat reunat punertaviksi. Tämän avulla saatiin aikaiseksi tehosteen punertava reuna.

Lämpösuihkuloitsussa käytettiin lisäksi kahta muuta partikkelijärjestelmää, joista toinen on savujärjestelmä ja toinen hiukkasjärjestelmä. Savujärjestelmä näkyy lämpösuihkujärjestelmän takana hieman tummempana. Se käyttäytyy hyvin samalla tavalla kuin

lämpösuihku, mutta sen piirtoetäisyys laitettiin yhtä pienemmäksi, jotta se piirtyy aina lämpösuihkun taakse. Sen tarkoituksena on saada tuli näyttämään savuavalta. Hiukkasjärjestelmä puolestaan koostuu pienistä palavista hiukkaista, jotka piirretään lämpösuihkun päälle lisäämään loitsuun hieman liikettä. Hiukkaspartikkelit myös käyttävät Noise-moduulia, joka lisää hiukkasten liikkeeseen arvaamattomuutta.

Kaikki käytetyt partikkelijärjestelmät käyttävät lisäksi Collision-moduulia, jonka avulla estetään partikkeleita menemästä kappaleiden läpi. Collision-moduulia hyödynnetään myös osumapartikkelijärjestelmän sijainnin määrityksessä. Sijainti saadaan selville, kun moduulista aktivoidaan törmäysilmoitukset ja otetaan ne kiinni OnParticleCollision()-funktiolla, joka näkyy esimerkkikoodissa 1. Funktio eroaa hieman tavallisesta törmäysfunktiosta siinä, että partikkelien kohdalla törmäystiedot täytyy vielä erikseen hakea, jos tarvitaan tietoa itse törmäyksestä. Haussa täytyy määritellä, minkä partikkelijärjestelmän törmäyksiä haetaan ja minkä objektin kanssa törmäykset ovat tapahtuneet. Sen jälkeen saadaan tietoa esimerkiksi törmäyksen tapahtumakohdasta ja kohdan normaalista. Nyt osumatehoste voidaan siirtää tapahtumakohtaan. Tässä tapauksessa osumapartikkelijärjestelmänä toimii ylöspäin kohoava savuvana.

```
private void OnParticleCollision(GameObject other)
{
      ParticlePhysicsExtensions.GetCollisionEvents(particleEmitter, other,
      collisionEvents);
      hitSystem.transform.position = collisionEvents[0].intersection;
      hitParticlesSystem.Emit(1);
}
```

Esimerkkikoodi 1. Partikkelitörmäyksen tietojen haku ja törmäystehosteen paikan määritys tietojen perusteella.

Eräs kohdatuista ongelmista oli saada lämpösuihku mukailemaan hahmon liikkeitä. Jos partikkelijärjestelmät käyttivät paikallista koordinaatistoa simulointiavaruutenaan, ne pysyivät koko ajan muodossaan, mikä sai tehosteen näyttämään jäykältä. Toinen vaihtoehto oli käyttää simulointiavaruutena maailmakoordinaatteja, mutta hahmon liikkuessa lämpösuihkun muoto rikkoontui täysin. Rikkoontuminen korjaantui hiukan Inherit Velocity -moduulia käyttämällä, joka lisäsi pelihahmon liikettä partikkeleihin. Se ei kuitenkaan tuottanut tarpeeksi hyvää tulosta.

Lopulta ratkaisuksi tuli käyttää paikallista koordinaatistoa, mutta lisätä partikkeleihin liikettä manuaalisesti. Liikkeen lisäämiseen käytettiin esimerkkikoodia 2. Koodi pyörii Update()-funktion sisällä, eli sitä suoritetaan päivitysnopeuden mukaan. Koodissa seurataan liikkeen suuntaa ja suuruutta, ja sen avulla määritetään Force over Lifetime -moduulin x- ja y-kertoimet, jotka lisäävät kertoimen mukaista voimaa kyseisen akselin suunnassa. Myös y-akselin ympäri tapahtuvaa käännöstä seurataan ja lisätään x:n suuntaiseen liikkeeseen. Moduulissa lisätään voimaa tiettyyn suuntaan paikallisessa koordinaatistossa, jolloin partikkelit muodostavat kaaren, joka kääntyy liikkeen vastaiseen suuntaan.

```
Vector3 movement = transform.position - lastPos;
movement = transform.InverseTransformDirection(movement);

float xRot = transform.eulerAngles.y - lastRot.y;
xRot = Mathf.Lerp(-maxRotationTurn, maxRotationTurn,
     Mathf.InverseLerp(-maxAngle, maxAngle, xRot));

float x = Mathf.Lerp(-maxMovementTurn, maxMovementTurn,
     Mathf.InverseLerp(-maxMovementx, maxMovementx, movement.x));
float y = Mathf.Lerp(-maxMovementTurn, maxMovementTurn,
     Mathf.InverseLerp(-maxMovementy, maxMovementy, movement.y));

for (int i = 0; i < modules.Length; ++i)
{
     modules[i].xMultiplier = - (x + xRot);
     modules[i].yMultiplier = - y;
}
lastPos = transform.position;
lastRot = transform.eulerAngles;
```

Esimerkkikoodi 2. Liikkeen lisääminen partikkelijärjestelmään.

Tulipalloloitsun toteutuksessa käytettiin ainoastaan kahta eri partikkelijärjestelmää. Toinen kuvaa loitsun muotoa ja toinen lisää siihen eloisuutta. Lisäksi loitsun päättyessä luodaan osumakohdalle räjähdystehoste. Kuva 11 esittää loitsun eri vaiheita. Vaihe 1 kuvaa loitsun muotoa, kun tulipallo luodaan. Seuraavan kuvan tulipallo syntyy, mikäli tulipalloa pidellään kädessä eikä lähetetä heti matkaan. Liekit siis nousevat hieman ylemmäs ja muuttuvat tummemmiksi. Kolmas kuva kuvastaa tulipallon muodon muutosta, kun se muuntautuu kuvaksi 4, jollaisena tulipallo näkyy liikkuessaan kohti kohdettaan. Kuvassa 3 käynnistetään myös hiukkasefekti. Lopuksi pallo katoaa räjähdystehosteeseen.

Kuva 11. Tulipalloloitsun vaiheet.

Edellisen loitsun tapaan tulipallon muotoa muutetaan koodin kautta muuntamalla partikkelijärjestelmän Force over Lifetime -moduulin arvoja. Loitsun syntyessä voima suunnataan ylöspäin ja liikkeelle lähdettäessä liikkeen vastaiseen suuntan. Loitsussa käytetään myös samaa porrastettua väriarvoa Color over Lifetime -moduulissa kuin lämpösuihkuloitsussa. Tällä tavalla saadaan aikaiseksi hieman yhtenäisempi värimaailma. Huomattavaa on myös, että partikkelijärjestelmän Renderer-moduulissa on asetettu piirtojärjestys siten, että nuorin piirretään eteen, jolloin tehosteen värit eivät pääse sekoittumaan.

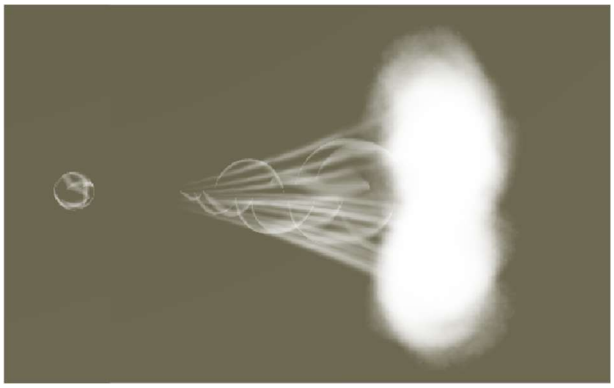
Kumpikin edellä mainituista loitsuista aiheuttaa palamista törmätessään palamisherkkiin kohteisiin, kuten vihollisiin ja poltettaviin esteisiin. Palaviin kohteisin lisätään liekit, jotka näkyvät kuvassa 12.

Kuva 12. Liekkitehoste, joka lisätään palaviin kohteisiin.

Liekit toteutettiin käyttäen samoja värejä kuin itse loitsutehosteissa. Liekeissä käytetään lisäksi Noise-moduulia, joka saa liekkien liikkeen arvaamattomaksi, sekä Size over Lifetime -moduulia, jonka avulla kutistetaan partikkeleita hiljalleen. Texture Sheet Animation -moduulin avulla muunnetaan partikkeleiden muotoa niiden elinajan aikana käyttäen neljää eri kuvaa. Moduulin avulla vaihdetaan partikkelin muotoa kuvasta toiseen sen elinajan aikana.

Ilmatehosteet

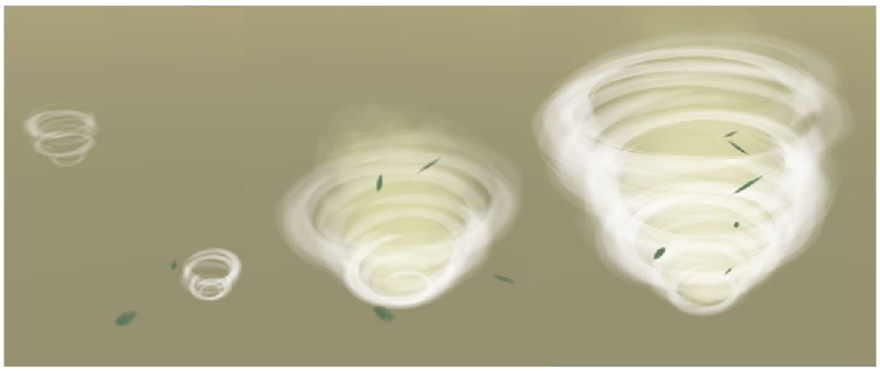
Ilmapuhallusloitsu alkaa pienestä tuulipallosta, joka häviää, kun loitsunapista päästetään irti. Tällöin kohdasta lähtee liikkeelle itse tuulipuhallus. Molemmista vaiheista on esimerkki kuvassa 13. Tuulipallo on toteutettu yhdellä partikkelijärjestelmällä. Partikkeleille ei anneta yhtään nopeutta, ja ne syntyvät kaikki yhteen paikkaan, jolloin syntyy yksi yhtenäinen muoto. Partikkeleissa käytetään tekstuuria, jossa on kaareva viiva. Tekstuurit piirtyvät pallonmuotoisen kappaleen päälle. Kun partikkeleita sitten pyöritetään niiden elinajan aikana ja annetaan sattumanvarainen alkurotaatio, saadaan aikaiseksi melko dynaaminen tuulipalloefekti.



Kuva 13. Tuulipuhallusloitsu.

Itse tuulipuhallus toteutettiin kolmea partikkelijärjestelmää käyttäen. Tärkeimmät ovat tuulipuhalluksen muodon luovat partikkelijärjestelmät. Ne ovat käytännössä aivan samanlaisia, mutta niiden rotaatiot on käännetty vastakkaisiin suuntiin. Muotoina käytettiin suuria pilvipartikkeleita, jotka käyttävät myös Trail-moduulia, joka näkyy kuvassa pitkinä viivoina. Viivojen on tarkoitus osoittaa ilman liikettä. Partikkelijärjestelmissä käytettiin myös Limit Velocity over Lifetime -moduulia, joka hidastaa partikkeleiden liikettä, kun ne kulkevat liian nopeasti. Tämän avulla saadaan pilvipartikkelit ensin liikkumaan nopeasti eteenpäin, mutta sen jälkeen hidastamaan vauhtiaan ja jäämään lopulta paikoilleen hetkeksi ennen katoamistaan. Kolmas partikkelijärjestelmä luo pyörivät pallopartikkelit, jotka ovat käytännössä samanlaisia kuin alussa olevassa tuulipallossa. Niiden on tarkoitus saada puhallus näyttämään hieman pyörivältä. Partikkelit toteutettiin palloina, sillä ne näyttivät hieman paremmilta eri suunnista katsottuina kuin tavalliset 2D-kuvat, joiden on pysyttävä samaan suuntaan suunnattuina riippumatta kameran liikkeestä.

Kahden elementin tuuliloitsuna on tornado. Kuva 14 kuvaa loitsun eri vaiheita. Syntyessään loitsu ilmestyy hahmon kädelle pienenä pyörteenä. Kun loitsu on valmis loitsittavaksi, se heitetään hahmon eteen, josta se sitten hiljalleen kasvaa täysikokoiseksi pyörteeksi. Hahmon on myös mahdollista nousta pyörteen päälle seisomaan ja ohjata pyörrettä mukanaan, jolloin hahmo saa korkeutta ja pystyy heittämään vihollisia pois tieltään. Pyörteeseen lisättiin myös toiminto, jossa tornado alkaa kasvaa tuulipuhallusloitsun osuessa siihen.

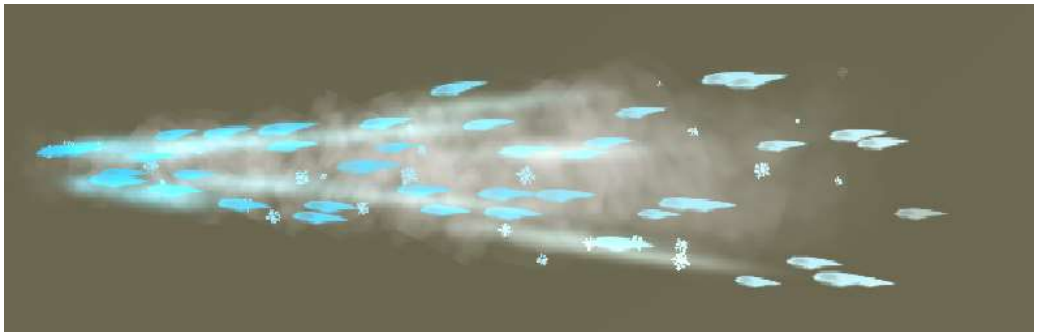


Kuva 14. Tornadoloitsun vaiheet.

Tornadoloitsu kostuu pääasiassa puolirenkaista, joita pyöritetään ympäri niiden elinajan aikana ja kasvatetaan niiden kokoa. Partikkelit käytävät Horizontal Billboard -renderöintitapaa, joka pitää partikkelit vaakatasossa. Hieman ongelmallista tässä on se, että pyörre katoaa lähes kokonaan näkyvistä, jos sitä katsoo vaakatasossa samalta tasolta. Tämän takia sekä muuten tunnelmaa luomaan pyörteessä on lisäksi kaksi muuta partikkelijärjestelmää. Ensimmäinen luo pyörteeseen lehtiä, jotka näyttävät nousevan ja kieppuvan pyörteen mukana. Lehdet saavat sattumanvaraisen ja kolmiulotteisen rotaation, minkä takia kaikki lehdet eivät näytä olevan suoraan kameraa kohti. Lisäksi järjestelmässä käytetään Velocity over Lifetime -moduulia, joka saa lehdet pyörimään ympyrää. Toinen taas luo pyörteen keskelle pölypilven, jonka tarkoituksena on elävöittää pyörrettä ja saada se näyttämään siltä, että se nostaa maasta pölyä. Se myös luo pyörteelle hieman väriä, mikä auttaa, jos sitä sattuu katsomaan vaakatasosta.

Jäätehosteet

Jääsuihkuloitsu on hyvin samankaltainen lämpösuihkuloitsun kanssa. Esimerkiksi muodoltaan ja toiminnaltaan ne käyttäytyvät samalla tavalla, mutta ne ovat toistensa vastaelementtejä. Jääsuihku kykenee sammuttamaan liekkejä ja lopettamaan palamisen. Jääsuihkun toteutukseen käytettiin kolmea eri partikkelijärjestelmää: jääpuikot, lumihiutaleet ja sumu. Partikkelijärjestelmät voidaan nähdä kuvasta 15.



Kuva 15. Jääsuihkuloitsu.

Sumu- ja lumihiutaletehosteiden toteutus oli hyvin samankaltaista tulitehosteiden kanssa, mutta jääpuikkojärjestelmä tarvitsi hieman enemmän pohdintaa. Jääpuikot ovat pitkulaisia 2D-kuvia, joiden on tarkoitus pysyä suuntautuneena menosuuntaa kohti. Tavallinen Billboard-renderöintitapa ei sovellu tähän tarkoitukseen, sillä se kääntää kuvia aina kameraa kohti. Se toimii hyvin muotojen kanssa, jotka ovat melko pyöreitä tai joiden suunnalla ei ole niin väliä. Jääpuikon kohdalla se kuitenkin tarkoittaa, että suoraan partikkelien syntymäkohdasta katsottuna jääpuikot ovat sivuttain eivätkä siis menosuuntaa kohti niin kuin pitäisi. Tämä kuitenkin korjaantui hyödyntämällä Stretched Billboard -renderöintitapaa, joka suuntaa partikkelit nopeuden suuntaan. Se myös mahdollistaa nimensä mukaisesti venyttämisen, mutta asetuksen voi jättää käyttämättä. Tämän renderöintitavan avulla jääpuikot kääntyvät kameraa kohti ainoastaan pituussuuntansa ympäri. Ainut huono puoli tässä tavassa on, että jääpuikkojen voi nähdä olevan 2D-kuvia, jos katsotaan suihkun päätysuunnista. Jääpuikot ovat kuitenkin niin pieniä ja nopeita, ettei se häiritse.

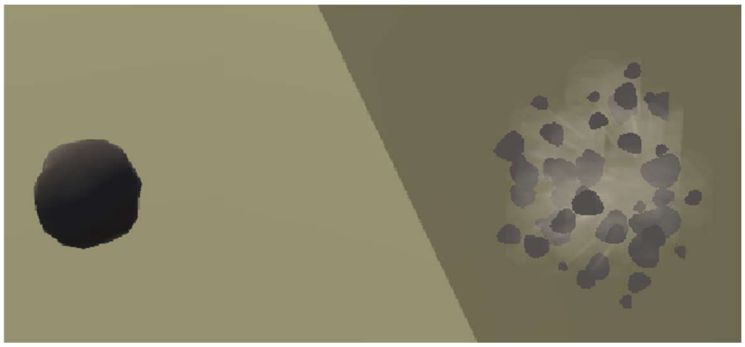
Jääaitaloitsu käyttää ensimmäisenä tehosteena 3D-malleja partikkeleinaan, mistä voi nähdä esimerkin alempana kuvassa 16. Jääaitaloitsussa pääosassa ovat maasta

kohoavat jääpiikit, jotka syntyvät maailmakoordinaatistoon partikkelijärjestelmän liikkuessa eteenpäin. Jos järjestelmä ei liikkuisi, piikit syntyisivät pienen neliön kokoiselle alueelle. Jääpiikkien suunnat kääntyvät satunnaisesti tiettyjen asteiden rajoissa, jolloin jokainen aita on hieman erilainen. Jääpiikkijärjestelmällä on myös kaksi alijärjestelmää: lumihiutalejärjestelmä ja sumujärjestelmä. Molemmat synnyttävät partikkeleita, kun jääpiikit syntyvät. Ne jäävät hetkeksi leijailemaan jääaidan ympärille, mutta katoavat hetken kuluttua jättäen pelkän aidan jäljelle. Aita jää näkyviin, kunnes sen sulamisaika on kulunut tai kunnes se tuhotaan tuliloitsujen avulla tai muilla vahvoilla voimilla.

Kuva 16. Jääaitaloitsu.

Maatehoste

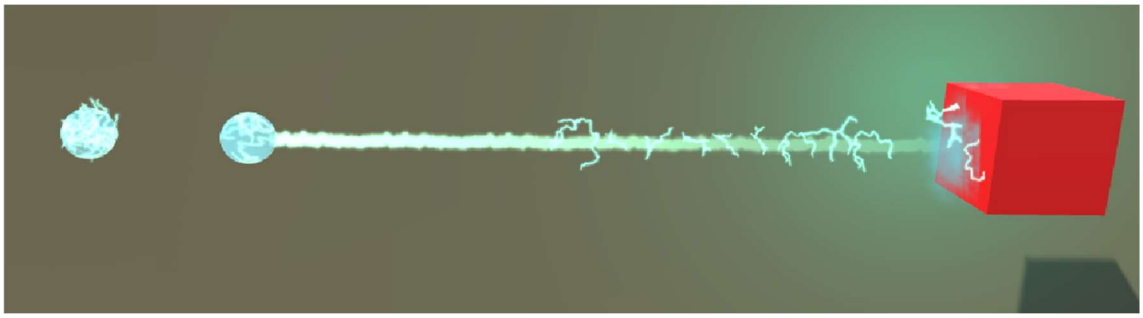
Ainut insinöörityön aikana toteutettu maaelementillinen loitsu oli kiviloitsu. Kiviloitsussa pääosassa on itse kivi, joka toteutettiin 3D-mallin avulla. Kiven liikkuessa sen vanaan syntyy pölyä ja pikkukiviä. Kivelle annetaan hieman ylöspäin suuntaava lähtövoima, jonka avulla kivi suuntaa ensin ylös ja alkaa sitten painovoiman vaikutuksesta pudota alaspäin. Johonkin osuessaan sen synnyttää osumatehosteena pienen räjähdyksen, jossa kivenpalasia ja pölyä lentää ympäriinsä. Räjähdystehoste on käännetty törmäyspisteen normaalin suuntaiseksi, jotta se osoittaa aina siitä pinnasta poispäin johon osutaan. Tästä on esimerkki kuvassa 17. Lisäksi kiviloitsuun toteutettiin ominaisuus, jossa kiviloitsu pystyy painamaan nappeja niihin osuessaan ja avaamaan tällä tavalla esimerkiksi suljettuja portteja. Kivi kykenee myös satuttamaan vihollisia niihin osuessaan.



Kuva 17. Kiviloitsu.

Sähkötehosteet

Yhden elementin sähköloitsu, eli salamaloitsu, oli yksi vaikeimmista loitsuista toteuttaa. Sen toteutuksesta ei ollut kunnollista visiota eikä referenssimateriaalia löytynyt kovin paljoa. Lopulta kuitenkin päädyttiin kuvassa 18 nähtävään tehosteeseen. Kuvan alussa näkyy pyöreä salamoiva pallo, joka osoittaa, että loitsu on ladattu kunnolla. Kun loitsun loitsii, syntyy pitkä suora, joka hakeutuu viholliseen, tässä tapauksessa punaiseen kuutioon. Suoraa pitkin lähtee liikkumaan pieniä salamoita synnyttävä partikkelijärjestelmä, joka luo salamoita sitä mukaa, kuin se etenee. Kun salamatehoste pääsee salaman kärkeen, luodaan paikalle osumatehoste, joka koostuu salamoista ja sinisestä savusta. Mikäli loitsua ei ladata riittävästi, on näkyvillä ainoastaan sininen pallo, mutta salamat puuttuvat. Jos loitsun loitsii tuolloin, syntyy pieniä salamoita ainoastaan pienelle matkalle loitsun edelle eikä isoa viivasalamaa luoda ollenkaan.



Kuva 18. Salamaloitsun vaiheet, kun loitsu on täyteen ladattu.

Suurimmaksi ongelmaksi muodostui viivasalaman pitkän muodon luominen. Salaman pitäisi myös hakeutua kohteeseen ja seurata kohteen liikettä, kunnes katoaa. Pitkän

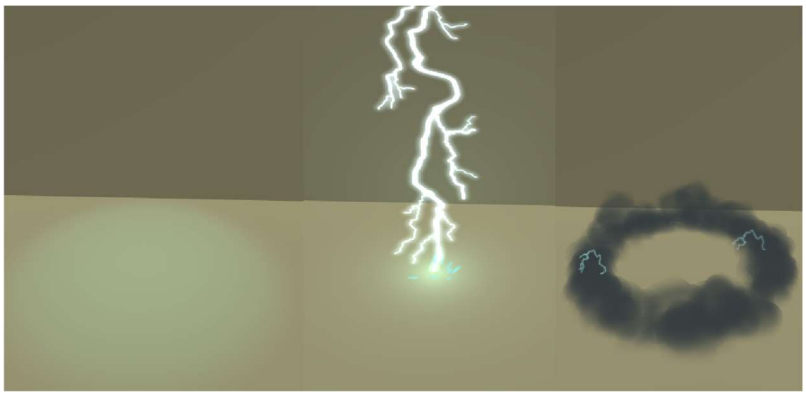
muodon ongelmana oli, että sen piti aina pituussuunnassa kääntyä kameraa kohti, mutta päädyistä katsottuna saattoi hyvin selvästi nähdä sen olevan litteä kuva. Hakeutuminen taas olisi ollut partikkeleilla hyvin vaikea toteuttaa, siksi lopulta päädyttiin käyttämään Line Renderer -komponenttia, joka piirtää viivan määritettyjen pisteiden välille. Koska pisteitä pystyi helposti muuttamaan suorituksen aikana, se oli selvästi järkevämpi vaihtoehto kuin partikkelien käyttäminen tässä tapauksessa. Tosin suoralla oli sama ongelma kuin partikkeleillakin: litteyden saattoi nähdä. Onneksi suoraan kuitenkin pystyi laittamaan tekstuurin ja muuttamaan sen väriä pisteiden välillä. Suora laitettiin myös alkamaan heti salamoivan pallon kyljestä, jolloin pallo peitti hyvin suoran litteän muodon.

Salamoivassa aloitustehosteessa käytettiin samaa tekniikkaa kuin tuulipuhalluksen aloitustehosteessa, eli tekstuurit laitettiin piirtymään pallon muotoisen kappaleen ympärille. Aloitustehosteessa on kaksi erilaista palloja luovaa järjestelmää, joista toinen luo ainoastaan sinisen värisiä läpinäkyviä palloja ja toinen luo palloja, joiden pinnalla liikkuu salamoita. Kun loitsua on ladattu tarpeeksi, käynnistetään aloituksen kolmas partikkelijärjestelmä, joka luo pallojen keskelle salamapartikkeleita, jotka muuttavat muotoaan Texture Sheet Animation -moduulin avulla. Lisäksi partikkelit käyttävät itse luotua varjostinta, joka saa salamat liikehtimään. Kun liikehdintä ja kuvien vaihtelu yhdistetään hyvin lyhyeen elinikään, syntyy melko aidonoloisia salamoita. Näitä partikkeleita käytetään myös suoraa pitkin liikkuvassa partikkelijärjestelmässä.

Kuvassa 19 on esimerkit kahden elementin salamaloitsun vaiheista. Ensin kentälle luodaan pyöreä valoalue, joka näyttää, mille alueelle salaman vaikutusalue ulottuu. Valon avulla pelaaja saa aikaa poistua vaikutusalueelta, jos hän on tarpeeksi nopea. Seuraavassa vaiheessa salama iskee maahan ja aiheuttaa alueelle salamoivan savupilven, joka lähtee leviämään.

Salamaa varten toteutettiin yksinkertainen varjostin, joka paljastaa salamaa ylhäältä alas. Liikettä ei juuri huomaa, sillä salamanisku on hyvin nopea, mutta se saa salaman näyttämään siltä, kuin se lähestyisi maata eikä vain ilmestyisi. Kätevintä oli se, että partikkelijärjestelmän Custom Data -moduulin avulla oli mahdollista antaa varjostimelle tietoa suoraan partikkelijärjestelmästä. Custom Data -moduuliin pystyi määrittämään esimerkiksi vektoriarvon ja laittamaan käyrän säätelemään sitä. Tällöin arvo muuttui käyrän perusteella partikkelin elinajan mukaan. Lisäksi Renderer-moduulissa täytyi aktivoida

Custom Vertex Stream -muuttuja, minkä jälkeen muuttujan alle avautui lista varjostimelle lähetettävistä muuttujista. Listaan piti lisätä Custom Data -moduulissa luotu muuttuja, minkä jälkeen arvoa pystyi ongelmitta käyttämään varjostimessa.

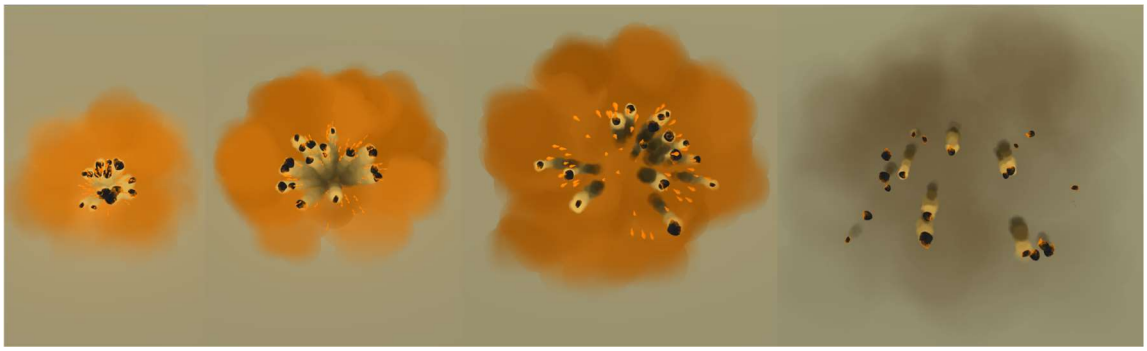


Kuva 19. Ison salamaloitsun vaiheet.

Monen elementin tehosteet

Pommiloitsu on maa- ja tulielementin yhdistelmäloitsu. Yhdistelmästä syntyy laavapommi, joka alkaa maahan osuttuaan sykkiä isommaksi ja pienemmäksi samalla muuttaen väriään kirkkaammaksi ja himmeämmäksi. Sykkiminen nopeutuu, kunnes pommi räjähtää, jolloin paikalle luodaan räjähdystehoste. Räjähdyksessä syntyy laavapisaroita, laavakiven palasia ja oranssina hehkuvaa savua. Nämä vaiheet voidaan nähdä kuvassa 20. Pommin sykkiminen on toteutettu varjostimen avulla, mutta räjähdys on tehty partikkelijärjestelmiä käyttämällä.

Räjähdystehosteessa on yhteensä käytetty viittä eri partikkelijärjestelmää. Itse räjähdyspilveen on käytetty yksinkertaista räjähdyspilvipartikkelia, jonka aloitusväri vaihtelee sattumanvaraisesti kahden värin välillä. Tämän avulla saatiin pilveen edes hiukan syvyysvaikutelmaa, sillä muutoin pilvi näytti tasaisen väriseltä, eikä muotoja juuri huomannut. Räjähdyksessä on lisäksi käytetty laavapisaroita, jotka lentävät kaaressa ulospäin räjähdyskohdasta. Laavapisarat käyttävät neljää eri tekstuuria, jotka vaihtuvat pisaran elinajan aikana. Lisäksi pisaroissa on käytetty Stretched Billboard -renderöintitapaa, joka saa ne pystymään menosuuntaa kohti kääntyneinä. Painovoimakertoimen avulla pisarat lentävät kaaressa ja putoavat lopulta kohti maata.



Kuva 20. Pommiloitsun räjähdystehosteen vaiheet.

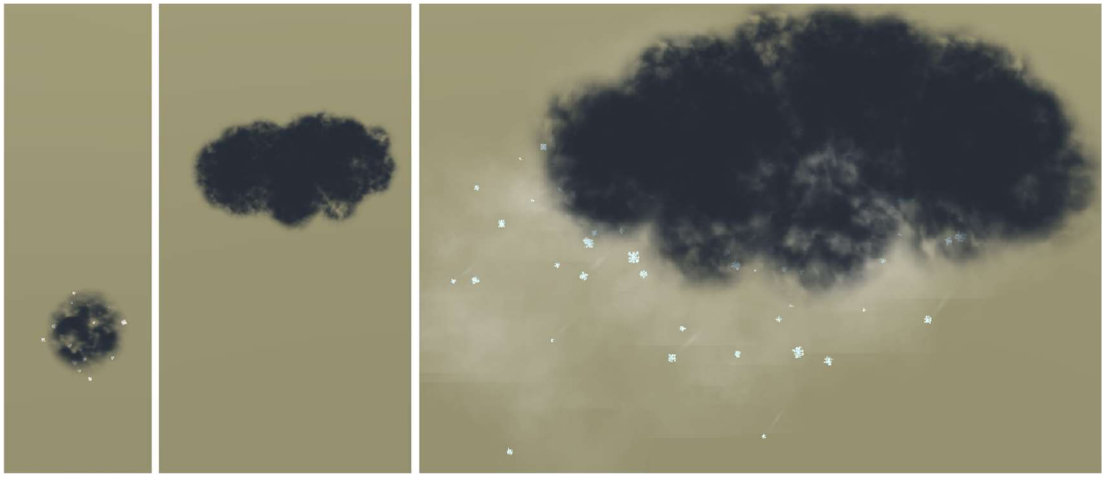
Pisaroiden lisäksi räjähdyksestä lentää laavapommin palasia. Palaset käyttävät myös Texture Sheet Animation -moduulia, mutta laavapisaroista poiketen ne käyttävät samaa tekstuuria koko elinikänsä. Elinajan aikana käytettävä tekstuuri on laitettu sattumanvaraiseksi, jolloin partikkelit käyttävät jotakin neljästä tekstuurista, mutta on todennäköistä, etteivät ne kaikki ole samanlaisia. Tämä on helppo tapa tehdä hieman vaihtelevuutta partikkeleihin, mutta silti käyttää samaa partikkelijärjestelmää. Jokaiselle partikkelille on myös lisätty alipartikkelijärjestelmä, joka piirtää savuvanaa palasten perään. Kun palaset osuvat maahan, ne jäävät paikalleen hetkeksi ennen katoamistaan.

Viidennen partikkelijärjestelmän tehtävänä on valaista räjähdys. Se käyttää Lights-moduulia, jossa on määritelty käytettävän valon voimakkuus ja laajuus. Color over Lifetime -moduulilla muutetaan valon väriä partikkelin elinajan aikana. Valo on toteutettu partikkelijärjestelmää käyttäen, koska sen avulla pystyy säätämään helposti valaistuksen kestoa ja valon voimakkuutta, eikä erillistä skriptiä kaivata.

Lumimyrskyloitsu kykenee jäädyttämään pelihahmoja jääsuihkuloitsun tapaan, mutta vaikuttaa yhtäaikaisesti laajemmalla alueella. Lumimyrskyloitsu syntyy pienestä pallomaisesta tehosteesta, joka lennätetään taivaalle, josta se leviää suureksi pilveksi. Kun pilvi on saavuttanut kokonsa, alkaa sataa lunta. Kuva 21 esittää näitä loitsun vaiheita.

Huomattavaa lumimyrskytehosteessa käytettävissä partikkelijärjestelmissä on esimerkiksi pilvipartikkelien toteutus. Ne käyttävät varjostinta, jossa käytetään kahta tekstuuria, joita liikutetaan eri nopeudella, ja ne on skaalattu erikokoisiksi. Mallina käytettiin Julian Loven mainitsemaa varjostinta, josta puhuttiin luvussa Tehosteiden suunnittelun neljä pilaria (s.12). Varjostimen avulla pilviin saatiin aikaiseksi monimutkaisempaa liikettä. Sen

avulla myös alkutehoste saatiin näyttämään pyörivältä, vaikka mitään liikettä itse partikkeleissa ei tapahdukaan. Alkutehosteen lumihiutaleet puolestaan ennakoivat pelaajalle, mitä tapahtuu, kun loitsu loitsitaan.



Kuva 21. Lumimyrskyloitsu.

Itse lumimyrsky tuotti aluksi pieniä ongelmia, sillä se ei näyttänyt tarpeeksi vaaralliselta. Loppujen lopuksi sumun ja sivulle suuntautuvan liikkeen avulla saatiin tehoste näyttämään vauhdikkaammalta ja lumimyrskymäisemmältä. Sivulle suuntautuva liike saa myrskyn näyttämään tuuliselta ja sumupartikkelit huonontavat näkyvyyttä.

Sumuloitsu on tulen ja jään yhdistelmäloitsu, jonka tarkoituksena on kätkeä pelaaja vihollisilta. Sumun haluttiin myös ilmestyvän pelaajan ympärille, mutta jättävän tyhjän aukon pelaajan läheisyyteen. Tämä aukko mahdollistaa vihollisten hyökkäämisen, jos pelaaja erehtyy liian lähelle. Sumun piti myös seurata pelaajaa, jottei synny tilannetta, jossa pelaaja pääsee poistumaan sumualueelta tahtomattaan.

Sumuloitsun toteutukseen käytettiin hyvin yksinkertaisia sumupartikkeleita. Ne luotiin suuren donitsin muotoon, mikä näkyy kuvassa 22. Donitsin muoto valittiin, koska sumuloitsun tarkoituksena on kätkeä pelaaja ja muodostua pelaajan ympärille. Donitsin muoto sisälsi myös valmiiksi aukon donitsin keskellä, ja aukon kokoa pystyi helposti muuttamaan. Itse partikkelijärjestelmä pidetään pelaajan lapsiobjektina, mutta partikkelit luodaan maailmakoordinaatistoon. Maailmakoordinaatistoon päädyttiin, koska sumun haluttiin pysyvän luontikohdassaan eikä seuraavan pelaajaa. Tämä kuitenkin edellyttää,

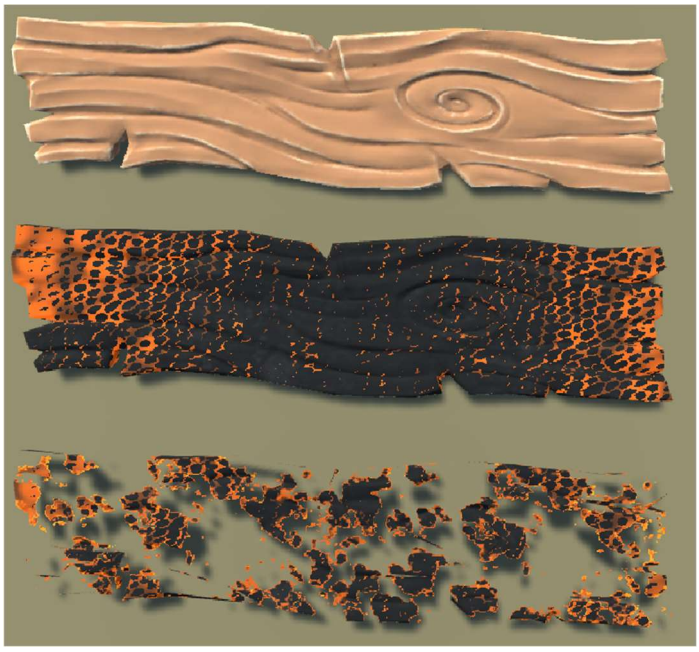
että luontinopeuden täytyy olla melko nopea ja partikkeleiden eliniän melko lyhyt, jottei partikkeleita ilmesty tarpeettoman paljon. Niitä pitää kuitenkin syntyä tarpeeksi, jotta pelaaja pystyy liikkumaan alueella ja näkemään edessään aina suunnilleen saman paksuisen sumuverhon.

Kuva 22. Sumuloitsu syntyy donitsin muotoon.

Tehosteeseen lisättiin myös lapsiobjekti, joka sisälsi Particle System Force Field -komponentin, jonka tarkoituksena on vaikuttaa lähistöllä oleviin partikkeleihin määritettyjen voimien avulla. Partikkelijärjestelmässä täytyy kuitenkin aktivoida External Forces -moduuli ja määrittää, mitkä voimakentät niihin voivat vaikuttaa. Lapsiobjektin voimakenttäkomponentissa voitiin määritellä vaikuttavan alueen koko ja muoto. Se säädettiin olemaan samankokoinen donitsin aukon kanssa. Lisäksi komponentissa käytettiin vetovoimaan vaikuttavia asetuksia siten, että siihen törmäävät partikkelit liikkuvat poispäin törmäyskohdasta. Tämä ominaisuus toteutettiin, jotta pelaajan ympärillä oleva aukko pysyisi koko ajan suunnilleen yhtä tyhjänä sumusta. Lisäksi sumu näyttää väistyvän pelaajan edestä, kun pelaaja liikkuu, mikä näyttää melko hyvältä.

4.2 Elementtitehosteiden toteutus

Liekkien lisäksi palaville kappaleille toteutettiin palamisvarjostin, jota kuva 23 havainnollistaa. Kuvassa on ylimpänä puinen lankku, joka syttyy hiljalleen palamaan, kun siihen on osuttu tulella. Keskimmäinen lankku kuvaa elementtiehosteen tilaa, kun se on muuttunut täysin palavaksi. Tällöin oranssilla hehkualueella tapahtuu pientä värin vaihtelua. Viimeisessä vaiheessa, kun lankku on palanut loppuun, se häviää palasina näkyvistä.



Kuva 23. Palamisvarjostimen vaiheet.

Elementtitehoste toteutettiin Unity-pelimoottorin pintavarjostimen avulla, sillä valon haluttiin vaikuttavan varjostinta käyttäviin kappaleisiin. Lisäksi pintavarjostin tukee GPU-ilmentymiä (engl. GPU instancing), joiden avulla samaa 3D-mallia ja materiaalia käyttävät kappaleet pystytään piirtämään saman piirtokutsun aikana. Jos palamista ei ole aktivoitu, varjostin suorittaa vain koodiesimerkissä 3 nähtävää koodia. Koodissa tex2D()-funktiolla haetaan tekstuurista väriarvoa syötteen mukana tulevan UV-arvon perusteella. UV-arvo antaa kyseisen pikselin koordinaatit nollan ja yhden väliltä vaaka- ja pystysuunnassa.

```
fixed4 c = tex2D (_MainTex, IN.uv_MainTex) * _Color;
fixed4 c2 = tex2D (_BurnedTexture, IN.uv_MainTex) * _Color;
o.Albedo = lerp(c.rgb, c2.rgb, burnAmount);
o.Normal = UnpackNormal(tex2D(_NormalMap, IN.uv_MainTex));
o.Alpha = c.a;
o.Smoothness = _Smoothness;
o.Metallic = _Metallic;
```

Esimerkkikoodi 3. Ote palamisvarjostimen surf()-funktiosta, jossa määritellään ulostulon arvoja.

UnpackNormal()-funktio puolestaan muuntaa tekstuurista saadut väriarvot normaaleiksi. Tekstuurina on käytetty normaalikarttaa, jonka tarkoituksena on luoda kappaleelle yksityiskohtia, joita siihen ei välttämättä ole luotu kärkipisteiden avulla. Esimerkiksi kuvassa

23 nähtävässä lankussa on käytetty normaalikarttaa puun uurteiden luomiseen. Koodissa haetaan myös palaneen tekstuurin väriarvo, mikä on oikeastaan käytössä vasta kun burnAmount-muuttujan arvo on nollaa suurempi. Lerp()-funktion avulla etsitään burnAmount-muuttujan arvon perusteella väriarvo tavallisen ja palaneen tekstuurin väliltä.

Jos palaminen tai häviäminen on alkanut, suoritetaan lisäksi esimerkkikoodissa 4 näkyvä koodi. Kuvassa 23 keskellä näkyvän lankun hohde luodaan antamalla Emission-muuttujalle arvo. Hohtava alue on määritelty tekstuurista saatavan alfa-arvon perusteella. Jos alfa-arvo on suurempi kuin materiaalin asetuksista määritetyn _EmissionCutoff-muuttujan arvo, on kyseinen kohta hohtava. Hohdon vahvuutta säädellään lisäksi noise- ja noise2-muuttujien arvoilla. Ne saavat arvonsa tekstuurista, joka määrittelee hohdon voimakkuuden tietyillä alueilla. Ensimmäisen muuttujan tex2D()-funktiossa liikutetaan tekstuurin UV-arvoja ajan mukaan, mikä saa palavan alueen näyttämään liikkuvalta.

```
fixed scrollX = _Time * _ScrollSpeedX;
fixed scrollY = _Time * _ScrollSpeedY;
fixed4 noise = tex2D(_EmissionStrengthMap, fixed2(scrollX, scrollY));
fixed4 noise2 = tex2D(_EmissionStrengthMap, IN.uv_EmissionStrengthMap);

half3 dissolveNoise = tex2D(_DissolveNoiseMap, IN.uv_DissolveNoiseMap);
dissolveNoise.r = lerp(0, 1, dissolveNoise.r);
cutoff = lerp(0, cutoff + _EdgeSize, cutoff);
half edge = smoothstep(cutoff + _EdgeSize, cutoff, clamp(dissolveNoise.r,
_EdgeSize, 1));

fixed4 em = tex2D(_EmissionMap, IN.uv_EmissionMap);
fixed4 emissionColor;
if (em.a > _EmissionCutOff)
{
     emissionColor = _EmissionColor * emissionAmount * noise2.a *
     noise.a;
}
else
{
     emissionColor = 0;
}
o.Emission = emissionColor + _EmissionColor * edge;

clip(dissolveNoise - cutoff);
```

Esimerkkikoodi 4. Ote koodista, joka suoritetaan, jos palaminen tai häviäminen on palavassa kappaleessa alkanut.

Koodissa suoritetaan myös kappaleen palasina häviäminen tekstuurin avulla. Katoamiseen käytetään mustavalkoista tekstuuria, josta mustat alueet katoavat ensimmäisenä ja valkoiset viimeisenä. Katoaminen suoritetaan clip()-funktiolla. Jos funktioon menevä arvo on negatiivinen, se leikkaa pikselin pois näkyvistä. Tässä tapauksessa tekstuurista

saatavasta arvosta vähennetään cutoff-nimisessä muuttujassa oleva arvo. Kun cutoff-muuttujan arvo saavuttaa arvon yksi, on koko kappale kadonnut näkyvistä. Koodissa luodaan lisäksi katoamisalueen reunat hohtaviksi.

Palamisvarjostimessa käytettyjen burnAmount-, emissionAmount- ja cutoff-muuttujien arvoja säädetään skriptien kautta. Näin niitä voidaan ajon aikana muokata halutulla tavalla ja saada aikaan sulavia muutoksia tilasta toiseen. Tämä toteutettiin käyttäen MaterialPropertyBlock-luokkaa, joka mahdollistaa materiaalin ominaisuuksien muuttamisen ajon aikana sekä sen, että samanlaiset kappaleet voivat käyttää samaa materiaalia, vaikka käyttävätkin eri arvoja muuttujissaan. Esimerkkikoodi 5 näyttää, mitä palavan objektin skriptissä täytyy olla, jotta arvon muuttaminen onnistuu.

```
MaterialPropertyBlock props = new MaterialPropertyBlock();
MeshRenderer meshRenderer = GetComponent<MeshRenderer>();
props.SetFloat("_EmissionAmount", emissionAmount);
meshRenderer.SetPropertyBlock(props);
```

Esimerkkikoodi 5. Ote palavassa kappaleessa olevasta skriptistä, jossa muutetaan varjostimen _EmissionAmount-muuttujan arvoa.

Jo edellä oleva esimerkki toimii ja mahdollistaa, että monta objektia pystyy käyttämään samaa varjostinta, vaikka niiden muuttujien arvot vaihtelevat. Pelin tehoja ajatellen on kuitenkin parempi hyödyntää Unityn GPU-ilmentymiä, jotka vähentävät tarvittavia piirtokutsuja. Sitä pystytään käyttämään erityisesti silloin, kun pelissä on samanaikaisesti useita objekteja, jotka käyttävät samaa mallia ja materiaalia. GPU-ilmentymät aktivoidaan materiaalin asetuksista, ja edellä olevankin koodin avulla piirtokutsut aluksi vähenevät. Muuttujien arvoja muutettaessa piirtokutsut kuitenkin lisääntyvät, sillä samaan piirtokutsuun yhdistetään ainoastaan samanväriset kappaleet ja muuttujien arvon muuttaminen muuntaa kappaleen väriä. Ne saadaan yhdistettyä samaan piirtokutsuun lisäämällä varjostimeen esimerkkikoodissa 6 nähtävät rivit. Rivien avulla määritetään muuttujat, joiden arvot voivat vaihdella kappaleiden välillä. [26.]

```
UNITY_INSTANCING_BUFFER_START(Props)
     UNITY_DEFINE_INSTANCED_PROP(fixed, _BurnAmount)
     UNITY_DEFINE_INSTANCED_PROP(fixed, _EmissionAmount)
     UNITY_DEFINE_INSTANCED_PROP(fixed, _Cutoff)
UNITY_INSTANCING_BUFFER_END(Props)
```

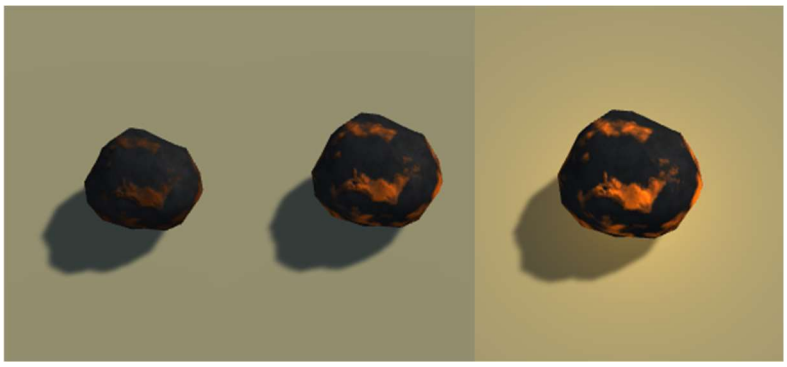
Esimerkkikoodi 6. Ote palamisvarjostimen ajon aikana muutettavien muuttujien määrittelystä.

Jotta muuttujien arvoja kyetään käyttämään varjostimen koodissa, täytyy ne ensin hakea. Esimerkkikoodi 7 näyttää, kuinka muuttujan arvo haetaan. Materiaalin ominaisuuksien määrittelyssä muuttujat voidaan lisäksi merkitä hakasulkuihin kirjoitetun PerRenderData-attribuutin avulla, jolloin ne katoavat materiaalin ominaisuuksista eikä niitä voi ilman skriptiä muuttaa.

```
fixed emissionAmount = UNITY_ACCESS_INSTANCED_PROP(Props, _EmissionA-
mount);
```

Esimerkkikoodi 7. Esimerkki ilmentymän arvon hakemisesta.

Laavapommia varten toteutettiin varjostin, jonka tehtävänä oli kasvattaa ja kutistaa pommia nopeutuvalla tahdilla. Samalla tahdilla säädetään myös oranssina hehkuvien alueiden kirkkautta. Pommiloitsun skriptissä on myös säädetty valon voimakkuutta tähän samaan tahtiin. Kuva 24 näyttää esimerkit laavapommin eri vaiheista.



Kuva 24. Laavapommi alkaa sykkiä maahan osuttuaan.

Varjostimen toiminta pohjautuu sinikäyrän toimintaan, sillä sen aaltomainen liike sopii hyvin sykkivän liikkeen toteuttamiseen. Lisäksi käyrän nopeutta, kokoa ja paikkaa on helppo säätää. Pommin koon muuttaminen toteutettiin varjostimen kärkipistefunktiolla, joka näkyy esimerkkikoodissa 8. Kärkipisteitä liikutetaan niiden normaalin suuntaan sinikäyrältä saadun arvon perusteella.

```
void vert(inout appdata_full v)
{
      float time = sin(UNITY_ACCESS_INSTANCED_PROP(Props, _Speed)) *
      _Amount * 0.1;
      v.vertex.xyz += v.normal * time;
}
```

Esimerkkikoodi 8. Laavapommin varjostimessa oleva kärkipistefunktio.

Varjostimen surf()-funktiossa hehkun määrää toteutetaan samalla periaatteella, mikä näkyy esimerkkikoodissa 9. Materiaaliasetuksissa on määritetty hehkulle minimi- ja maksimiarvot, joiden välillä sinikäyrän on tarkoitus liikkua. Sinifunktio on tässä tapauksessa täytynyt skaalata noiden arvojen välille kertomalla sinifunktio minimin ja maksimin keskiarvolla ja lisäämällä siihen sitten keskiarvon ja minimin summa.

```
float intensity = _IntensityMin;
float speed = UNITY_ACCESS_INSTANCED_PROP(Props, _Speed);
if (speed != 0)
{
      float sinMultiplier = (_IntensityMax - _IntensityMin) * 0.5;
      intensity = sin(speed) * sinMultiplier + sinMultiplier + _Intensi-
      tyMin;
}
```

Esimerkkikoodi 9. Laavapommiloitsun surf()-funktiossa säädetään hehkun voimakkuutta.

Kuten edellä olevista esimerkeistä saattaa huomata, tulee myös laavapommin _Speed-muuttujan arvo skriptin välityksellä ja se käyttää GPU-ilmentymiä. Skriptissä on muuttuja, jonka arvoa kasvatetaan Unityn sisäänrakennetun Time.deltaTime-muuttujan arvolla, joka kertoo edellisestä näytön päivityksestä kuluneen ajan. Lisäksi muuttuja kerrotaan itsellään, ennen kuin se lähetetään varjostimelle, sillä pelkän aika-arvon lähettäminen sai aikaan vain tasaista liikettä.

Itsellään kertominen oivallettiin oikeastaan vasta, kun pommiloitsua toteutettaessa kohdattiin ongelma, jossa jokainen käynnistetty pommi oli edellistä aina hiukan nopeampi. Alun perin varjostimessa yritettiin käyttää sisäänrakennettua _Time-muuttujaa, jota yleisesti pystytään käyttämään hyvin sinikäyrissä. Pommiloitsussa liikkeen tuli kuitenkin olla kiihtyvää, joten aikamuuttujaa kerrottiin skriptistä tulevalla nopeusmuuttujalla, jota kasvatettiin sielläkin Time.deltaTime-muuttujan avulla. Tämä sai aikaiseksi kiihtyvän liikkeen, mutta huomaamatta jäi yksityiskohta, joka kertoo, että _Time-muuttuja kertoo kulunutta aikaa pelin käynnistyksestä. Toisin sanoen, kun sinin sisällä kerrotaan kahta ajan kuluessa muuttuvaa lukua, voidaan sinin sisäinen funktio ajatella x^2-paraabelina. Paraabelissa nollasta kumpaankin suuntaan liikuttaessa aletaan lähestyä ääretöntä. Koska ääretöntä lähestytään sinin sisällä, on liike koko ajan kiihtyvää, ja mitä pidemmällä x-akselilla ollaan, sitä nopeampaa liike on. Jos pommi käynnistetään melkein heti, on luku melko pieni ja sykkiminen näyttää sopivan nopealta. Jos taas käynnistyksestä on

kulunut esimerkiksi 20 sekuntia, on kiihtyminen huomattavasti nopeampaa, sillä käyrällä on edetty huomattavasti pidemmälle. Tämän vuoksi päädyttiin käyttämään skriptin kautta laskettavaa ajan arvoa, joka alkaa aina nollasta. Samalla aika-arvo pystyttiin valmiiksi korottamaan toiseen.

Kaikkein monimutkaisin elementtitehoste, joka peliin yritettiin toteuttaa, oli jään maalaaminen maahan. Ajatuksena oli, että jääsuihkuloitsulla kykenisi maalaamaan liukasta jäätä maahan, johon osuessaan viholliset ja muut kappaleet alkaisivat liukua. Tätä varten toteutettiin IcePainter-skripti, jonka tarkoitus oli tehdä kaikki tarvittava työ. Skripti sisältää piirtomateriaalin, jonka varjostinta käytetään hyväksi piirto-operaation suorittamisessa. Se sisältää myös viittauksen kyseisessä objektissa käytettävään materiaaliin, jonka varjostimelle annetaan piirtotekstuuri, jota käytetään määrittämään, mihin jäätä piirretään. Alussa tekstuuri on tyhjä ja materiaali näyttää maalta.

Kun jääsuihkun partikkelit osuvat maahan, ne etsivät maakappaleesta IcePainter-skriptiä ja kutsuvat Paint()-funktiota, jolle lähetetään osumapisteen koordinaatit. Funktiosta on ote esimerkkikoodissa 10. Paint()-funktiossa luodaan pistettä kohti säde (engl. Raycast), jonka osumatiedoista saadaan irti kappaleen UV-koordinaatit, joihin osuttiin. Tieto lähetetään piirtomateriaalille, jota käytetään piirto-operaation suorittamiseen. Tämän jälkeen käytetään Graphics.Blit()-funktiota, jonka tehtävänä on kopioida lähdetekstuuri kohteena olevaan renderöintitekstuuriin (engl. RenderTexture) varjostinta käyttäen. Ensin tämänhetkisen piirtotekstuurin arvot kopioidaan väliaikaistiedostoon, jonka täytyy olla samankokoinen piirtotekstuurin kanssa. Seuraavaksi väliaikainen tekstuuri kopioidaan piirtotekstuuriin käyttäen piirtomateriaalia.

```
drawMaterial.SetVector("_Coordinate", new Vector4(hit.textureCoord.x,
hit.textureCoord.y, 0, 0));

RenderTexture temp = RenderTexture.GetTemporary(drawTexture.width, draw-
Texture.height, 0, RenderTextureFormat.ARGBFloat);

Graphics.Blit(drawTexture, temp);
Graphics.Blit(temp, drawTexture, drawMaterial);

RenderTexture.ReleaseTemporary(temp);
```

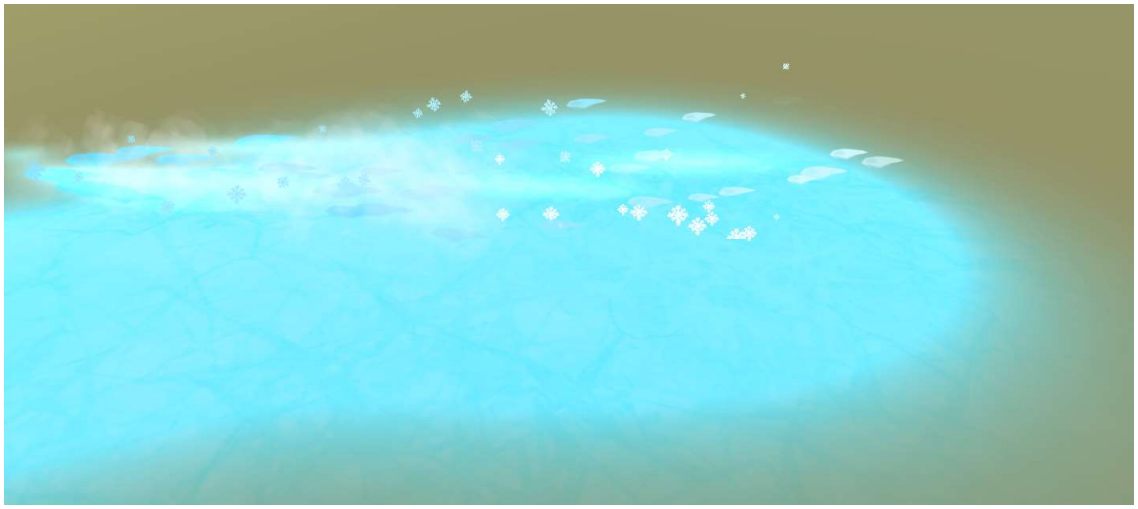
Esimerkkikoodi 10. Ote IcePainter-skriptin Paint()-funktiosta, jossa tekstuuriin maalaaminen toteutetaan.

Graphics.Blit()-funktio siirtää lähdetekstuurin piirtovarjostimen _MainTex-muuttujan arvoksi. Varjostimen varsinainen toiminta tapahtuu pikselifunktiossa, joka näkyy esimerkkikoodissa 11. Koodissa lisätään alkuperäiseen lähdetekstuurista saatuun väriarvoon uusi laskettu väriarvo ja varmistetaan saturate()-funktion avulla, että arvot pysyvät nollan ja yhden välillä. Laskemalla pikselin etäisyyttä skriptistä saadusta koordinaatista määritetään kyseisen pikselin väriarvo. Saatu väriarvo täytyy lisätä alkuperäiseen väriarvoon, jotta vanhat väritetyt alueet eivät katoaisi.

```
fixed4 frag (v2f i): SV_Target
{
      //Coming from Graphics.Blit() source
      fixed4 originalColor = tex2D(_MainTex, i.uv);
      float draw = pow(saturate(1 - distance(i.uv, _Coordinate.xy)),
      _DrawWidth);
      fixed4 drawColor = _Color * (draw * _Strength);
      return saturate(originalColor + drawColor);
}
```

Esimerkkikoodi 11. Ote piirtovarjostimen pikselifunktiosta.

Näiden funktioiden avulla saadaan aikaiseksi maan värittäminen jäätekstuurilla, mistä näkyy esimerkki kuvassa 25.



Kuva 25. Jääsuihku maalaa jäätä maahan.

Varsinaiseksi ongelmaksi muodostui lopulta väriarvon saaminen takaisin piirretystä tekstuurista, kun haluttiin selvittää, oliko pelihahmo jään päällä vai ei. Pelihahmoihin lisättiin IceCheck-luokka, jonka tarkoituksena oli lähettää säteitä kohti maata. Säteiden avulla

saatiin tieto maaobjektin UV-koordinaateista, joita käytettiin hyväksi väriarvon saamisessa. Väriarvon takaisin saaminen edellytti kuitenkin sitä, että renderöintitekstuurista piti lukea pikseliarvot 2D-tekstuuriin (engl. Texture2D). Renderöintitekstuuri ei sisältänyt funktioita, joita olisi voitu käyttää väriarvon takaisin saamiseen, 2D-tekstuuri sen sijaan sisälsi. Lukeminen osoittautui kuitenkin erittäin raskaaksi operaatioksi. Syyksi selvisi se, että lukufunktio joutui itse asiassa lukemaan arvot GPU:lta CPU:lle, mikä on hyvin hidasta. Samasta syystä renderöintitekstuuria ei pystynyt suoraan muuttamaan 2D-tekstuuriksi. Vaikka muuttaminen ei tuottanutkaan virheilmoituksia, ei tulos ollut halutunlainen, vaan tulokseksi saatiin pelkkiä harmaan arvoja sisältävä tekstuuri. Renderöintitekstuuri oli saanut väriarvonsa GPU:n puolella, mutta CPU:n puolella oleva viite siihen pysyi tyhjänä, eli siis harmaana. Ainoa löydetty keino oli siis pikseleiden lukeminen 2D-tekstuuriin.

Insinöörityön aikana toteutettiin myös muutamia partikkeleille tarkoitettuja varjostimia. Niillä on muutamia erityispiirteitä, jotka kannattaa ottaa huomioon. Esimerkkikoodissa 12 näytetään eräässä partikkelivarjostimessa käytettyjä asetuksia. Niistä oleellisimmat ovat piirtojärjestyksen määrittävä Queue-tunniste, joka määrittää partikkelit piirrettäväksi läpinäkyvien kappaleiden yhteydessä. Blend-asetus puolestaan määrittää käytettävän sekoitustavan. Cull Off -asetuksen avulla partikkelit piirretään molemmilta puolilta. ZWrite Off -asetus puolestaan määrittää, ettei partikkelin pikseleitä piirretä syvyyspuskuriin. Tätä käytetään yleensä, kun piirretään osittain läpinäkyviä efektejä. Lighting-asetus puolestaan määrittää, ettei valo vaikuta partikkelien väriin. [27.]

```
Tags
{
      "Queue" = "Transparent"
      "RenderType" = "Transparent"
      "IgnoreProjector" = "True"
      "PreviewType" = "Plane"
}
Blend SrcAlpha OneMinusSrcAlpha
Cull Off
Lighting Off
ZWrite Off
LOD 200
```

Esimerkkikoodi 12. Ote partikkelivarjostimen asetuksista.

Lisäksi partikkelivarjostimessa täytyy mahdollisesti ottaa huomioon myös partikkelijärjestelmän antamat tiedot. Jos partikkelijärjestelmässä käytetään Custom Vertex Stream

-asetusta ja sinne määritetään lisää ominaisuuksia, joita varjostimen halutaan käyttävän, ne täytyy ottaa huomioon myös varjostimen puolella. Varjostimen kärkipistefunktion syötteeseen täytyy määrittää vastaavat arvot, jotka on partikkelijärjestelmän puolella määritetty, jotta niitä voidaan varjostimessa käyttää.

5 Tehosteiden arviointi

5.1 Toteutetut tehosteet

Insinöörityössä toteutettiin yhteensä 12 eri loitsutehostetta, eli juuri niin monta kuin alun perin arvioitiin. Toiveena oli tietysti toteuttaa tehosteita enemmänkin, mutta aika ei riittänyt niiden toteuttamiseen. Useat tehosteet veivät oletettua enemmän aikaa, mikä rajoitti toteutettujen tehosteiden määrää. Ongelmana oli myös, ettei kaikkia tehosteita ollut suunniteltu vielä aivan valmiiksi, joten niiden toteuttamista ei vielä haluttu aloittaa, jos ne joudutaankin myöhemmin muuttamaan täysin. Oli siis parempi aloittaa kaikkein selkeimmistä loitsuista, joita tiedetään pelissä kaivattavan.

Useimmat tehosteista saatiin sellaiseen kuntoon, että niitä voidaan pelissä sellaisenaan käyttää ainakin, kunnes pelin visuaalinen ilme alkaa hahmottua paremmin. Silloin käytettyjä tekstuureja saatetaan muuttaa, mutta tehosteiden rakenne pysyy todennäköisesti melko samana. Muutamien loitsujen ulkonäkö jäi melko huonoksi, joten varsinkin niiden tekstuurit ja tyylit tulevat jossain vaiheessa muuttumaan. Hyvänä esimerkkinä on ainakin tuulipuhallusloitsu, jonka elementit eivät aivan sovi yhteen, ja siihen pitäisi jollain tapaa saada lisää rakennetta. Myös esimerkiksi räjähdystehosteessa käytetyt savupilvet jäivät todella muodottomiksi, joten niitä täytyy myöhemmin vielä korjata. Tosin muutamat tehosteet onnistuivat myös ajateltua paremmin: esimerkkinä tulisuihkuloitsu, jonka visio ei aluksi ollut kovin selkeä, mutta oikea tyyli löytyi kokeilemalla.

Loitsuihin toteutettiin myös useita niihin liittyviä toiminnallisuuksia. Esimerkiksi tornadoloitsuun lisättiin ominaisuus, jossa pelaaja kohoaa tornadon huipulle, jos hän astuu sen lähelle tai hyppää siihen. Kun pelaaja on huipulla, hän pystyy ohjaamaan tornadoa ja voi halutessaan hypätä tornadosta pois, jolloin tornado jatkaa matkaansa ja katoaa. Tornadoon osuessaan viholliset lentävät kauemmaksi. Myös esimerkiksi tuli- ja jääloitsut

laitettiin toimimaan keskenään. Jos jääloitsulla loitsii kohti liekkejä, alkavat liekit savuta ja lopulta sammuvat. Myös palamisvarjostimen hehkuminen lakkaa, jos tuli on kappaleessa sammunut. Vastaavasti tulisuihku kykenee sulattamaan jääaitaloitsun muodostaman aidan sekä jääsuihkun vihollisiin luomia jääpaloja. Lisäksi kivi- ja sähköloitsut laitettiin aktivoimaan asioita, esimerkiksi portteja.

Osasta toteutetuista tehosteista jäi kuitenkin uupumaan muutamia ominaisuuksia, jotka olisi haluttu toimiviksi. Esimerkiksi liekkien leviämistä ei vielä ehditty toteuttaa kunnolla. Suurimmaksi esteeksi tuli vision puute siitä, kuinka sitä pitäisi lähteä toteuttamaan, joten se jätettiin myöhemmälle eikä sitä loppuen lopuksi projektin aikana ehditty toteuttaa. Lisäksi aivan insinöörityön viimeisenä kokeiluna toteutettu jään maalaamistehoste saatiin toimimaan jotenkin, mutta tämänhetkinen toteutus saattaa valmiiseen peliin olla liian raskas, sillä se vei suuren osan yhden näytön päivitykseen kulutetusta ajasta. Tämän vuoksi se täytyy todennäköisesti koettaa tehdä toisella tavalla, esimerkiksi maalaamalla tekstuuriin ilman varjostinta skriptin kautta, mutta tämän tavan tehokulutuksesta ei ole tietoa. Voi myös olla, ettei jään maalaamista kyetä toteuttamaan, vaan se on loppujen lopuksi toteutettava muilla keinoilla.

5.2 Parannettavaa

Insinöörityössä toteutettuihin tehosteisiin oltiin ihan tyytyväisiä, mutta parannettavaakin jäi jonkin verran. Osaan tehosteista olisi voinut kuluttaa hieman enemmän aikaa, jolloin niiden lopputulos olisi todennäköisesti ollut hieman paremman näköinen. Tosin tällöin ei olisi saatu yhtä monta tehostetta toteutettua. Myös tehosteiden suunnitteluun olisi voitu käyttää hieman enemmän aikaa. Melko monta tehostetta lähdettiin toteuttamaan ilman kunnollista visiota siitä, miltä tehosteen kuuluisi valmiina näyttää. Tämä johti siihen, että muutamaakin tehostetta jouduttiin moneen otteeseen muuttamaan ja parantelemaan, mikä puolestaan söi aikaa muiden tehosteiden toteutuksesta. Kunnollisella suunnittelulla ja referenssiaineiston hankkimisella tältä olisi todennäköisesti vältytty.

Paljon aikaa kului myös moniin kokeiluihin, joita ei sitten loppujen lopuksi kannattanutkaan yritetyllä tavalla toteuttaa ja ne piti projektista poistaa. Tosin näistä virheistä opittiin, eikä niitä enää toivottavasti tulevaisuudessa toisteta. Lisäksi tekemättä jääneet

ominaisuudet jäivät häiritsemään, sillä ne olisi ollut hyödyllistä saada tämän projektin aikana tehtyä.

5.3 Tulevaisuus

Loitsujen toteutuksessa päästiin hyvään vauhtiin, ja jatkossa pystytään helposti jatkamaan ja parantamaan insinöörityössä toteutettuja tehosteita. Kun pelin visuaalinen tyyli hieman kehittyy, voidaan partikkelijärjestelmien ja varjostimien käyttämiä tekstuureja helposti vaihtaa toisiin. Toiminnallisuus on monissa toteutettu valmiiksi asti, jolloin voidaan alkaa suunnitella arvoituksia, joiden ratkaisemiseen loitsuja voidaan käyttää ilman suurempia muutoksia.

Jäljelle jäi vielä kahdeksan loitsua, jotka täytyy suunnitella ja toteuttaa. Niiden toteuttaminen käy todennäköisetsi helpommin ja mukavammin, kun jo toteutettuja loitsuja voidaan käyttää malleina. Lisäksi moniin ongelmatilanteisiin on valmiit ratkaisut ja virheitä on mahdollista välttää paremmin. Joissain partikkelijärjestelmissä on myös mahdollista käyttää samoja partikkeleita, joita on jo kehitetty.

6 Yhteenveto

Insinöörityössä tutustuttiin paremmin Unity-pelimoottorin partikkelijärjestelmiin ja varjostimiin ja toteutettiin niiden avulla toimivia loitsutehosteita Frog Shaman -nimiseen 3D-seikkailupeliin. Pelin tehosteista saatiin lopulta toteutettua 12 loitsua 20:stä.

Insinöörityön tutkimusvaiheessa perehdyttiin partikkeli- ja elementtitehosteiden historiaan, joka auttoi ymmärtämään niiden toimintaa paremmin. Lisäksi perehdyttiin toisiin samankaltaisiin peleihin, joissa käytetään paljon erilaisia loitsutehosteita. Niistä saatiin aikaiseksi paljon referenssimateriaalia, jota pystyttiin käyttämään hyväksi toteutettujen tehosteiden suunnittelussa. Niiden avulla heräsi myös ajatuksia siitä, millaisia komponentteja tehosteet saattaisivat tarvita. Insinöörityössä tutkittiin myös tarkemmin loitsujen luonnissa käytettäviä elementtejä, joita olivat tuli, ilma, jää, maa ja sähkö.

Loitsujen toteutuksessa pyrittiin toteuttamaan pelin suunnitteluvaiheessa määritetyt ominaisuudet loitsuille. Toteutus aloitettiin selkeimmistä ja pelissä suurimmassa osassa olevista loitsuista, kuten loitsuista, joiden pelin sisäisessä luomisessa käytetään ainoastaan yhtä elementtiä. Tällaisia loitsuja olivat esimerkiksi tulipallo- ja tulisuihkuloitsut.

Toteutetut loitsut todettiin toimiviksi, mutta myös parantamisen varaa jäi. Suurimmasta osasta loitsuja saatiin toteutettua kaikki vaaditut ominaisuudet, kuten niiden ulkonäkö ja toiminnallisuus. Muutamista loitsuista jäi kuitenkin osia toteuttamatta, sillä niiden toteuttamiseen ei jäänyt tarpeeksi aikaa. Lisäksi pelin visuaalinen ilme muuttuu tulevaisuudessa jonkin verran, joten loitsujen ulkonäkö saattaa muuttua. Suurimmassa osassa muutokseksi riittää todennäköisesti ainoastaan partikkelijärjestelmissä ja varjostimissa käytettyjen tekstuurien muuttaminen.

Insinöörityötä tehdessä partikkelijärjestelmien ja varjostimien käyttö tuli tutummaksi ja niiden käytöstä opittiin paljon uutta. Loitsujen toteutuksen aikana kohdattiin myös monenlaisia ongelmia, joista useimpiin löydettiin ratkaisut, joita voidaan tulevaisuudessa käyttää hyväksi samankaltaisiin ongelmiin törmättäessä. Toteutetuista loitsutehosteista on myös paljon hyötyä, sillä ne ovat isossa osassa pelin pelattavuutta.

Lähteet

1 Particle Systems. Verkkoaineisto. Unity. <https://docs.unity3d.com/Manual/ParticleSystems.html>. Luettu 9.2.2019.

2 Landsteiner, Norbert. 2016. SpaceWar!. Verkkoaineisto. Masswerk. <https://www.masswerk.at/spacewar/>. Luettu 11.2.2019.

3 Asteroids – The classic arcade game. Verkkoaineisto. Classic Gaming. <http://www.classicgaming.cc/classics/asteroids/>. Luettu 11.2.2019.

4 Reeves, William. 1983. Particle Systems – A Technique for Modeling a Class of Fuzzy Objects. Verkkoaineisto. ACM Transactions on Graphics. Vol. 2, s. 91–108. <https://cal.cs.umbc.edu/Courses/CS6967-F08/Papers/Reeves-1983-PSA.pdf>. Luettu 11.2.2019.

5 Particle Systems. 2008. Verkkoaineisto. Electronic visualization laboratory. <https://www.evl.uic.edu/aej/527/lecture05.html>. Päivitetty 28.1.2008. Luettu 11.2.2019.

6 Hergaarden, Mike. 2011. Graphics Shaders. Verkkoaineisto. VU Amsterdam. <https://www.cs.vu.nl/~eliens/download/literatuur-shaders.pdf>. Luettu 19.2.2019.

7 Failes, Ian. 2018. RenderMan at 30: A Visual History. Verkkoaineisto. VFXV The Magazine of visual effects society. <http://vfxvoice.com/renderman-at-30-a-visual-history/>. Luettu 19.2.2019.

8 History of OpenGL. 2017. Verkkoaineisto. OpenGL. <https://www.khronos.org/opengl/wiki/History_of_OpenGL>. Päivitetty 11.9.2017. Luettu 19.2.2019.

9 Cg (programming language). 2018. Verkkoaineisto. Wikipedia. <https://en.wikipedia.org/wiki/Cg_(programming_language)#cite_note-2>. Päivitetty 3.8.2018. Luettu 19.2.2019.

10 Mages of Mystralia. Verkkoaineisto. Borealys Games Inc. <https://game.magesofmystralia.com/>. Luettu 12.2.2019.

11 Magicka. Verkkoaineisto. Arrowhead. <http://arrowheadgamestudios.com/games/magicka/>. Luettu 25.2.2019.

12 Elements/Details. 2019. Verkkoaineisto. GamePedia. <https://magicka.gamepedia.com/Elements/Details>. Päivitetty 25.1.2019. Luettu 25.2.2019.

13 What are the 4 elements? Verkkoaineisto. Home Science tools. <https://learning-center.homesciencetools.com/article/four-elements-science/>. Luettu 20.2.2019.

14 Harris, Tom. How Fire Works. Verkkoaineisto. HowStuffWorks. <https://science.howstuffworks.com/environmental/earth/geophysics/fire.htm>. Luettu 20.2.2019.

15 Why Does Wind Blow. Verkkoaineisto. SciJinks. <https://scijinks.gov/wind/>. Luettu 22.2.2019.

16 Brain, Marshall & Lamb, Robert. How Tornados Work. Verkkoaineisto. HowStuffWorks. <https://science.howstuffworks.com/nature/climate-weather/storms/tornado.htm>. Luettu 22.2.2019.

17 Lightning. Verkkoaineisto. National Geographic. <https://www.nationalgeographic.com/environment/natural-disasters/lightning/>. Luettu 22.2.2019.

18 Love, Julian. 2013. Technical Artist Bootcamp: The VFX of Diablo. Verkkoaineisto. GDC Vault. <https://www.gdcvault.com/play/1017660/Technical-Artist-Bootcamp-The-VFX>. Luettu 23.2.2019.

19 Unity (game engine). 2019. Verkkoaineisto. Wikipedia. <https://en.wikipedia.org/wiki/Unity_(game_engine)>. Päivitetty 21.2.2019. Luettu 24.2.2019.

20 GameObjects. Verkkoaineisto. Unity. <https://docs.unity3d.com/Manual/GameObjects.html>. Luettu 24.2.2019.

21 Using Particle Systems in Unity. Verkkoaineisto. Unity. <https://docs.unity3d.com/Manual/PartSysUsage.html>. Luettu 24.2.2019.

22 Writing Shaders. Verkkoaineisto. Unity. <https://docs.unity3d.com/Manual/ShadersOverview.html>. Luettu 25.2.2019.

23 ShaderLab Syntax. Verkkoaineisto. Unity. <https://docs.unity3d.com/Manual/SL-Shader.html>. Luettu 25.2.2019.

24 The software. Verkkoaineisto. Blender. <https://www.blender.org/about/>. Luettu 25.2.2019.

25 What is Krita? Verkkoaineisto. Krita. <https://docs.krita.org/en/Krita-FAQ.html#what-is-krita>. Luettu 25.2.2019.

26 GPU instancing. Verkkoaineisto. Unity. <https://docs.unity3d.com/Manual/GPUInstancing.html>. Luettu 19.3.2019.

27 ShaderLab: Culling & Depth Testing. Verkkoaineisto. Unity <https://docs.unity3d.com/Manual/SL-CullAndDepth.html>. Luettu 20.3.2019.