KARELIA-AMMATTIKORKEAKOULU
Tietojenkäsittelyn koulutus

Sami Timonen

ASEGENERAATTORI UNREAL ENGINESSÄ

Opinnäytetyö
Kesäkuu 2020

Karelia AMMATTIKORKEAKOULU

OPINNÄYTETYÖ
Helmikuu 2020
Tietojenkäsittelyn koulutus

Tikkarinne 9
80200 JOENSUU
+358 13 260 600 (vaihde)

Tekijä
Sami Timonen

Nimeke
Asegeneraattori Unreal Enginessä

Toimeksiantaja
Kollektiwe Oy

Tiivistelmä

Tämä opinnäytetyö käsittelee virtuaalisen ampuma-aseen satunnaista generointia ja siihen liittyvää kehitystyötä. Asegeneraattori oli toimeksianto KollektiWe Oy:ltä, ja sen tarkoituksena oli selvittää virtuaalisen aseen pilkkomista pienempiin osiin generaattoria varten, sekä itse generaattorin ohjelmoiminen. Opinnäytetyössä eritellään virtuaalisen aseen tarvitsemia osia ja rakenteita, joiden toimintaa ja ominaisuuksia tarkastellaan syvemmin.

Opinnäytetyö toteutettiin Unreal Engine -pelimoottorilla, ja se tuli toteuttaa yhteensopivaksi virtuaalitodellisuusympäristön kanssa vaatien näin virtuaalilasit toimiakseen. Asegeneraattori täytyi toteuttaa mahdollisimman helppokäyttöiseksi ja muokattavaksi.

Virtuaalinen ase vaatii vain muutamia ominaisuuksia, että se toimii oikean aseen kaltaisesti. Lisäominaisuudet antavat mahdollisuuksia tasapainoisemmalle ja monimuotoisemmalle generoinnille. Visuaalisesti virtuaalista asetta voi pilkkoa lähes loputtomiin, mutta on löydettävä kultainen keskitie visuaalisten variaatioiden ja toteutuksen vaativuuden väliltä.

Kieli
Suomi

Sivuja 49
Liitteet 0
Liitesivumäärä 0

Asiasanat
Virtuaaliase, Generaattori, Unreal Engine, Satunnaisuus

Karelia
UNIVERSITY OF APPLIED SCIENCES

THESIS
April 2020
Business Information Technology

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

Author
Sami Timonen

Title
Gun generator in Unreal Engine

Commissioned by
Kollektiwe Oy

Abstract

This thesis examines a random generation of a virtual gun and its development process. The gun generator was an assignment from KollektiWe Oy and its purpose was to breakdown a virtual gun for the gun generator and programming of the generator itself. The thesis differentiates virtual guns pieces and structures, which are then examined more closely.

The implementation was made with the Unreal Engine game engine. The gun generator also had to be compatible with virtual reality and therefore work with a virtual reality headset. The gun generator had to be made as user friendly and easy to modify as possible.

A virtual gun needs only a few attributes to work like a real gun. Additional attributes give opportunities for a more balanced and a more diverse gun generation. Visually a virtual gun can be split almost endlessly, but a balance must be found between visual variations and implementation.

Language
Finnish

Pages 49
Appendices 0
Pages of Appendices 0

Keywords
Virtual Gun, Generator, Unreal Engine, Randomness

Sisältö

1 Johdanto

Asegeneraattori on työkalu, jonka tarkoituksena on luoda erilaisia ampuma-aseita videopeliympäristössä. Asegeneraattorilla luodaan visuaalisesti ja toiminnallisesti erilaisia aseita, joita on mahdollisimman helppo hienosäätää ilman ohjelmointia. Asegeneraattori on toteutettu Unreal Engine -pelimoottorilla, ja se toimii virtuaalilaseilla pelatessa.

Asegeneraattori on toimeksianto KollektiWe Oy:ltä, ja se on tarkoitus laittaa myyntiin kehitystyökaluksi muiden kehittäjien käyttöön. On myös mahdollista, että KollektiWe Oy jatkokehittää asegeneraattoria videopeliksi, jolloin asegeneraattorin myyminen kehitystyökaluna hylättäisiin ainakin joiltain osin. Opinnäytetyön tarkoituksena on selvittää, millaisiin osiin ja rakenteisiin virtuaalinen ase pitää pilkkoa, jotta se voidaan generoida satunnaisesti uudelleen vastaamaan oikean aseen toimintaa.

2 Vaatimukset

Asegeneraattorin vaatimuksena on tuottaa aseita, jotka koostuvat neljästä eri osasta: piipusta, rungosta, lippaasta ja olkatuesta. Jokaisella osalla tulee olla kokonaiseen aseeseen vaikuttavia muuttujia, jotka muuttavat aseen käyttäytymistä halutulla tavalla. Muuttujia voivat olla esimerkiksi aseen luotien tekemä vahinko, ampumisnopeus ja rekyyli. Osat voivat sisältää myös erikoisempia efektejä aseille, kuten räjähtävät, hakeutuvat tai jakautuvat ammukset. Näiden efektien tulee toimia muiden efektien kanssa yhteen, ja pelaaja voi määrittää niiden tapahtumajärjestyksen. Aseiden osien ominaisuuksia tulee olla helposti muokattavissa koodin ulkopuolelta.

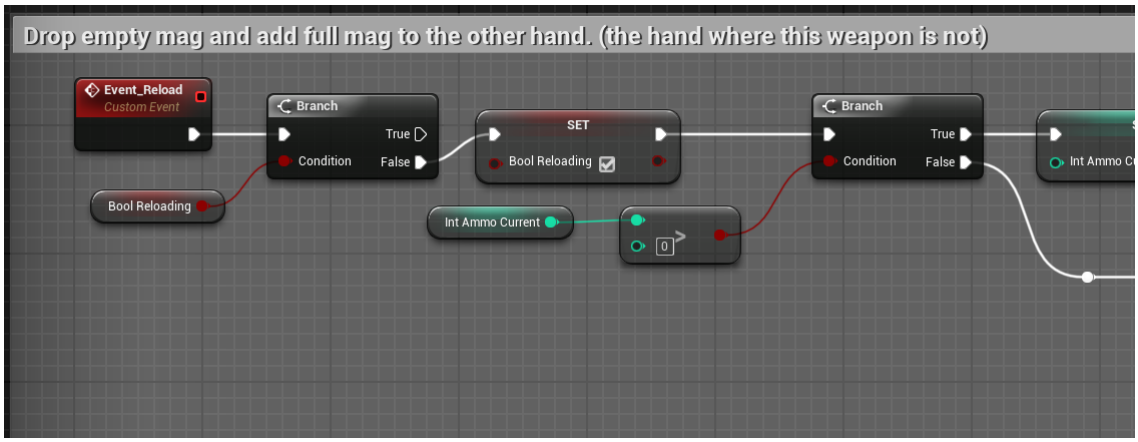
Visuaalisesti aseiden osat koostuvat useasta erilaisesta pienemmästä meshistä. Meshit ovat 3D-malleja, jotka koostuvat monikulmioista kolmiulotteisessa x, y ja

z koordinaatistossa (Rouse 2016). Unreal Enginessä x-akseli on eteen-taakse-akseli, y-akseli on oikea-vasen-akseli ja z-akseli on ylös-alas-akseli. Meshejä yhdistelemällä saadaan useita erilaisia variaatioita, ja aseen osan visuaalisuuteen voi vaikuttaa osan muuttujat tai muut efektit. Meshien tai muiden visuaalisten elementtien tekeminen ei kuulu minun toimeksiantooni, vaan tehtäväni oli asegeneraattorin logiikan ohjelmoiminen.

Toisin sanoen asegeneraattorin tulee generoida toiminnallisuudeltaan erilaisia aseita, joiden ulkonäkö antaa jonkinlaisen yleiskäsityksen aseen toiminnasta. Asegeneraattorin tulee myös toimia virtuaalitodellisuusympäristössä, ja sen tulee tukea yleisimpiä virtuaalilaseja kuten HTC Viveä.

3 Työkalut

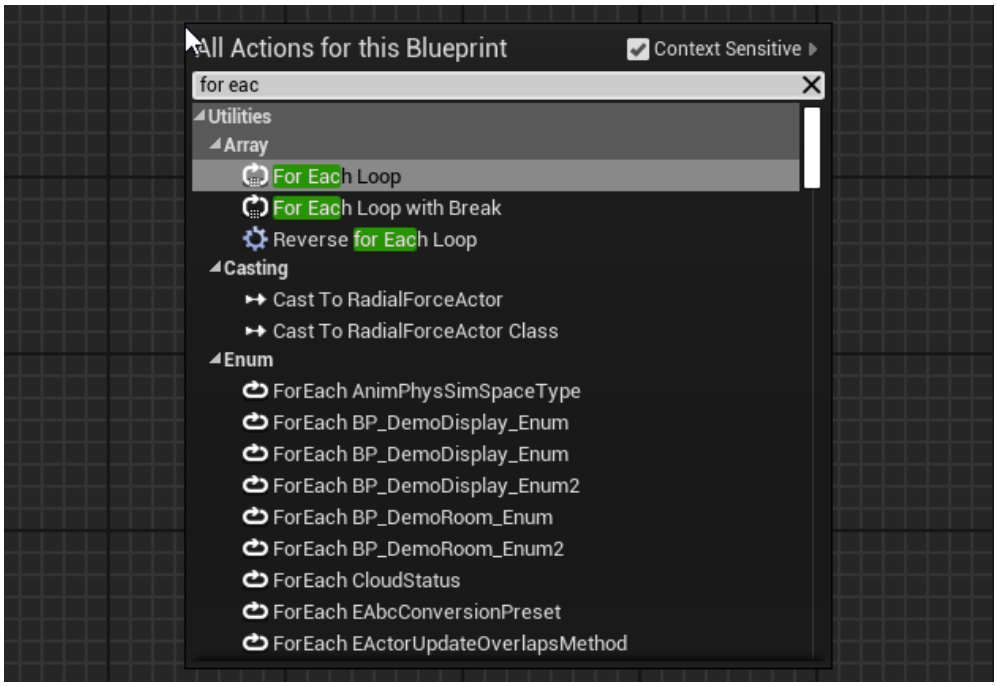
Unreal Engine 4.24 on valittu kehitystyökaluksi, koska projekti aloitettiin alun perin Unreal Enginellä, ja Kollektiwe Oy oli käyttänyt kyseistä ohjelmistoa myös muissa sen projekteissa. On hyödyllistä, että yritys käyttää useampaan eri projektiin samoja työkaluja, koska näin työkalun käytössä kehittyneet taidot siirtyvät suoraan seuraavaan projektiin. Unreal Engine on Epic Gamesin kehittämä ja julkaisema pelimoottori, joka hyödyntää Unreal Enginen omaa visuaalista skriptausta eli blueprinttejä (kuva 1) sekä C++:aa (Sutocren 2016). C++ on yleiskäyttöinen olio-ohjelmointikieli, joka pohjautuu C-kielestä (Stroustrup 2020). Unreal Enginen versio 4.24 oli asegeneraattorin tekohetkellä uusin versio, ja siksi se on valittu käyttöön.



Kuva 1. Unreal Enginen visuaalinen blueprint skriptaus on ohjelmointia, mutta koodi on esitetty visuaalisesti kuvassa näkyvien solmujen avulla.

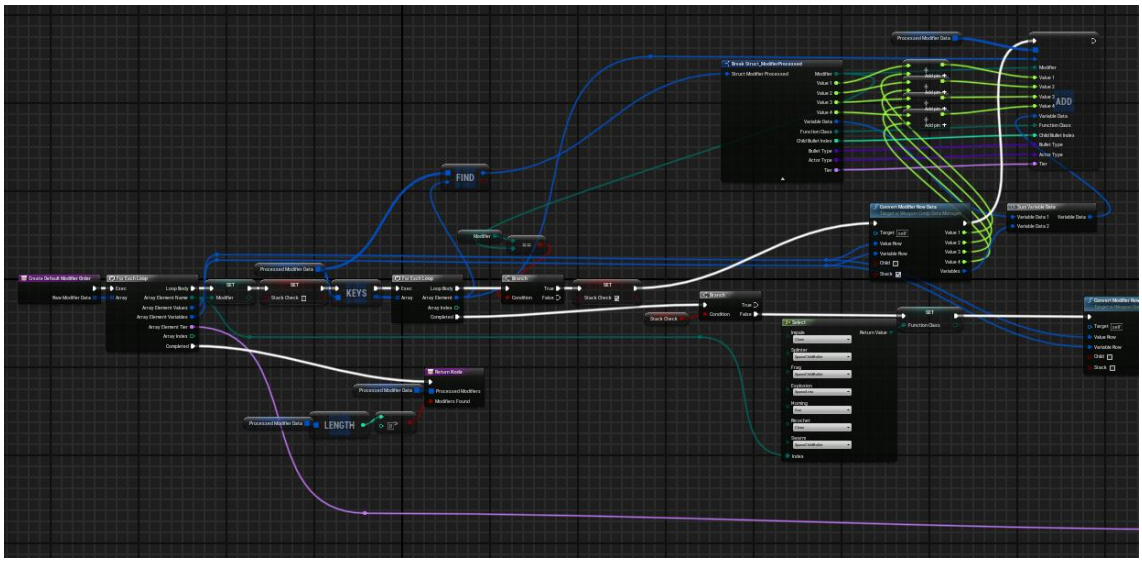
Suurin osa toteutuksesta tehdään Unreal Enginen blueprinteillä, mutta jotkin komponentit täytyy toteuttaa Unreal Enginen C++ API:n avulla, koska niitä ei voi blueprinttien avulla tehdä. Esimerkiksi tiedostorakenteeseen pääsee ainoastaan kunnolla käsiksi C++ API:n avulla. Unreal Enginen C++ API:n FindFiles-komennolla saadaan tieto kansion sisällä olevista tiedostoista, ja LoadObject-komennolla voi tiedoston ladata tiedostopolusta. Tämä on kätevää esimerkiksi meshien lataamiseen tiedostosta käsin, ja näin ne ovat automaattisesti mukana aseen visuaalisessa generoinnissa.

Blueprinttejä käytetään enemmän kuin C++:aa, sillä blueprinttien avulla työskentely on huomattavasti nopeampaa. Blueprintit tuovat työskentelyyn nopeutta, koska käyttäjän ei tarvitse kiinnittää huomiota syntaksiin, joten kirjoitusvirheistä ei synny turhia virheitä. Omaan kokemukseeni perustuen blueprintit ilmaisevat myös selkeämmin, kuin C++, jos jokin asia on pielessä. Esimerkiksi blueprinteillä työskennellessä Unreal Engine vie käyttäjän käyttöliittymän avulla suoraan virheen aiheuttamaan sijaintiin. C++:lla myös koodin kirjoittaminen on aikaa vievää puuhaa. Blueprinteillä logiikan luominen on yleensä paljon nopeampaa, koska käyttäjä voi vain etsiä haluamansa funktion valikosta (kuva 2), sekä muuttujia voi vetää ja pudottaa koodiin käyttöliittymästä.



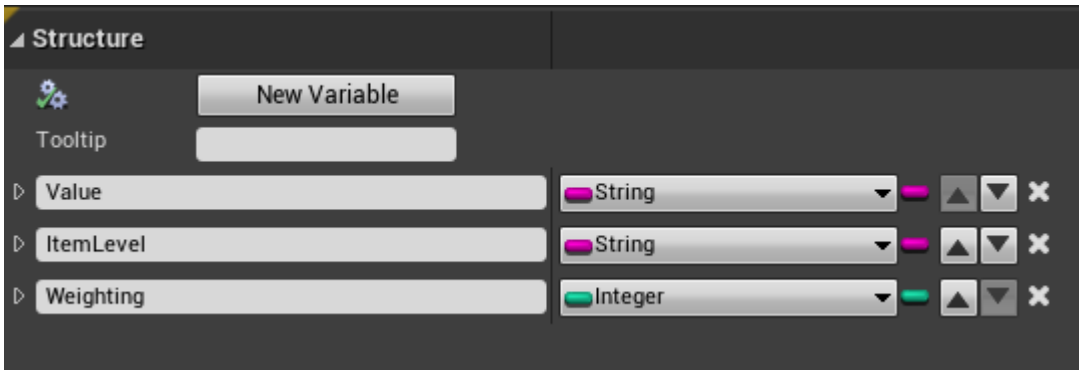
Kuva 2. Unreal Enginestä voi hakea haluttua toiminnallisuutta.

Toisaalta blueprinteillä esimerkiksi laajemmat *For Loop* ja *For Each Loop* toistosilmukkatoteutukset olivat mielestäni sotkuisempia, sekä niistä oli työlästä tehdä selkeitä (kuva 3). For Each Loop -toistosilmukan avulla voidaan käydä läpi esimerkiksi listan kaikki listatut elementit, ja jokaisen elementin kohdalla päästään käsiksi elementin tietoihin automaattisesti. Blueprinteillä listan elementti tulee vetää toistosilmukan blueprint-elementistä haluttuun toiminnallisuuteen, ellei listan elementistä erikseen tehdä omaa muuttujaansa. Muuttujan luominen vie blueprinttien ruudulta ylimääräistä tilaa, ja se aiheuttaa ylimääräisiä askeleita ohjelmointiin etenkin silloin, jos For Each Loop -toistosilmukassa käsiteltävä elementti sisältää useita erilaisia tietotyyppejä.



Kuva 3. Visuaalinen skriptaus voi mennä tietyissä olosuhteissa nopeasti sotkuiseksi.

CSV-tiedostoja käytetään kehitysvaiheessa aseen muuttujien kontrolloimiseen koodin ulkopuolelta. CSV-tiedostot ovat taulukkotiedostoja, jossa arvot on eritelty pilkuilla tai jollain muulla erottimella, ja siitä CSV on saanut nimensä: *comma-seperated values* (Barnes 2019). CSV-tiedostot valittiin mukaan siksi, koska Unreal Engine tukee CSV-tiedostojen tuomista projektiin taulukoina. Unreal Enginessä voi luoda omia rakennemuuttujia (kuva 4), joiden avulla taulukon sarakkeet määritellään (kuva 5), ja rivin nimen tietämällä kehittäjä saa taulukosta luettua kyseisen rivin arvot.



Kuva 4. Unreal Enginen rakennemuuttuja.

	Row Name	Value	ItemLevel	Weighting
1	Tier1	1.01\1.1	91\100	100
2	Tier2	0.91\1.0	81\90	100
3	Tier3	0.81\0.9	71\80	100
4	Tier4	0.71\0.8	61\70	100
5	Tier5	0.61\0.7	51\60	100
6	Tier6	0.51\0.6	41\50	100
7	Tier7	0.41\0.5	31\40	100
8	Tier8	0.31\0.4	21\30	100
9	Tier9	0.21\0.3	11\20	100
10	Tier10	0.1\0.2	0\10	100

Kuva 5. Kuva vahinkokertoimen arvon luokittelevasta CSV-taulukosta.

Vaihtoehtona CSV-tiedostoille olisi ollut myös JSON-tiedostot, mutta CSV-tiedostot edustavat enemmän tavalliselle ihmiselle luettavaa taulukkomaista rakennetta. Taulukkorakenne sopii asegeneraattorille paremmin, sillä sitä on helppo työstää Excelillä, ja sen rivejä on helpompi vertailla keskenään kuin JSON-tiedostoissa. JSON-tiedostot ovat rakenteeltaan kovin koodimaisia, ja vaatimuksena on arvojen muuttaminen helposti koodin ulkopuolelta.

4 Yleiskatsaus

Tässä luvussa käydään lävitse asegeneraattorin toteutusta yleistasolla. Aseen osan lopullisiin arvoihin vaikuttavat arkkityyppi, bonusarvot, erikoisefektit ja rungon tapauksessa myös tulitusmekanismi. Kaikille näille on jokaista aseen osaa varten useampia CSV-taulukoita, joista asegeneraattori lukee tilannetta vastaavan arvon, ja antaa sen aseen osalle. CSV-taulukoiden arvoja voi muokata kansiorakenteesta käsin, joten asegeneraattoria voi hienosäätää helposti koodin ulkopuolelta.

4.1 Arkkityypit

Aseiden osat luodaan arkkityypin mukaan vastaamaan kyseisen arkkityypin toiminnallisuutta. Arkkityyppien avulla aseita ei tarvitse generoida täysin satunnaisesti, vaan arkkityypit antavat kätevän tavan tasapainottaa aseen arvoja CSV-taulukoista käsin. Ilman arkkityyppejä aseiden generoiminen olisi erittäin ennalta-arvaamatonta, ja aseista voisi tulla usein toiminnallisuudeltaan epämieluisia tai epäloogisia, koska kaikki aseeseen liittyvät arvot pitäisi arpoa satunnaisesti. Seuraavat arkkityypit ovat asegeneraattorissa tällä hetkellä määritelty mukaan: assault, sniper, SMG ja launcher.

Assault-arkkityyppi vastaa toiminnallisuudeltaan tavallisen konekiväärin ominaisuuksia ja pyrkii olemaan jokaiselta arvoltaan mahdollisimman keskiverto. Assault-arkkityyppi toimii keskivertoisuutensa takia suunnannäyttäjänä muille arkkityypeille. Assault-arkkityyppi on keskipitkälle kantamalle tarkka ja ampuu noin 300 panosta minuutissa, ja sen ammusten lähtönopeus on noin 300 metriä sekunnissa. Assault-arkkityypin lippaaseen mahtuu noin 20 ammusta, ja se on rekyyliltään siedettävä mutta voimakas. Jos vastaavasti todellisen maailman assault-arkkityyppinä pidettäisiin M16-kivääriä, voidaan havaita, miten M16-kivääri ampuu yli puolet nopeammin eli 800 panosta minuutissa. M-16 kiväärin luotien lähtönopeus on myös yli kaksinkertainen asegeneraattorin assault-arkkityyppiin verrattuna eli noin 800 metriä sekunnissa. (Smith 2020).

Arkkityyppiarvot ovat asegeneraattorissa siis huomattavasti heikompia kuin oikean maailman vastaavassa aseessa olisi. Nämä erot johtuvat siitä, ettei asegeneraattorin luomat aseet saisi liian suuria arvoja pelimoottorin kannalta. Jos ase ampuu liian monta panosta lyhyen ajan sisällä, ja aseessa on joitakin erikoisefektejä tai niiden yhdistelmiä, voi suorituskyky Unreal Enginen puolella kärsiä. Erikoisefektejä käsitellään tarkemmin luvussa kuusi. Assault-arkkityypin ja M16-kiväärin arvojen eroihin vaikuttaa myös se, että aseiden osat saavat myös muita bonuksia arkkityyppiarvojen päälle, joten aseen arvot voivat nousta aiemmin mainittuja arvoja suuremmiksi. Näitä bonuksia käydään läpi myöhemmin tässä luvussa.

Sniper-arkkityyppi ampuu huomattavasti assault-arkkityyppiä vähemmän panoksia minuutin aikana, mutta luotien lähtönopeus on sniper-arkkityypissä huomattavasti suurempi. Sniper-arkkityyppi on myös assault-arkkityyppiä tarkempi ja omaa myös suuremman rekyylin. Sniper-arkkityyppi tekee myös assault-arkkityyppiä enemmän vahinkoa, mutta vahingon tekemistä käsitellään tarkemmin luvussa viisi. Sniper-arkkityypillä tavoitellaankin todellisen maailman kiikarikivääreitä vastaavia aseita, jotka ampuvat hitaasti, mutta niissä on sarjatuliaseita enemmän tehoa.

SMG-arkkityyppi on puolestaan assault-arkkityyppiä epätarkempi, mutta ampuu enemmän panoksia minuutin aikana. SMG on lyhenne sanoista submachine gun ja tarkoittaa automaattista asetta, joka ampuu käsiaseen panoksia (Sobieck 2015). SMG-arkkityypissä on myös vähemmän rekyyliä, sen luotien lähtönopeus on hitaampi, sekä sen lippaaseen mahtuu enemmän panoksia assault-arkkityyppiin verrattuna. SMG-arkkityyppi tavoitteleekin todellisen maailman konepistooleja ja kevyitä sarjatuliaseita ominaisuuksiltaan. SMG-arkkityypin on tarkoitus olla tehokas lyhyelle kantamalle.

Launcher-arkkityyppi eroaa huomattavasti muista arkkityypeistä, ja se tavoitteleekin kranaatinheittimen kaltaista lähestymistapaa. Launcher-arkkityyppi ampuu vähemmän luoteja minuutin aikana kuin assault-arkkityyppi. Launcher-arkkityypin luodit ovat roikkuja eli luodit ovat hitaita, ja niitä voi ampua esimerkiksi esteiden yli. Luoti voi näin pudota esteen taakse ja tehdä vahinkoa kohteeseen, mitä ampuja ei välttämättä edes näe.

4.2 Yhdistelmät

Luvusta 4.1 voi saada sellaisen käsityksen, että asegeneraattori luo aseita ja aseiden osia tiettyyn kategoriaan, eivätkä nämä kategoriat sopisi keskenään yhteen. Asegeneraattorin tarkoituksena on myös arkkityyppien sekoittaminen keskenään, eli jokainen aseen osa luodaan erikseen arkkityyppien perusteella, mutta

ase voi koostua keskenään erilasten arkkityypin osista. Generoidulla aseella voi olla esimerkiksi sniper-arkkityypin piippu, SMG-arkkityypin runko, launcher-arkkityypin lipas ja assault-arkkityypin olkatuki. Tällaiset fuusiot ovat eri arkkityyppien sekoituksia, jolloin niiden toiminnallisuus vastaa arkkityyppien yhdistelmää. Jokainen aseen osa on vastuussa tietyistä aseen arvoista, joten arkkityyppien jakautuneisuus aseiden osien kesken vaikuttaa erittäin paljon aseen toimintaan. Osien vaikutusta aseen arvoihin käydään läpi tarkemmin luvussa viisi.

4.3 Tulitusmekanismi

Kaikki aseet eivät suinkaan ammu samalla tavalla. Asegeneraattorissa on useita erilaisia tulitusmekanismeja, joita voi rajata käyttöön tietyille arkkityypeille CSV-taulukosta käsin. Aseen tulitusmekanismi määritellään sen rungossa, koska rungossa on eniten tulitusmekanismiin liittyviä visuaalisia osia, joten aseen visuaalista generointia voidaan mukauttaa tarvittaessa tulitusmekanismiin sopivaksi. Asegeneraattorissa on kuusi erilaista tulitusmekanismia: single, semi, auto, burst, windup ja charge.

Single tarkoittaa yksittäisen panoksen ampumista kerrallaan, eli jokaisen ammutun panoksen välissä käyttäjän täytyy manuaalisesti syöttää uusi panos pesään. Tulitusmekanismina single on siis erittäin hidas, ja vaatii käyttäjältä paljon ylimääräistä tekemistä muihin tulitusmekanismeihin verrattuna.

Semi on puoliautomaattinen tulitusmekanismi, eli käyttäjän pitää irrottaa ote liipaisimesta, että uuden panoksen voi ampua edellisen jälkeen. Puoliautomaattinen on jo huomattavasti nopeampi kuin yksittäisiä luoteja ampuva single. Puoliautomaattisella tulitusmekanismilla ei välttämättä voi ampua käyttäjän liipaisinsormen nopeuden mukaisesti, koska aseen tulitusnopeus vaikuttaa myös siihen, kuinka nopeasti uuden panoksen voi puoliautomaattisella aseella ampua.

Auto on täysin automaattinen tulitusmekanismi. Ampujan tarvitsee painaa vain liipaisin pohjaan, jolloin ase sylkee ulos kaikki lippaassa olevat panokset aseen

tulitusnopeuden mukaisesti. Automaattista asetta voi yrittää käyttää myös puoliautomaattisesti, jos käyttäjä päästää liipaisimen heti ammuttuaan irti. Automaattiase voi silti ampua vahingossa useamman panoksen, jos ampuja ei ole tarpeeksi tarkka liikkeissään.

Burst on tulitusmekanismi, jossa yksi liipaisimen painallus aiheuttaa kahden tai useamman panoksen ryöpyn. Ampuja ei voi kuitenkaan valita, kuinka monta panosta per ryöppy ase laukaisee yhdellä liipaisimen painalluksella, eikä ampuja voi keskeyttää käynnistettyä ryöppyä. Ryöpyn tulitusnopeus on hieman suurempi kuin vastaavassa automaattiaseessa, mutta ryöppyjen välissä oleva tauko on pidempi kuin vastaavan puoliautomaattisen aseen kanssa.

Windup on auto-tulitusmekanismin kanssa hyvin samanlainen. Erona automaattiin on windupin tulitusnopeuden kasvu ajan myötä. Windup-tulitusmekanismi aloittaa panosten ampumisen hitaasti, ja mitä pitempään käyttäjä pitää liipaisinta painettuna, sitä nopeammin ase ampuu panoksia. Windupin tulitusnopeus on aluksi huomattavasti saman tasoista auto-tulitusmekanismia hitaampi, mutta windupin tulitusnopeus menee jonkin ajan päästä automaattimekanismin tulitusnopeuden ohi. Tulitusnopeus palautuu takaisin hitaaseen, kun ampuja päästää liipaisimesta irti tai aseen lippaasta loppuu panokset.

Charge-tulitusmekanismi on eniten mielikuvituksellisin kaikkiin muihin tulitusmekanismeihin verrattuna. Charge tulitusmekanismi vastaa eniten semi-tulitusmekanismia käyttäytymiseltään, mutta charge ei ammu panosta silloin, kun käyttäjä painaa liipaisinta, vaan silloin kun käyttäjä päästää liipaisimesta irti. Mitä pidempään käyttäjä pitää liipaisinta pohjassa, sitä enemmän vahinkoa ammuttu luoti tekee sen osuessa kohteeseen.

4.4 Osan taso ja luokka

Jokaiselle luodulle aseen osalle voidaan määrittää taso esimerkiksi pelaajan etenemisen perusteella tai jollakin mulla kehittäjän päättämällä perusteella. Taso on

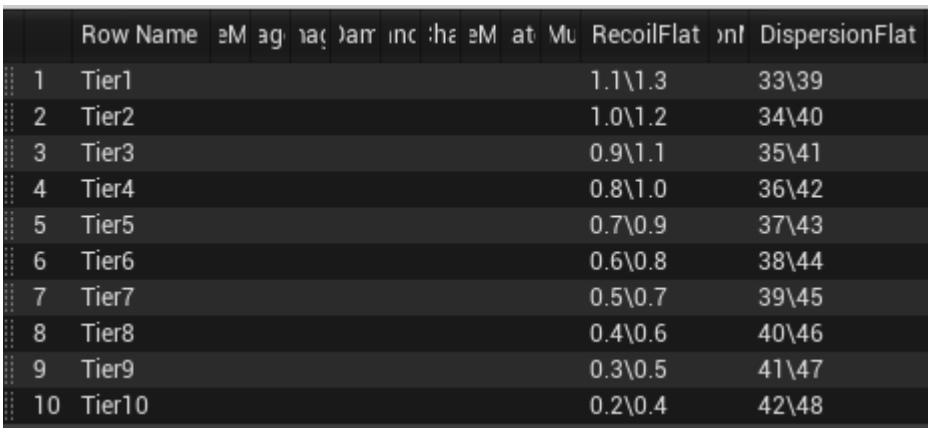
kokonaislukuarvo, ja tässä asegeneraattorissa hyödynnetään tasoja 0-100. Aseen osan taso voi vaikuttaa aseen osan arkkityypin arvoihin, bonusarvoihin ja erikoisefektien voimakkuuteen. Muuttujien arvoja käsitellään luvussa viisi, ja erikoisefektejä käsitellään luvussa kuusi. Luokkien tasovaatimukset kullekin joukolle voidaan määrittää CSV-taulukoiden avulla. Esimerkkinä kuvan 6 assault-arkkityypin luokittelutaulukosta käy ilmi, että assault-arkkityypin viides luokka (*tier5*) vaatii aseen osan tason olevan välillä 51-60. Kuvassa 6 on tapaus, jossa aseen osan taso antaa vain yhden mahdollisen assault-arkkityypin luokan lisättäväksi aseen osaan. Tällainen lähestymistapa antaa kehittäjälle mahdollisuuden lisätä aseiden osien tehoa pelin edetessä hallitusti.

	Row Name	Value	ItemLevel	Weighting
1	Tier1	Assault	91\100	100
2	Tier2	Assault	81\90	100
3	Tier3	Assault	71\80	100
4	Tier4	Assault	61\70	100
5	Tier5	Assault	51\60	100
6	Tier6	Assault	41\50	100
7	Tier7	Assault	31\40	100
8	Tier8	Assault	21\30	100
9	Tier9	Assault	11\20	100
10	Tier10	Assault	0\10	100

Kuva 6. Assault-arkkityypin luokittelutaulukko.

Asegeneraattori mahdollistaa myös lähestymistavan, jossa useampi luokka voi täyttää aseen osan tason vaatimuksen. Näissä tapauksissa otetaan mukaan CSV-taulukon painoarvo (engl. *weighting*), jonka avulla voidaan määrittää todennäköisyys luokan valitsemiselle. Mitä korkeampi painoarvo, sitä todennäköisemmin luokka valitaan. Painoarvo 200 on kaksi kertaa parempi kuin painoarvo 100.

Kun arkkityypin luokittelutaulukosta on saatu arkkityypin luokka aseen osan tason perusteella, siirrytään aseen osakohtaiseen CSV-taulukkoon lukemaan aseen osalle tulevia arvoja. Esimerkiksi kuvan 7 taulukosta luokan viisi rekyyliarvo arvotaan piipulle väliltä 0.7-0.9 ja hajonta-arvo väliltä 37-43. Luvussa viisi tarkastellaan näiden arvojen toiminnallisuutta.



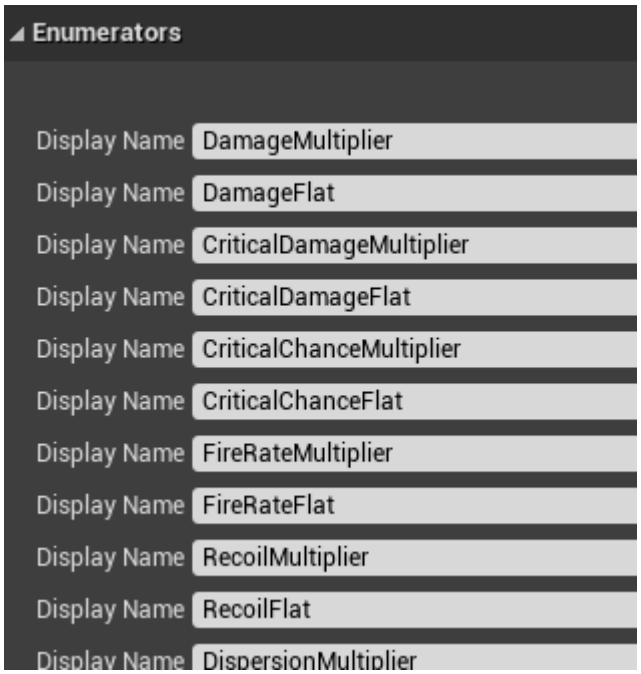
	Row Name	RecoilFlat	DispersionFlat
1	Tier1	1.1\1.3	33\39
2	Tier2	1.0\1.2	34\40
3	Tier3	0.9\1.1	35\41
4	Tier4	0.8\1.0	36\42
5	Tier5	0.7\0.9	37\43
6	Tier6	0.6\0.8	38\44
7	Tier7	0.5\0.7	39\45
8	Tier8	0.4\0.6	40\46
9	Tier9	0.3\0.5	41\47
10	Tier10	0.2\0.4	42\48

Kuva 7. Assault-arkkityypin piipun arvojen luokittelutaulukko.

5 Muuttujat

Aseen muuttujat määrittelevät aseen perustoiminnallisuuden, ja jokainen aseen osa on vastuussa tietyistä muuttujista. Aseiden osien sisältämiä muuttujia on helppo muuttaa myös CSV-taulukoista, eli tässä luvussa kerrotaan, mistä ase saa minkäkin arvonsa, sekä mitä mikäkin muuttuja tekee. Muuttujia ei kuitenkaan ole sidottu lopullisesti tässä kappaleessa mainittuihin aseiden osiin, vaan muuttujien ja aseiden osien suhteita voi muuttaa myös CSV-taulukoista käsin.

Muuttujia käsitellään aseiden ja erikoisefektien tasapainotukseen kahdella eri tavalla: yhteenlaskuna ja kertolaskuna. Yhteenlaskua varten on flat-arvot eli kokonaisarvot, ja kertolaskua varten on multiplier-arvot eli kerroinarvot (kuva 8). Lopullisen muuttujan arvo lasketaankin seuraavasti: flat-arvo kertaa multiplier-arvo. Näitä muuttujia voidaan käyttää myös ylimääräisinä bonuksina aseen osissa.



Kuva 8. Muuttujien flat- ja multiplier-jaottelu.

Damage eli vahinko aiheuttaa kohteen osumapisteiden vähenemistä, jos kohteella osumapisteitä on. Aseiden luodit tekevät siis vahinkoa vihollisiin tai muihin esineisiin, jotka voivat pelimaailmassa ottaa vahinkoa. Vahingon tasapainotus riippuu täysin vihollisten tai muiden vahinkoa ottavien esineiden osumapisteistä, joten se on täysin projekti riippuvainen ominaisuus. Tämän asegeneraattorin aseet saavat vahinkoarvonsa niiden lippaista.

Critical multiplier eli kriittisen vahingon kerroin on matemaattinen kerroinarvo kriittisten osumien tekemään vahinkoon. Luodit voivat satunnaisesti osuessaan tehdä tavallista enemmän vahinkoa, ja näitä osumia kutsutaan kriittisiksi osumiksi. Jos esimerkiksi luoti osuessaan tekee 20 vahinkoa ja kriittisen vahingon kerroin on 2.5, on kriittisen vahingon arvo tällöin 50. Tämän asegeneraattorin aseet saavat kriittisen vahingon kertoimen arvon niiden olkatuesta.

Critical chance eli kriittisen osuman mahdollisuus määrittää sen, millä todennäköisyydellä luodin osuma on kriittinen. Kriittiset osumat tekevät lisää vahinkoa, mutta voivat aiheuttaa myös muita efektejä vihollisiin, jos kehittäjä niin haluaa. Kriittisen vahingon laskeminen käytiin läpi aiemmassa kappaleessa. Tämän asegeneraattorin aseet saavat kriittisen vahingon madollisuuden arvon niiden

olkatuista. Olkatukeen oli hankala keksiä mitään aseeseen oikeasti vaikuttavia asioita, joten sinne laitettiin kriittisestä vahingosta vastaavat arvot, koska ne ovat muutenkin vähän mielikuvituksellisia arvoja, mutta ne ovat minun käsitykseni mukaan kuitenkin yleisiä varsinkin RPG-videopeleissä.

Fire rate eli aseen tulitusnopeus vastaa siitä, kuinka nopeasti aseella voi ampua. Tulinopeuden mittayksikkö on RPM eli panosta per minuutti. Tämän asegeneraattorin aseet saavat tulitusnopeus arvon niiden rungoista, koska runkoon on kytketty myös tulitusmekanismin tyyppi.

Recoil eli rekyyli aiheuttaa aseelle potkaisuefektin, kun sillä ammutaan. Mitä suurempi rekyylin arvo, sitä enemmän ase potkaisee ammuttaessa. Mitään järkevää mittayksikköä rekyylin arvolle ei voi laittaa, koska rekyylin vaikutukset riippuvat peliprojektissa tehdyistä muista ratkaisuista. Tässä asegeneraattoriprojektissa ase on kiinnitetty pelaajaan *physics constraint* -komponentilla, ja tämän komponentin arvot vaikuttavat myös rekyylin vahvuuteen. Physics constraint -komponenttia käytetään Unreal Enginessä kahden esineen yhdistämiseen fysiikoiden avulla. Esimerkiksi ovi voidaan yhdistää sen saranoiden kohdalta seinään physics constraint -komponentin avulla, ja kehittäjä voi valita kuinka paljon kitkaa oven ja seinän välille tulee. Tämän asegeneraattorin aseet saavat rekyylin arvon niiden piipuista.

Dispersion eli hajonta on käytännössä aseen tarkkuus tai epätarkkuus. Suurempi arvo ilmaisee epätarkempaa asetta. Mittayksikkönä hajonnalle on asteet. Yksi aste tarkoittaa luodeille yhden asteen hajonta aluetta, eli luodit voivat mennä piipusta maksimissaan 0.5 astetta ylös, alas, oikealle, vasemmalle tai johonkin tämän alueen rajojen sisäpuolelle. Tämän asegeneraattorin aseet saavat hajonnan arvon niiden piipuista.

Projectile speed eli luodin nopeus vaikuttaa nimensä mukaisesti luotien lähtönopeuteen ampumishetkellä. Mittayksikkönä luotien nopeudelle on m/s eli metriä per sekunti. Tämän asegeneraattorin aseet saavat luotien nopeuden arvon niiden rungoista.

Armor piercing eli panssari läpäisy vähentää vihollisten panssareiden tehokkuutta. Panssari toimii vihollisilla tavallisena vahingon vähentäjänä toimivana arvona, joka on määritelty ainoastaan vihollisessa itsessään. Panssarin läpäisylle ei siis ole varsinaisesti mitään mittayksikköä, koska se on vahingon kanssa täysin projektiriippuvainen. Esimerkkiprojektissa panssari vähentää vahinkoa kertoimen avulla. Esimerkiksi panssarin arvo 0.1 vähentää vihollisen ottamaa vahinkoa 10 %. Panssarin läpäisy arvo vähentää tätä vihollisen panssarin arvoa, joten panssarin läpäisy lisää luotien tekemää vahinkoa panssaroituihin vihollisiin. Tämän asegeneraattorin aseet saavat panssarin läpäisyn arvon niiden lippaista.

Magazine size eli lippaan koko määrittää aseen lippaan kapasiteetin varastoida ammuksia. Mittayksikkönä lippaan koolle on tietenkin siihen mahtuvien panoksien määrä, mutta koska panoksien koko vaikuttaa myös lippaan sisältämään ammusmäärään, käytetään mittayksikkönä tavallisen kokoista, ammuskoko 1.0, ammusta. Ammusten koko käydään läpi myöhemmin tässä luvussa. Tämän asegeneraattorin aseet saavat lippaan koon arvon niiden lippailta.

Reload eli lataamisaika arvo mahdollistaa lippaan lataamisen keston ajan kontrolloimisen. Lataamisajan arvon mittayksikkö on sekunteja. Tämän asegeneraattorin aseet saavat lataamisajan arvon niiden lippaista. Lataamisajan toiminnallisuutta ei ole asegeneraattorissa tällä hetkellä käytössä, koska sitä varten suunniteltuja animaatioita ei ole vielä toteutettu.

Ammo size eli ammuksen koko määrittää lippaan koon lisäksi lippaaseen mahtuvien panosten määrän. Erikoisefektejä ja muita toiminnallisuuksia suunnitellessamme huomasimme, että olisi todella kätevää, jos meillä olisi mahdollisuus lisätä tarvittavien luotien määrää joihinkin toiminnallisuuksiin. Aluksi käytössä oli systeemi, jossa erikoisefektit voivat vaatia useita panoksia lippaasta, vaikka pelaaja ampuisi vain yhden luodin. Tämä käytäntö ei ollut riittävän tarkka kontrollointiin, ja se tuntui rokottavan lipaskokoja aivan liikaa. Ammuksen koko ratkaisi tämän ongelman oikein hyvin, koska nyt erikoisefekti voi vaatia panoksen koon suurenemista, jolloin panoksia mahtuu vähemmän lippaaseen. Lippaaseen

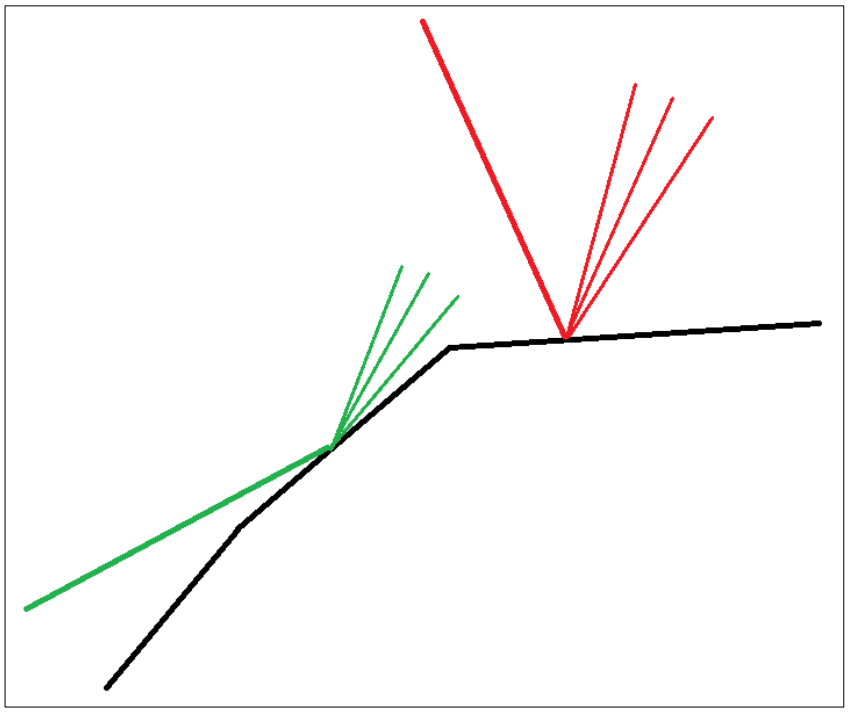
mahtuvien panosten määrä saadaan laskemalla lippaan koko kerrottuna ammusten koolla. Tämän asegeneraattorin aseet saavat ammusten koon arvon niiden rungoista.

6 Erikoisefektit

Erikoisefektien tarkoitus on muuntaa aseen toiminnallisuutta huomattavasti enemmän kuin tavallisen muuttujan arvon säätäminen. Erikoisefektien toimivat myös keskenään toistensa kanssa yhtä aikaa, ja pelaajalla on mahdollisuus valita missä järjestyksessä erikoisefektit tapahtuvat. Tässä luvussa käydään läpi näitä erikoisefektejä, sekä kerrotaan tarkemmin niiden toiminnasta.

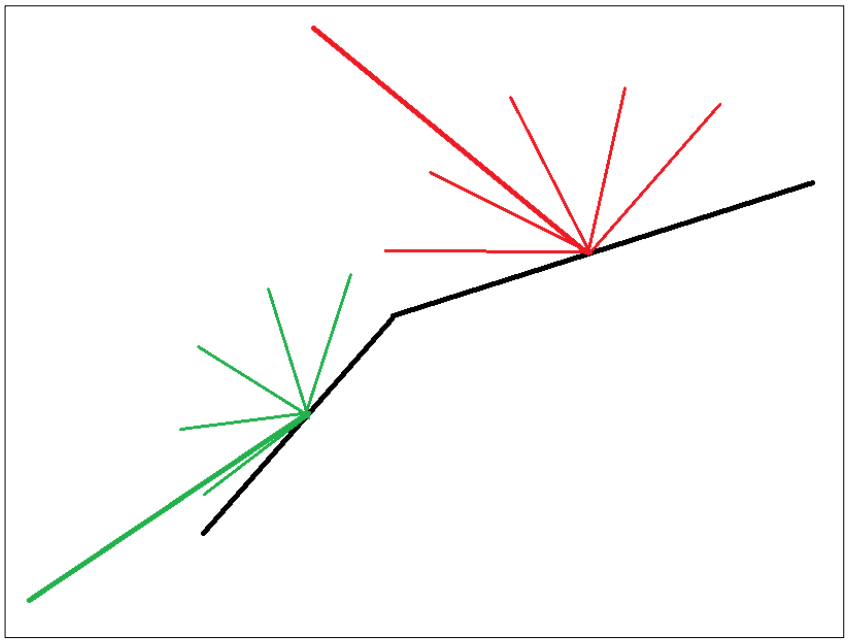
Impale-niminen erikoisefekti antaa aseen luodeille ominaisuuden läpäistä vihollisia tai esineitä. Läpäisyn matkan pituutta, luodin uudelleenohjautuvuutta osumassa, luodin vahingon pienenemistä osumassa ja osumassa menetettyä vauhtia voi kontrolloida impale-erikoisefektin arvoja sääteleväst CSV-taulukosta. Läpäisty matka mitataan luodin osumakohdasta ja poistumiskohdasta saaduista pisteistä.

Splinter-niminen erikoisefekti tekee luodeista pirstaloituvia niiden osuessa johonkin pelimaailmassa. Pirstaleet eivät ole varsinaisesti pirstaleita, vaan ne ovat uusia luoteja, jotka lähtevät matkaan pirstaloituvan luodin osumakohdasta. Nämä uudet luodit ottavat huomioon alkuperäisen luodin tulokulman osuttuun pintaan nähden (kuva 9). Pirstaloituvien luotien pirstaleiden määrää, lähtökulman satunnaisuutta ja vahinkoa voi säätää koodin ulkopuolelta splinter-erikoisefektin arvoja säätelevästä CSV-taulukosta.



Kuva 9. Pirstaloituvan luodin tulokulman vaikutus pirstaleisiin. Kaksi esimerkkiä.

Frag-niminen erikoisefekti tekee luodeista sirpalekranaatteja. Ero splinter-erikoisefektin kanssa on sirpaleiden käyttäytyminen. Frag-erikoisefekti toimii sirpalekranaatin tavoin, eli luodin tulokulmalla ei sirpaleiden lähtösuuntaan ole merkitystä. Sirpalekranaattiluodin osumakohdasta räjähtää ulospäin eri suuntiin uusia luoteja satunnaisesti (kuva 10). Sirpalekranaattiluotien määrää, vahinkoa ja lähtönopeutta voi säätää koodin ulkopuolelta frag-erikoisefektin arvoja säätelevästä CSV-taulukosta.



Kuva 10. Sirpalekranaattiluotien tulokulman vaikutus sirpaleisiin. Kaksi esimerkkiä.

Explosion-niminen erikoisefekti tekee luodeista räjähtäviä. Räjähtävät luodit luovat osuessaan räjähdysalueen, jolloin kaikki vahingolle alttiit viholliset tai esineet ottavat räjähdysalueella vahinkoa. Räjähtävien luotien vahinkoa ja räjähdyksen sädettä voi muuttaa koodin ulkopuolelta explosion-erikoisefektin arvoja säätelevästä CSV-taulukosta.

Homing-niminen erikoisefekti tekee luodeista kohteeseen hakeutuvia. Pelaaja voi maalittaa kohteita ennen ampumista osoittamalla asettaan vihollisia kohti. Vihollisia voi olla maalitettuna useita yhtä aikaa, ja luodit hakeutuvat satunnaista maalitettua vihollista kohti. Hakeutuvien luotien maalitettujen vihollisten määrää ja hakeutumisen voimakkuutta voi säätää koodin ulkopuolelta homing-erikoisefektin arvoja säätelevästä CSV-taulukosta.

Ricochet-niminen erikoisefekti tekee luodeista sen osuessaan kimpoavia. Unreal Enginessä on oma sisäänrakennettu kimmokesysteemi, joka löytyy projectile movement -komponentista nimellä Bounce. Aluksi projektissa käytettiin bounce-toiminnallisuuta, mutta sen rajoitteiden takia asegeneraattoriin luotiin oma kimmokekomponenttinsa. Kimmokeluodit hyödyntävät splinter-erikoisefektin lailla

luodin osuman tulokulmaa, jonka perusteella kimpoava luoti vaihtaa suuntaansa. Kimmoke ei luo uutta luotia, vaan alkuperäinen luoti vain muuttaa suuntaansa osumahetkellä. Kimmokkeesta voi muuttaa kimpoamisen määrää, satunnaista kulman muutosta ja osumassa menetettyä vauhtia koodin ulkopuolelta ricochet-erikoisefektin arvoja säätelevästä CSV-taulukosta.

7 Visuaalinen generointi

Aseen satunnaisen toiminnallisuuden lisäksi aseen ulkonäkö on yhteydessä sen toiminnallisuuteen. Esimerkiksi sniper-arkkityypin osat pyrkivät näyttämään tehokkaammilta ja suuremmilta kuin assault-arkkityypin osat, ja SMG-arkkityypin osat ovat pienempiä ja kevyemmän näköisiä kuin assault-arkkityypin osat.

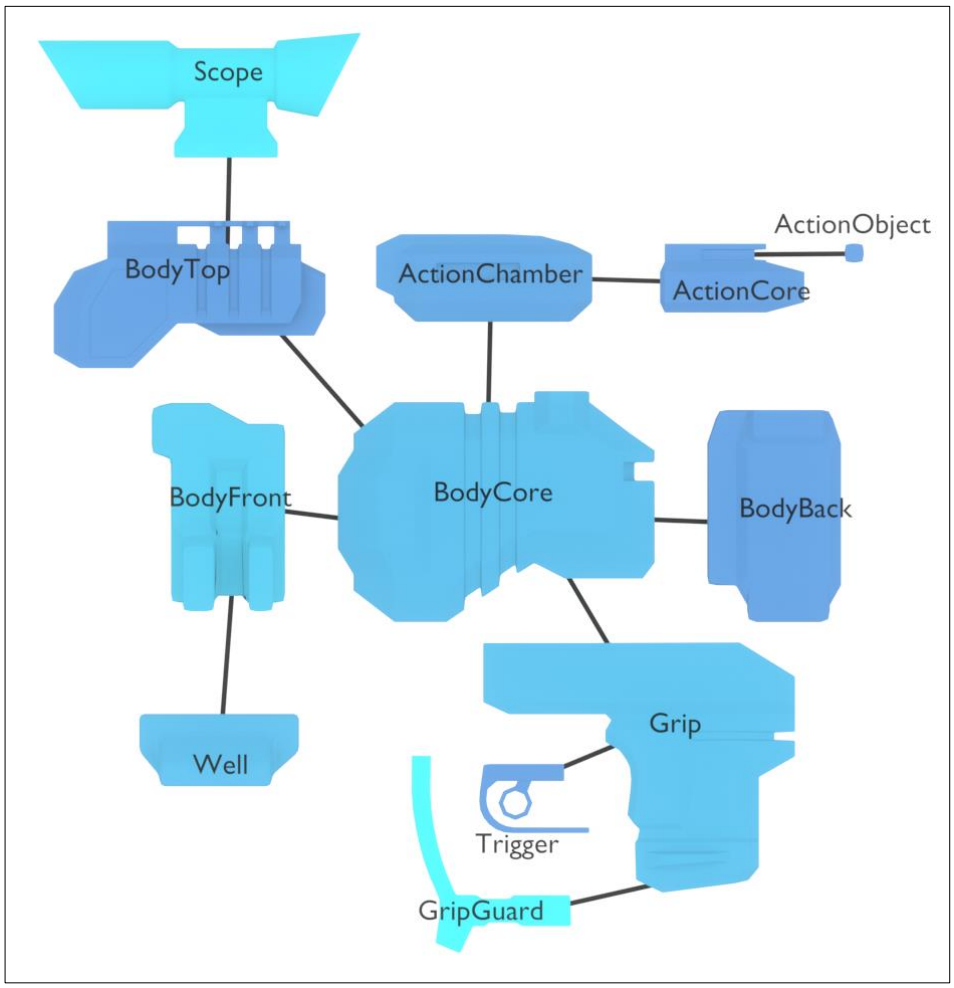
Aseen osat koostuvat pienemmistä mesheistä, jotka generoinnin aikana liitetään kiinni toisiinsa. Näitä pienempiä meshejä on eri osilla useita erilaisia, ja meshien vaihtelevuus antaa aseelle aina yksilöllisen ulkonäön.

7.1 Meshit

Jokaiselle aseen osalle on omat meshinsä, joita yhdistelemällä saadaan satunnaisesti luotu yksilöllinen aseen osa. Aseiden osien luominen aloitetaan aina yhdestä ja saman nimisestä meshistä, josta voi toki olla eri näköisiä ja kokoisia variaatioita. Aseen osan visuaalinen generointi alkaa jokaisen osan ydinpalasta, josta käytetään termiä *core.* Visuaalinen generointi lähtee ydinpalasta ulospäin valitsemaan kriteereihin sopivia muita paloja, muodostaen näin yhtenäisen kokonaisuuden aseen osalle. Aseen eri osien meshit on eroteltu omiin kansioihinsa, koska se helpottaa niiden tutkimista ja käsittelyä koodissa. Kunkin yksittäisen meshin variaatiot ovat omassa kansiossaan, ja tämä kansio sijaitsee sen aseen osan kansiossa, johon kyseinen meshi generaattorissa kuuluu. Esimerkiksi kaikki

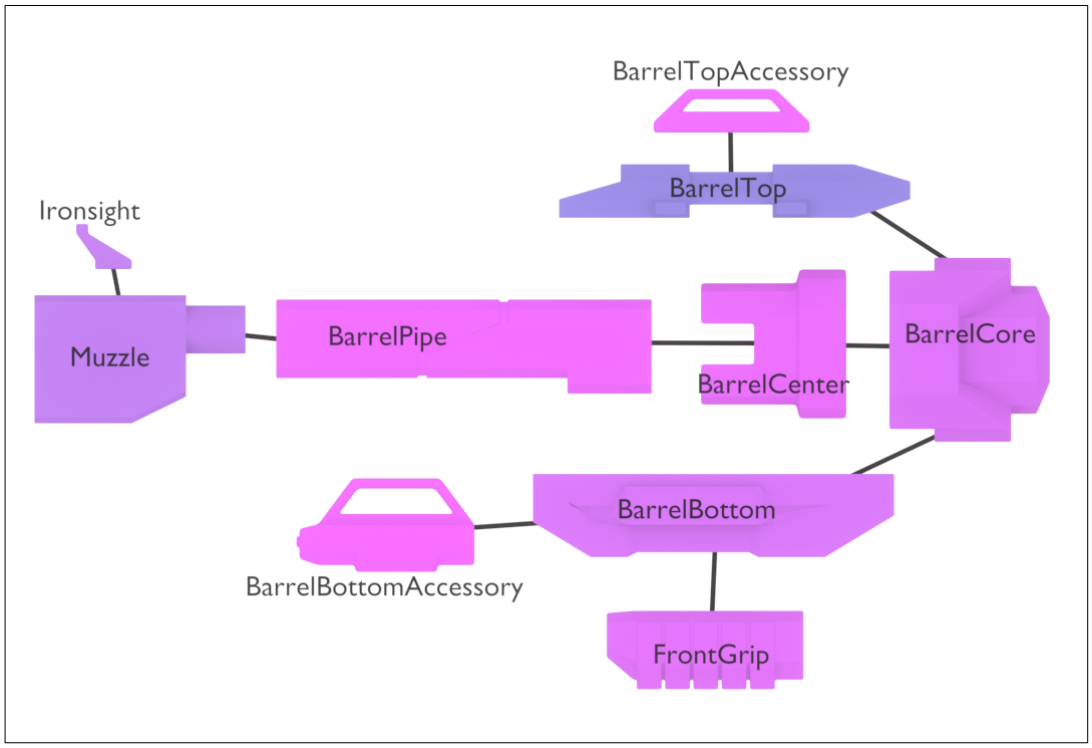
erilaiset BodyTop-meshit ovat omassa kansiossaan, joka on aseen rungon kansiossa.

Rungon generointi (kuva 11) alkaa *BodyCore*-nimisestä meshistä, johon tulee kiinni seuraavia meshejä: rungon etuosa eli *BodyFront*, rungon takaosa eli *BodyBack*, rungon yläosa eli *BodyTop*, syöttömekanismin pesä eli *ActionChamber* ja kahva eli *Grip*. BodyFronttiin tulee kiinni lippaan kaivo eli *Well,* BodyBackiin ei tule enää lisää meshejä, BodyTopiin tulee kiinni tähtäin eli *Scope*, Gripiin tulee kiinni liipaisin eli *Trigger* ja liipaisinkaari eli *GripGuard*. ActionChamberiin tulee kiinni syöttömekanismi eli *ActionCore* ja ActionCoreen tulee kiinni syöttömekanismin vedettävä osa eli *ActionObject.*



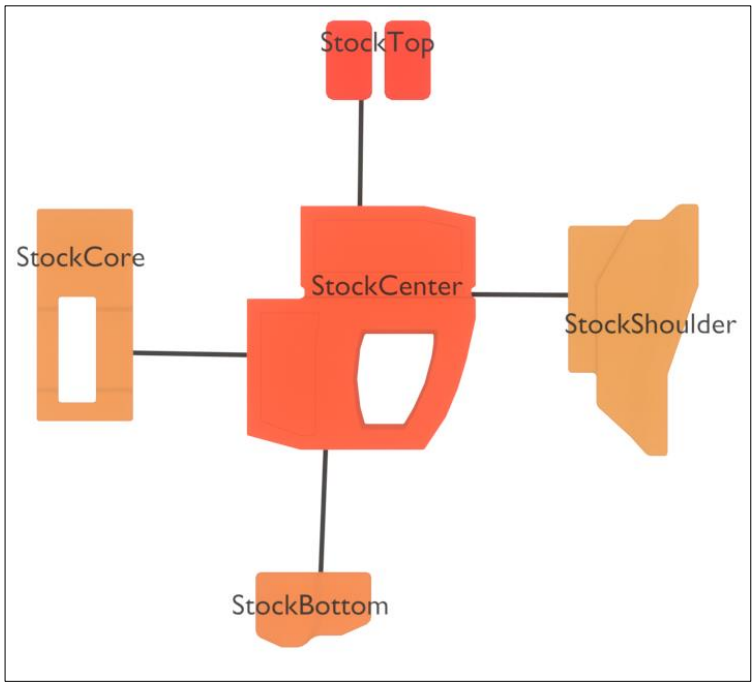
Kuva 11. Aseen rungon visuaalinen generointi.

Piipun generointi (kuva 12) alkaa *BarrelCore*-nimisestä meshistä, johon tulee kiinni seuraavia meshejä: piipun yläosa eli *BarrelTop*, Piipun alaosa eli *BarrelBottom* ja piipun keskikohta eli *BarrelCenter*. BarrelTopiin tulee kiinni lisävaruste eli *BarrelTopAccessory*, BarrelBottomiin tulee kiinni etutunkki eli *FrontGrip* ja lisävaruste eli *BarrelBottomAccessory*, BarrelCenteriin tulee kiinni yksi tai useampi piippu eli *BarrelPipe*. BarrelPipeen tulee kiinni suujarru eli *Muzzle* ja Muzzleen tulee kiinni jyvä eli *Ironsight*.



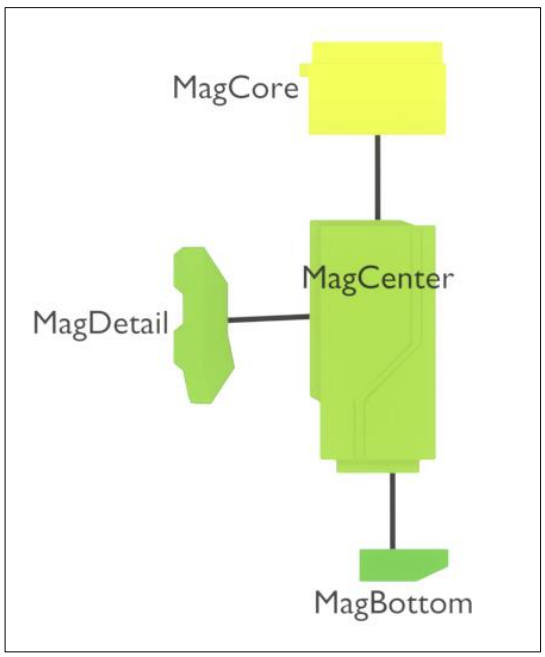
Kuva 12. Aseen piipun visuaalinen generointi.

Olkatuen generointi (kuva 13) alkaa *StockCore*-nimisestä meshistä, johon tulee kiinni olkatuen keskikohta eli *StockCenter*, johon tulee kiinni olkatuen yläosa eli *StockTop*, olkatuen alaosa eli *StockBottom* ja olkapäätuki eli *StockShoulder*.



Kuva 13. Aseen olkatuen visuaalinen generointi.

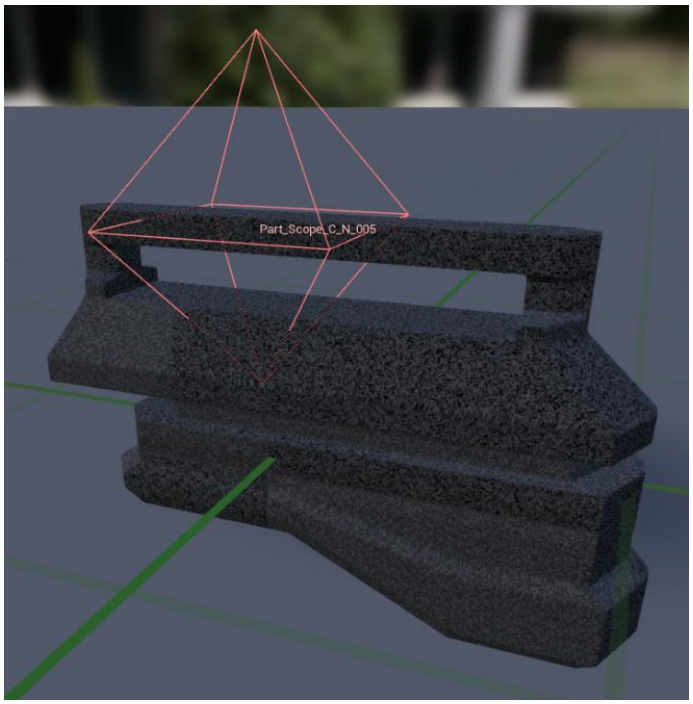
Lippaan generointi (kuva 14) alkaa *MagCore*-nimisestä meshistä, johon tulee kiinni lippaan keskikohta eli *MagCenter,* johon tulee kiinni lippaan alaosa eli *Mag-Bottom* ja lippaan yksityiskohta eli *MagDetail*.



Kuva 14. Aseen lippaan visuaalinen generointi.

7.2 Socketit

Luvussa 7.1 mainitut meshit kiinnitetään toisiinsa Unreal Enginessä sockettien avulla. Socketit ovat Unreal Enginen apuväline muiden asioiden tai esineiden liittämisessä staattisten meshien (engl. *static mesh*) ja luurankomeshien (engl. skeletal mesh) kanssa (kuva 15). Socketilla on meshiin nähden sijainti kolmiulotteisessa koordinaatistossa, jossa kolmiulotteisen koordinaatiston keskipiste on meshin pivot-pisteessä. Pivot-piste on meshin keskikohta, jonka ympäri meshin rotaatio tapahtuu. Pivot ei välttämättä ole meshin fyysisessä keskipisteessä, vaan 3D-taiteilija voi päättää pivot-pisteen sijainnin itse. Socketin rotaatio on vakiona samansuuntainen kuin meshi, jossa socket on kiinni, mutta sockettia voi kääntää kehittäjän valitsemaan suuntaan. Socketin skaala vaikuttaa socketin kokoon kolmiulotteisessa koordinaatiossa, ja yleensä socketin skaala pidetään tavallisena, koska liitettävän komponentin kokoa ei haluta liittämishetkellä yleensä muuttaa. Kehittäjä voi liittää staattiseen tai luuranko meshiin asioita ilman sockettejakin, mutta silloin kehittäjän täytyy itse siirtää liitetty komponentti sopivalle paikalle käsin. Socketti mahdollistaa kiinnitettävälle meshille ennalta määrätyn sijainnin, rotaation ja skaalan, joita kehittäjä voi käyttää hyväksi liittämisessä tai muissa meshiin liittyvissä toiminnallisuuksissa. (Unreal Engine 2020d).



Kuva 15. BodyTop-mesh jossa on aseen optiikkaa varten socket: Part_Scope_C_N_005.

Asegeneraattori hyödyntää socketteja ovelalla tavalla meshien liitosprosessissa. Jokaisen osan ydinmeshissä on seuraavalle meshille nimetty paikka. BodyCoressa on esimerkiksi socket, jonka nimessä on BodyFront, ja tämän nimen perusteella BodyFront mesh osaa mennä paikoilleen BodyCoren määräämään sijaintiin. Sockettien oveluus tuleekin siinä, että algoritmi katsoo ydinmeshistä eteenpäin generointia ainoastaan sockettien nimien perusteella. Generaattori siis ei ota kantaa, onko aseen osassa kaikki luvussa 7.1 mainitut palaset kiinni, vaan kiinnittää meshien socketteihin paloja sockettien nimen perusteella. Tämä mahdollistaa erittäin laajan visuaalisen generoinnin muokattavuuden ilman mitään koodimuutoksia.

7.3 Sockettien nimeämislogiikka

Socketin nimeen sisältyy myös paljon muutakin informaatiota, kuin pelkkä siihen liitettävän meshin nimi. Sockettien nimeäminen toimii generaattorissa

seuraavalla tavalla: Tyyppi_MeshinNimi_Arkkityypit_Erikoissäännöt_Tunniste. Alempana tarkastellaan mitä mikäkin osa tässä nimeämislogiikassa tekee.

Tyyppi on avainsana, jolla erotetaan, tuleeko sockettiin mesh vai jotain muuta toiminnallisuutta koodin puolelta. *Part*-avainsana socketissa ilmaisee, että siihen vaaditaan jokin mesh visuaaliseen generointiin. Avainsanoja on generaattorissa käytössä muitakin, kuten *function.* Function-avainsanalla varustetut socketit liittyvät aseen toiminnallisuuksiin. Niillä voi esimerkiksi määrittää paikan, josta luodit lähtevät liikkeelle tai mistä käyttäjä voi ottaa aseesta kiinni.

Meshin nimeksi tulee sockettiin haluttu meshin nimi esimerkiksi BodyFront. Meshin nimen perusteella valitaan kansio, josta listataan kaikki kansion sisältämät meshit. Jos nimen mukaista kansiota ei löydy, keskeytetään generointi virheviestin saattelemana.

Arkkityyppi kohtaan kerrotaan yhdellä kirjaimella, minkä arkkityypin meshejä saa kyseiseen sockettiin laittaa. Tällainen erittely mahdollistaa visuaalisesti hupsujen yhdistelmien eliminoimisen. Esimerkiksi suureen kranaatinheitin tyyppiseen BodyCoreen olisi hupsua laittaa paljon pienempi konepistooliin tarkoitettu BodyFront. Kuitenkin jos esimerkiksi assault-arkkityypin BodyCoreen kävisi sekä SMG-arkkityypin että launcher-arkkityypin meshejä, voi socketin nimeen laittaa molempien arkkityyppien tunnisteet, jolloin generaattori valitsee hyväksyttyjen arkkityyppimeshien joukosta satunnaisen meshin.

Erikoissäännöt voivat sisältää mitä tahansa meshien erotteluun tai erikoistoiminnallisuuteen liittyviä sääntöjä, mutta niitä ei ole tässä asegeneraattorissa vielä hyödynnetty. Erikoissääntö lisättiin nimeämislogiikkaan sen varalta, että myöhemmin mahdolliset harvinaisemmat tapaukset eivät vaatisi nimeämiskäytänteen uudelleensuunnittelua.

Tunniste on kolmenumeroinen tunnistekoodi niitä tilanteita varten, missä halutaan useampi identtinen meshi eri sijainteihin. Unreal Engine ei salli useamman

socketin olevan täysin saman niminen, joten tunnistetta on näissä muuten identtisesti nimetyissä socketeissa pakko käyttää.

Jos meshissä on useampi samanniminen socketti, lisätään jokaiseen sockettiin samalainen mesh. Jos näissä identtisiin socketteihin lisätyissä mesheissä on socketteja uusille mesheille, ovat niihin lisätyt meshit identtisiä keskenään symmetrian takia. Toisin sanoen kaikki socketit samassa meshissä, jotka pyytävät samannimistä meshiä liitettäväksi, tulevat aina saamaan identtisen generoinnin siitä pisteestä eteenpäin.

Socketin nimi voisi olla esimerkiksi: Part_BodyFront_AL_N_000. Tällainen socketin nimi tarkoittaisi, että siihen tulee kiinnittää BodyFront niminen mesh, jolla on tunnisteet A eli assault ja L eli launcher, eikä sillä ole erikoissääntöjä N eli normal, ja sillä on tunniste 000, joka ei vaikuta meshien generointiin mitenkään.

7.4 Nimeämislogiikan hyödyt ja haitat

Että sockettien nimeämislogiikka toimisi oikein, on meshit nimettävä kansiorakenteessa myös vastaavalla tavalla ilman part-etuliitettä. Tämä tietysti tarkoittaa sitä, että jos meshejä nimeää edes hieman väärin, eivät ne toimi ollenkaan tai toimivat puutteellisesti. Puutteellinen toiminta tarkoittaa väärien meshien joutumista vääriin socketteihin.

Kaikki tämä kuulostaa kovin hankalalta ja virhealttiilta työskentelyltä, ja sitä se myös jossain määrin on. Mutta tällä nimeämislogiikalla 3D-taiteilija voi luoda itse luvussa 7.1 esitellystä periaatteesta riippumattoman asegeneraattorin, noudattamalla ainoastaan ydinpalojen nimeämislogiikkaa. Ydinpalojen jälkeen 3D-taiteilija voi päättää itse sockettien nimet ja niihin kiinnittyvät meshit, ja tätä prosessia voi pilkkoa aivan kuinka pieneksi vain tarve on.

8 Koodi

Aluksi valitaan generoitava aseen osa tai generoidaan koko ase kerralla. Osien luontiprosessi alkaa aseen osan arkkityypin määrittämisellä. Arkkityyppi määritellään osan tason perusteella, ja arkkityyppiin liittyvät arvot luetaan arkkityyppien painotuksen määrittelevästä CSV-taulukosta. Osan tasoarvoa ei tässä asegeneraattorissa määritellä erikseen, koska mitään pelillisyyttä ei ole, mutta kehittäjä voi valita aseen osan tason itse sopivaksi.

Kun arkkityyppi on päätetty, valitaan CSV-taulukoiden avulla osaan vaikuttavat bonusarvot, joihin vaikuttavat aseen osan taso ja harvinaisuus. Aseiden osilla on neljä erilaista harvinaisuustasoa: tavallinen eli *common*, harvinainen eli *rare*, eeppinen eli *epic* ja legendaarinen eli *legendary*. Harvinaisuus vaikuttaa ainoastaan bonusarvojen määrään, eikä harvinaisuudelle ole mitään taulukkomäärityksiä, mutta tätä toiminnallisuutta voidaan laajentaa tarvittaessa.

Bonusarvojen jälkeen aseen osalle lisätään erikoisefekti aseen tason perusteella. Erikoisefektien tarvitsemat arvot luetaan niiden arvot määrittelemästä CSV-taulukosta, ja aseen osan tasapainotukseen käytettävät arvot luetaan erikoisefektien tasapainotuksen määrittelevästä CSV-taulukosta. Yhdelle erikoisefektille on siis yhteensä kolme erilaista taulukkoa: yksi erikoisefektin arvoille, yksi erikoisefektin tasapainotusta varten ja yksi erikoisefektin luokan määrittämiselle aseen osan tason perusteella.

Kun aseen osalle on päätetty kaikki toiminnallisuuteen vaikuttavat arvot ja muuttujat, siirrytään visuaalisen generoinnin pariin. Aloitusmeshi valitaan satunnaisesti arkkityypin rajaamasta joukosta, jonka jälkeen aloitusmeshin socketit määrittävät muut visuaaliset ominaisuudet. Visuaalisen generoinnin funktiosta saadaan myös lista siihen kuuluvista toiminnallisuuksien socketeista, joita hyödynnetään asegeneraattorin erilaisissa toiminnallisuuksissa.

Aseen runkoon tulee myös toiminnallisuus ammuksen panospesään lataamiselle. Pelaajan tulee ladata tyhjään pesään panos aseen lataamisen yhteydessä, sekä single-tulitusmekanismin kanssa ampuessa. Syöttömekanismin aktivoimiseksi pelaajan täytyy vetää syöttömekanismin kahvaa tarpeeksi pitkä matka taaksepäin, että ase laittaa uuden ammuksen pesään. Syöttömekanismin kahvaksi määritellään se meshi, joka sisältää tälle toiminnallisuudelle määritellyn socketin. Tähän socketin sijaintiin laitetaan osa, josta pelaaja voi ottaa kiinni ja vetää taaksepäin. Syöttömekanismin taaksepäin vedettävä etäisyys määritellään rungosta löytyvien kahden muun socketin etäisyyden perusteella. Nämä socketit voivat sijaita missä tahansa kohtaa runkoa, mutta tässä asegeneraattorissa ne sijaitsevat ActionCore-meshissä.

8.1 Mestariase

Kaikkia neljää osaa yhdistämällä ei maagisesti saada toimintavalmista asetta, vaan itse ase on generaattorissa näkymätön *actor*, joka tarvitsee kaikki neljä osaa toimiakseen oikein. Actorit ovat Unreal Enginen geneerinen objektiluokka, joilla on maailmassa sijainti, rotaatio ja skaala. Actoreita voi luoda maailmaan ja tuhota maailmasta pelin aikana blueprinttien tai C++ koodin avulla. Kehittäjä voi myös ohjelmoida haluttua toiminnallisuutta ja tallentaa haluttua dataa actoreihin. (Unreal Engine 2020b.) Tätä näkymätöntä actoria kutsutaan mestariaseeksi eli *WeaponMasteriksi*. Mestariase sisältää kaikki toiminnallisuudet aseen ampumisesta lataamiseen ja aseen arvojen laskemiseen.

Kun mestariaseella on tiedossa kaikki siihen liitetyt aseen osat, hakee mestariase niiltä kaikki aseen toimintaan vaikuttavat muuttujat, jonka jälkeen se laskee aseelle lopulliset arvot toiminnallisuutta varten. Kaikki mestariaseen laskelmiin vaikuttavat arvot voidaan muuttaa CSV-taulukoista käsin koodin ulkopuolelta.

Mestariase liittää piipun, lippaan ja olkatuen aseen runkoon niiden oikeille paikoilleen perustuen toiminnallisuussockettien sijaintiin ja rotaatioon. Lippaan sijaintiin laitetaan myös törmäystarkastelukomponentti lataamista varten. Asetta

voi pitää myös kahdella kädellä kiinni, ja tätä toiminnallisuutta varten mestariase liittää törmäystarkastelukomponentin aseen etupuolella olevaan kahvaan sen toiminnallisuussocketin sijainnin perusteella.

Mestariaseeseen liitetään tulitusmekanismikomponentti rungosta saadun tulitusmekanismitiedon perusteella. Jokainen tulitusmekanismi on siis oma komponenttinsa, ja lisää tulitusmekanismeja saa tehtyä kehittämällä uusia tulitusmekanismikomponentteja. Tulitusmekanismit laskevat aseen tulitusnopeuden perusteella, kuinka usein aseella on luvallista ampua. Tulitusmekanismikomponentit pitävät kirjaa, milloin viimeksi pelisession aikana pelaaja on ampunut ja onko pelaajalla liipaisin pohjassa vai ei.

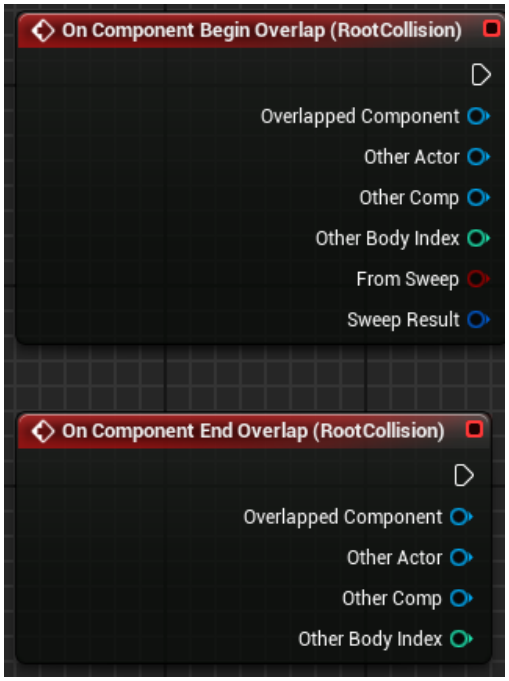
Tulitusmekanismikomponentin salliessa aseen ampumisen, tarkistaa mestariase onko aseessa tarpeeksi ammuksia ampumisen suorittamiseen. Jos panoksia on tarpeeksi, lähtee luoti piipun päässä olevasta toiminnallisuussocketista sen rotaation suuntaisesti matkaan. Luodin suuntaan vaikuttaa myös aseen hajonnan arvo ja nopeuteen vaikuttaa luodin nopeusarvo. Luodin ampumisen yhteydessä luodaan hylsy sille tarkoitetun toiminnallisuussocketin sijaintiin.

Kun aseen lataamiselle tarkoitettua nappia painetaan, siirtää mestariase aseen lippaan pelaajan vapaaseen käteen ja luo näköislippaan putoamaan aseesta maahan. Tämä näköislipas ei sisällä mitään toiminnallisuuksia, vaan se on puhtaasti visuaalinen elementti. Kun pelaaja vie kädessä olevan lippaan koskettamaan lippaan kiinnityssijaintia, loksahtaa se paikalleen, sekä mestariaseen ammukset menevät jälleen täyteen. Jos pelaajalla lataamisen yhteydessä on edes yksi panos jäljellä, on tämä panos silloin valmiina pesässä, eikä pelaajan tarvitse suorittaa ammuksen lataamista panospesään enää erikseen.

8.2 Luoti

Luotien osumatarkastelu toteutetaan aina Begin Overlap -tapahtumana, eli luodit eivät käytä ollenkaan kiinteää törmäystarkastelua. Komponenttien törmäilystä ja

päällekkäisyydestä huolehtivaa tekniikkaa kutsutaan Unreal Enginessä osumatarkasteluksi (Unreal Engine 2020a). Unreal Enginessä on tapahtumat osumatarkastelun päällekkäisyyden alkamiselle ja loppumiselle. Näitä tapahtumia kutsutaan *Begin Overlap* ja *End Overlap* -tapahtumiksi (kuva 16). Nämä tapahtumat aktivoituvat pelitilanteessa silloin, kun jokin komponentti menee toisen komponentin sisään tai tulee toisen komponentin sisältä pois. Tällainen käytäntö tarkoittaa sitä, että luodit menevät vakiona kaikista esineistä läpi, mutta osumishetkellä tieto muun muassa luodin osumakohdasta ja osutusta komponentista välitetään luodille. Luoti kuitenkin menee epäaktiiviseksi, jos se on tehnyt kaikki sille määrätyt tehtävät, eikä se epäaktiivisena vaikuta enää muihin maailmasta löytyviin objekteihin.



Kuva 16. Osumatarkastelun päällekkäisyystapahtumat.

Erikoisefektit on toteutettu pääasiassa luodin komponentteina. Jos aseeseen tulee tietty erikoisefekti, lisätään sitä vastaava komponentti luotiin, ja asetetaan se mahdollisesti aktiiviseksi riippuen erikoisefektien järjestyksestä. Jos seuraavan osuman aikana erikoisefektin pitäisi tapahtua, tulisi komponentin olla aktiivinen ennen osumaa. Erikoisuutena on ricochet- ja impale-erikoisefektit, koska niiden toiminnallisuus on ketjuttava. Jos jokin erikoisefekti on määrätty tapahtumaan ricochet- tai impale-erikoisefektin jälkeen, täytyy tällaisen erikoisefektin odottaa,

että ricochet- tai impale-erikoisefektin toiminnallisuus on päättynyt. Esimerkiksi ricochet-erikoisefekti voi tehdä luodista kahdesti kimpoavan. Jos explosion-erikoisefekti on määrätty tapahtuvan ricochet-erikoisefektin jälkeen, osuu luoti ensin ympäristöön kahdesti kimmoten, jonka jälkeen ricochet-erikoisefekti menee epäaktiiviseksi, ja explosion-erikoisefekti aktivoituu tapahtumaan luodin kolmannella osumalla.

9 Optimointi

Optimoidessa projektia logiikkaa hiotaan tai graafista näyttävyyttä vähennetään, että alusta jaksaa sulavasti suorittaa pelitilannetta. Loputtomasti ei kuitenkaan voida yksinkertaistaa koodin logiikkaa, muuten kaikki tarvittavat ominaisuudet eivät enää toimi oikein tai halutulla tavalla. Optimoidessa täytyy tasapainotella suorittimen tehokkuuden ja muistin käytön välillä.

Asegeneraattori tekee optimoinnin eteen tiettyjä valintoja ja käytänteitä koodin puolella, että generaattori veisi mahdollisimman vähän prosessointitehoa sitä hyödyntävältä alustalta pelitilanteessa. Mahdollisimman paljon asegeneraattorin tekemiä asioita pyritään siirtämään videopelin tai muun Unreal Engine projektin latausruutuun, ja näin nopeuttaen sen käyttötilanteen suorituskykyä. Latausruudussa kuitenkin muistiin ladataan paljon asioita, joita tarvitaan vasta myöhemmin, joten muistin käyttö kasvaa huomattavasti.

9.1 Ennakkoon luodut luodit

Asegeneraattorin luomat aseet voivat erikoisefektien takia luoda äärimmäisen paljon luoteja todella pienen ajan sisällö. Luodit ovat Unreal Enginen actoreita, eivätkä pelkästään maailman halki lentäviä meshejä, koska luodin tulee viedä tietty määrä informaatiota törmäyspisteisiin. Aseen ampuessa olisi suorituskyvyn kannalta kallista luoda uusi actori, koska Unreal Enginessä actorin luominen

vaatii käytettävältä alustalta actorin luomiseen liittyviin funktioihin ylimääräisiä resursseja. Vaikka uuden actorin luominen vaatisi vain vähän ylimääräistä suorituskykyä, on mahdollista, että uusia luoteja tarvitsee luoda todella paljon lyhyen ajan sisällä. Nämä luodut luodit tulisi vielä poistaa moottorista, kun ne ovat tehtävänsä täyttäneet, ja actorien poistaminen vie myös alustalta suorituskykyä. (Unreal Engine 2020c.)

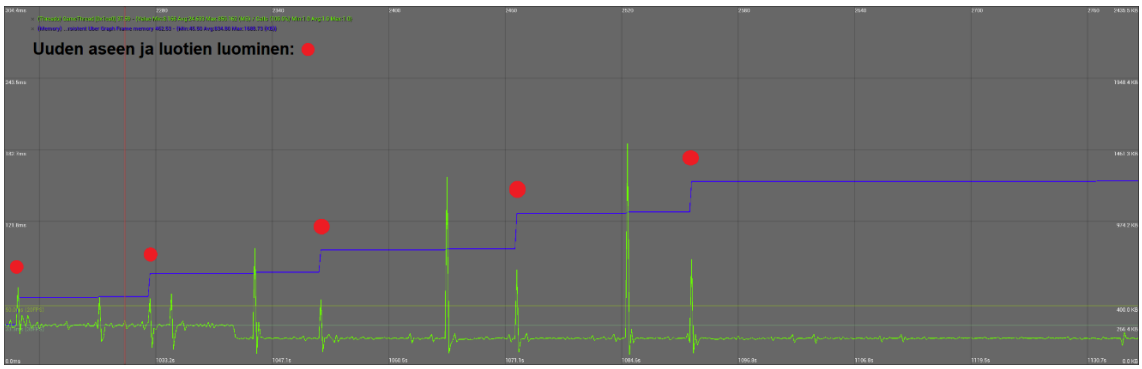
Välttääksemme ylimääräisten actorien luomisen ja tuhoamisen projektin ajon aikana, luomme tarvittavan määrän luoteja latausruudun aikana. Projektista riippuen latausruutu voi olla käynnistyksen aikana tai esimerkiksi kentän vaihdon aikana. Latausruudun aikana voimme luoda satoja tai jopa tuhansia luoteja, antaa niille tarvittavat arvot ja laittaa ne piiloon ennen kuin ne otetaan käyttöön. Kun käyttäjä ampuu asegeneraattorin luomalla aseella, aktivoituu piiloon luotu luoti, joka siirtyy aseen piipun päähän ja lähtee siitä kohti määränpäätänsä. Kun luoti on tehnyt kaikki sille osoitetut tehtävät, se määritellään epäaktiiviseksi, jolloin sitä voidaan käyttää taas uudestaan tarpeen vaatiessa.

Asegeneraattori voi kuitenkin tarvita useita erilaisia luoteja toimintansa aikana. Näitä erilaisia luoteja voivat olla esimerkiksi frag- tai splinter -erikoisefektin luomat uudet luodit, jotka eivät ole yhtä tehokkaita eivätkä ne sisällä tietenkään omaa efektiään. Tästä syystä asegeneraattorin on luotava valmiiksi useita erilaisia luoteja riippuen aseen erikoisefekteistä. Luotien luonnin yhteydessä eri efekteille määrätään niille kuuluva joukko luoteja, joita ne voivat sitten tarvittaessa käyttää. Frag- tai splinter-erikoisefektin tapahtuessa kyseisen erikoisefektin komponentti pyytää luoteja sille määrätystä joukosta siirtymään efektin tapahtumasijaintiin ja aktivoitumaan.

9.2 Muistin käyttö

Pelitilanteeseen valmiiksi luomalla tarvittavia objekteja säästytään pelitilanteessa objektin luomisprosessilta, mutta valmiiksi luodut objektit on tallennettava muistiin odottamaan niiden käyttöhetkeä. Tämän käytännön takia tietokoneen muistista

täytyy varata tilaa kaikille näille ennakkoon luoduille objekteille, vaikka niitä ei koskaan pelitilanteessa tarvittaisi. Kuvasta 17 nähdään, kuinka uuden aseen ja siihen liittyvien luotien luominen aiheuttaa piikin ruudunpäivitysnopeuteen ja lisää muistin käyttöä. Ruudunpäivityspiikki ennen muistikäyrän kasvua tapahtuu tiedostosta ladattavien meshien yhteydessä. Vaikka uuden aseen luomisen yhteydessä poistetaan entinen ase ja sen luodit pelistä, ei muistin käyttö pienene kuitenkaan vielä tässä vaiheessa, koska roskienkeruu ei ole vielä aktivoitunut. Unreal Enginessä on sisäänrakennettu *Garbage Collection* eli roskienkeruujärjestelmä, jonka tehtävä on pelisession aikana poistaa muistista objekteja, joita ei enää käytetä mihinkään tai jotka on erikseen merkitty poistettaviksi (Unreal Engine 2020e). Viimeisimmän aseen luonnin jälkeen, aloitetaan aseella ampuminen frag-erikoisefektin kanssa. Mutta kuten kuvasta 17 nähdään, ei viimeisimmän aseen luonnin jälkeen ruudunpäivitykseen tule piikkejä, koska uusia luoteja ei tarvitse pelitilanteessa enää luoda.



Kuva 17. Unreal Enginen Profiler-työkalu mittaamassa muistin käyttöä (sininen) ja ruudunpäivitysaikaa (vihreä). Matalampi käyrä on molemmille parempi.

10 Testaaminen

Testaus on tärkeä osa jokaista projektia, koska ilman testaamista ei tiedetä toimivatko kaikki ominaisuudet halutulla tavalla. Testaamiseen voi osallistua useita henkilöitä, ja testaamisen tuloksena löydetyt ongelmat on hyvä laittaa talteen.

Ongelmaraporttien avulla kehittäjä pystyy korjaamaan löydetyt virheet varsinkin, jos lokitiedostoja on hyödynnetty kattavasti.

Projektin testaaminen suoritettiin pääasiassa manuaalisesti käsin viesti- ja tulostuslokien sekä testauspiirtotyökalujen avustamana. Testaamiseen käytettiin HTC Vive virtuaalilaseja, että kaikki virtuaalilasien tukemat ominaisuudet saatiin testattua. Toimin itse pääasiassa testaajana, mutta myös yksi kollegani auttoi myös muutamalla ongelmaraportilla.

10.1 Viestiloki

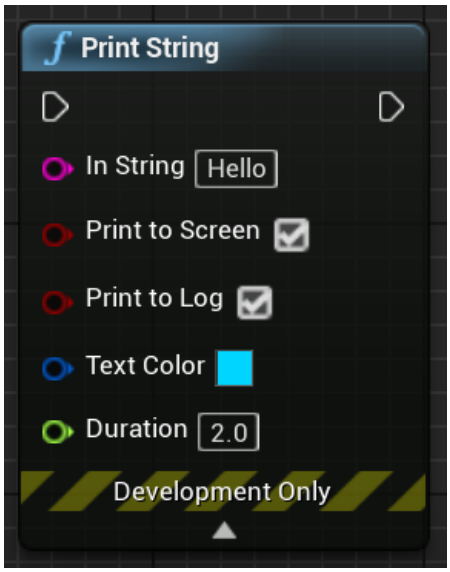
Unreal Enginen viestilokiin *Play in Editor* -luokittelun alle tulostuu kaikki sellaiset virheet, jotka Unreal Enginen sisällä menee pieleen pelimoottorin ajon ja pelisession aikana. Viestilokiin menee siis sellaiset lokitulostukset, jotka kertovat kehittäjälle, jos koodissa on Unreal Enginen funktioiden tai muuttujien kanssa ongelmia.

Yleensä nämä ongelmat liittyvät muuttujiin, joista Unreal Enginen ei löydä oikeanlaista dataa. Otetaan esimerkiksi viittauksen sisältävä muuttuja, jota ei koskaan alusteta. Viittaukset ovat muuttujia, jotka osoittavat olemassa olevaan dataan. Jos tällaiselle tyhjälle muuttujalle yritetään tehdä jotain, tulostuu viestilokiin virhe. Virheessä kerrotaan yrityksestä päästä käsiksi viittaukseen, jota muuttuja ei sisältänyt, jonka takia vaadittua tehtävää ei voitu suorittaa. Toinen selkeä skenaario viittausten kanssa on sellaisten muuttujien käsittely, joissa viittaus kyllä löytyy, mutta Unreal Enginen roskienkeruu on poistamassa tai jo poistanut viitattavaa asiaa.

10.2 Tulostusloki

Unreal Enginen tulostuslokiin tulostuu kaikki Unreal Enginen sisäiset ja käyttäjän tekemät omat lokiviestit. Lokiviestit voivat ilmaista virheistä, varoituksista tai jotain ihan tavallista informaatiota jostain käsitellystä tapahtumasta.

Blueprintteihin kehittäjä voi lisätä omia lokiviestejä *Print String*-funktion avulla (kuva 18). Print String-funktion avulla kehittäjä voi lähettää yksilöidyn lokiviestin kyseisestä koodin sijainnista. Lokiviestin voi halutessaan tulostaa pelimoottorin ajon aikana ruudulle tai tulostuslokiin tai molempiin, kuten kuvasta 18 nähdään valintaruudut Print to Screen ja Print to Log vaihtoehdoille. Jos kehittäjä päättää tulostaa tekstin myös ruudulle, voi hän valita tekstin värin sekä tekstin ajallisen keston ruudulla.



Kuva 18. Print String -komento Unreal Enginen blueprinteissä.

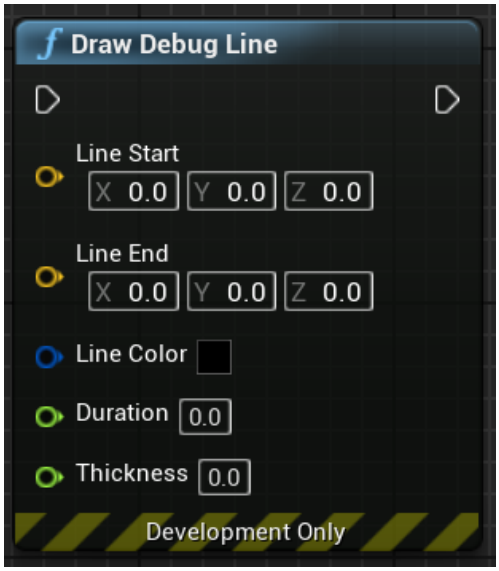
C++ koodin puolella lokiin tulostaminen on tehty UE_LOG -makron avulla. UE_LOG-makro on määritelty projektin otsikkotiedostossa, ja makro tarvitsee jokaiseen lokitapahtumaan tiedon tulostettavasta viestistä ja lokitapahtuman tason. Lokitapahtuman tason avulla hallitaan lokiviestin väriä ja suodattamista. Lokitapahtuman yleisimmät tasot ovat *Error* eli virhe, *Warning* eli varoitus, *Fatal* eli kriittinen virhe ja *Log* eli tavallinen lokitulostus (Unreal Engine 2020f). Blueprinteillä ei samanlaista lokitapahtuman tasoa voi määritellä, mutta huomasin blueprinttien

Print String -komennon ymmärtävän myös sanoja Error ja Warning. Olinkin projektissa muotoillut kaikki virhe- ja varoitustapahtumat alkamaan sanoilla Error tai Warning, ja ilokseni huomasin, että tulostuslokiin nämä virhe- tai varoitusviestit tulostuvat niiden oikeilla väreillään. Error-sanalla alkavat virheviestit ovat punaisia, ja Warning-sanalla alkavat varoitusviestit ovat keltaisia.

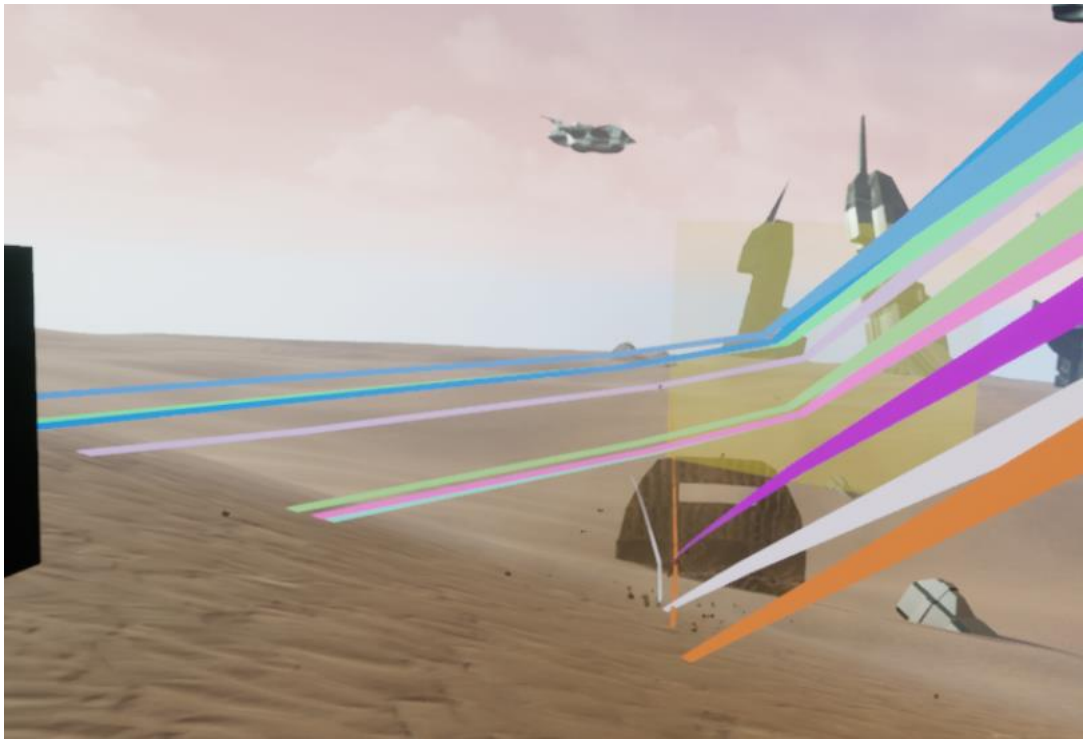
10.3 Testauspiirtotyökalut

Testauspiirtotyökalut ovat Unreal Enginen sisäisiä aputyökaluja kehitykseen, joilla voi helposti piirtää haluttuun sijaintiin kuvioita tai viivoja. Luotien sijaintien selvittämiseksi projektissa on käytetty *Draw Debug Line* -komentoa (kuva 19) jokaisen ruudunpäivityksen hetkellä, koska nopeiden luotien radan ymmärtäminen on ilman tällaista piirto-ominaisuutta todella vaikeaa.

Draw Debug Line -komento tarvitsee toimiakseen piirrettävän viivan alkupisteen sijainnin sekä lopetuspisteen sijainnin, jotka ovat kuvassa 19 *Line Start* ja *Line End*. Draw Debug Line -komennolla voi myös määrittää piirrettävän viivan värin, ajallisen keston sekä paksuuden. Projektissa jokainen luoti saa satunnaisen värin, jonka avulla on mahdollista nähdä yksilöllisesti jokaisen luodin lentorata, jolloin ne eivät sekoitu keskenään helposti (kuva 20).



Kuva 19. Draw Debug Line -komento Unreal Enginen blueprinteissä.



Kuva 20. Luotien visualisoiminen Draw Debug Line -komennon avulla.

11 Valmiin generaattorin esittely

Tässä luvussa esitellään valmiin generaattorin luomia aseita ja niiden ominaisuuksia. Tässä luvussa referoidaan paljon KollektiWe Oy:n tekemää GunGen 1.0 Preview esittelyvideota, jossa asegeneraattorin ominaisuuksia esitellään. Tästä syystä ennen luvun lukemista esittelyvideon katselu on suositeltavaa, mikäli mahdollisuus siihen on. Visuaalisesti asegeneraattorista puuttuu vielä tekstuurit, ja siksi esittelyversiossa tekstuurit ovat yksivärisiä, mutta ne on tarkoitus vaihtaa lopulliseen tuotteeseen myöhemmin. Videopeleissä tekstuurilla tarkoitetaan 3D-mallin pinnan visuaalista materiaalia, jonka tarkoitus on saada pinnalle halutut yksityiskohdat ja tarkemmat muodot, joita ei itse 3D-mallissa ole (Bogost 2008). Esittelyvideossa luotien nopeutta on pudotettu huomattavasti, ja luodit piirtävät perässään luvussa 10 selitetyn Draw Debug Line -viivan, että luotien lentorata ja aseiden erikoisefektit olisivat selkeämmin havaittavissa.

KollektiWe Oy (2020) esittelee videon alussa asegeneraattorin visuaalista puolta generoimalla useita satunnaisia aseita. Vaikka asegeneraattorin visuaalisen puolen koodi toimii odotetulla tavalla, voi joillakin aseilla olla ongelmia tähtäimien kanssa. Joidenkin aseiden piipun päässä on tähtäämistä varten tarkoitettu jyvä, jonka linjaamalla rungossa olevan tähtäinelementin kanssa, tietää käyttäjä aseen olevan tällöin suorassa. Joissain aseissa käyttäjä ei kuitenkaan voi nähdä tätä jyvää tai se on toiseen tähtäinelementtiin nähden liian alhaalla, jolloin käyttäjä ei voi linjata asetta oikein. Tämä johtuu visuaalisessa generoinnissa käytettävien meshien suunnitteluhaasteista. Mahdollisuus koodissa kehitetylle ratkaisulle kuitenkin on olemassa. Jyvää voisi esimerkiksi venyttää rungon tähtäinelementin tasolle, jolloin aseen linjaaminen onnistuisi oikein, mutta tähän versioon sitä ei tarvittu.

Pelaaja voi vaihtaa erikoisefektien tapahtumajärjestystä siihen tarkoitetusta käyttöliittymästä. Pelitilanteessa ruudunpäivitysajat voivat tästä hieman kärsiä, koska generaattorin on poistettava kaikki entiset luodit ja luotava uudet luodit uusien arvojen perusteella. Uudelleenjärjestelyn ansiosta pelaajalla on kuitenkin kontrolli erikoisefektien järjestyksestä, jolloin pelaaja voi säätää niitä oman mieltymyksensä mukaan. Jos erikoisefektien järjestys ei olisi tällä tavoin säädettävissä, olisi pelaajan todella vaikeaa löytää mieleinen ase efekteineen pelkän satunnaisuuden avulla. KollektiWe Oy (2020) julkaiseman esittelyvideon kohdassa 0:30 aseen ampuma luoti kimpoaa kahdesti, jonka jälkeen luoti hakeutuu kohti punaista kuutiota ja osuessaan hajoaa sirpalekranaatiksi. Tämän jälkeen käyttäjä vaihtaa ricochet- ja homing-erikoisefektin paikkaa keskenään, jolloin luoti lähtee heti hakeutumaan kohti punaista kuutiota.

KollektiWe Oy (2020) esittelee videon kohdassa 0:45 ampuma asetta, jossa on käytössä single-tulitusmekanismi, jolloin käyttäjän täytyy syöttää manuaalisesti ammus panospesään jokaisen ammutun panoksen jälkeen. Ammuksen manuaalinen panospesään syöttäminen luo panospesästä poistettavalle hylsylle visuaalisen hylsy-meshin, joka käyttäytyy oikean hylsyn lailla. Aseella ammutut luodit menevät myös läpi videolla näkyvästä läpikuultavasta keltaisesta kuutiosta.

Aluksi luoti räjähtää ensimmäisen osuman yhteydessä, mutta erikoisefektien vaihtamisen jälkeen ammus ensin läpäisee kohteen ja räjähtää vasta sen jälkeen. Läpäistyn matkan osoittaa kuutioon piirrettävä oranssi läpäisyviiva.

12 Unreal Engine Marketplace

Unreal Enginen Marketplacesta kehittäjät pystyvät ostamaan tai hankkimaan ilmaiseksi muiden kehittäjien ja sisällöntuottajien tekemiä valmiita Unreal Enginessä käytettävää sisältöä. Sisältö voi olla esimerkiksi 3D-malleja, toiminnallisuuksia sisältäviä blueprinttejä, ääniä tai suurempia kokonaisuuksia.

Valmis asegeneraattori on tarkoitus laittaa myyntiin Unreal Enginen Marketplace -kauppapaikkaan muille Unreal Engine kehittäjille, mutta asegeneraattorin tulevaisuus voi olla myös yrityksen sisällä kehitettävässä videopelissä tai molemmissa. Vaikka asegeneraattorin tulevaisuutta ei ole vielä tarkkaan määritelty, käydään tässä luvussa läpi mahdollisuutta julkaista asegeneraattori muille kehittäjille Unreal Enginen Marketplacessa.

12.1 Julkaisuprosessi

Sisällön julkaisemiseen Unreal Enginen Marketplacessa sisällön tuottaja tarvitsee julkaisijaprofiilin. Julkaisijaprofiiliin tarvitaan tietenkin Epic Games -tili ja erillinen rekisteröityminen osoitteessa publish.unrealengine.com. Julkaisemiseen vaaditaan myös maksu- ja verotietojen ilmoittamista. Kun kaikki julkaisua varten tarvittavat tiedot on ilmoitettu, voi sisällöntuottaja lisätä myyntiin tarkoitetun tuotteen profiiliinsa, jonka jälkeen sen voi lähettää Epic Gamesin hyväksyttäväksi.

Tuotetta ei hyväksytä myyntiin Marketplaceen, jos tuote on: laillisesti ongelmallinen, huonolaatuinen, liian helposti toistettavissa, keskeneräinen tai kykenemätön toimimaan mainostetulla tavalla. Jos tuotetta ei ensimmäisellä kerralla

Marketplaceen hyväksytä, voi kehittäjä tehdä ohjeiden mukaisia muutoksia, jolloin tuote voidaan lähettää uudestaan hyväksymisprosessiin. (Unreal Engine 2019).

12.2 Tuotteen myyminen

Tuote maksetaan Marketplacessa, ja tuotteen julkaissut taho saa verojen jälkeen 88% tuotteen myynnistä ja 12% menee Epic Gamesille. Tuotteen julkaisijalle maksetaan kuukauden myynnistä kertyneet tuotot viimeistään 45 päivää kalenterikuukauden loppumisen jälkeen, kunhan julkaisijalle kertyy vähintään sadan Yhdysvaltojen dollarin edestä maksettavaa. (Unreal Engine 2020g).

Kehittäjän täytyy myös pitää huolta, että myynnissä oleva tuote on yhteensopiva mahdollisimman monen Unreal Engine version kanssa. Jos mahdollinen ostaja haluaa käyttää Marketplacessa myynnissä olevaa tuotetta, mutta ostaja käyttää eri versiota Unreal Enginestä kuin myyjä, ei ole takeita tuotteen toimivuudelle ostajan Unreal Engine versiossa. Tällaisten versioiden yhteensopivuusongelmien takia tuote saattaa jäädä ostajalta ostamatta.

13 Unreal Enginen ongelmat

Unreal Engine on työkaluna oikein pätevä tällaisen asegeneraattorin tekemiseen. Unreal Enginen avulla työ on nopeaa ja sujuvaa sisäänrakennettujen toiminnallisuuksien ansiosta. Tämä työkalu kehittyy koko ajan, ja siihen lisätään uusia ominaisuuksia uusien versioiden yhteydessä.

Nopeasti kehittyvässä työkalussa kuitenkin on vaarana, että kaikki uudet toiminnallisuudet eivät välttämättä toimi aivan halutusti tai ne saattavat jopa hajottaa vanhoja toiminnallisuuksia. Unreal Enginessä on muutamia toiminnallisuuksia,

joiden virheellinen tai ei oletettu toiminta hidasti työntekoa. Tässä luvussa tarkastellaan kehitystyön aikana ilmenneitä ongelmia tarkemmin.

13.1 Osumatarkastelu

Impale-erikoisefekti tuotti ylivoimaisesti eniten päänvaivaa, vaikka ajatus sen takana on yksinkertainen. Suurin ongelma oli saada luodin poistumiskohdan tarkka sijainti, koska Unreal Enginessä ei tälle toiminnallisuudelle ollut mitään olemassa olevaa työkalua. Loogisesti ajateltuna osumakohdan sijainti otetaan Begin Overlap -tapahtumasta ja poistumiskohta End Overlap -tapahtumasta. Mutta kuten kuvasta 16 voidaan tehdä huomio, on End Overlap -tapahtumassa paljon vähemmän välitettyä tietoa, ja siitä puuttuu myös tieto, missä sijainnissa osumatarkastelu on lopettanut päällekkäisyyden. Begin Overlap -tapahtumassa kaikki tarvittava informaatio saadaan *Sweep Result* -rakenteen sisältä, josta saadaan muun muassa osumakohdan sijainti ja osutun pinnan normaalivektori.

Eli koska End Overlap -tapahtuma ei suoraan anna luodin poistumissijaintia, voidaan luodin sijainti ottaa sitten End Overlap -tapahtuman tapahtuessa, mutta tässä lähestymistavassa ongelma on luotien nopeus. Unreal Engine laskee luotien rataa monta kertaa ruudunpäivityksien välissä, ja blueprinteissä luodin sijainti saadaan ainoastaan jokaisen ruudunpäivityksen kohdalla, jonka takia End Overlap -tapahtumasta ei saa tarkkaa luodin sijaintia. Esimerkiksi jos kuvan 21 ensimmäisessä tapauksessa haluttaisiin tieto kolmannen ja neljännen simulaatioaskeleen välisestä luodin poistumissijainnista, antaisi End Overlap -tapahtuman hetkellä luoti omaksi sijainnikseen toisen ruudunpäivityssijainnin, eli sijainnin *2. frame*-kohdasta. Vaikka Unreal Enginestä saisi helposti tietoon luodin sijainnit ruudunpäivityksien välistä, eivät ne olisi silti tarkkoja luodin poistumissijainteja, vaan ne olisivat luodin senhetkisiä sijainteja maailmassa. Näiden syiden takia End Overlap – tapahtuma ei ole toimiva tällaisen tiedon hankintaan.

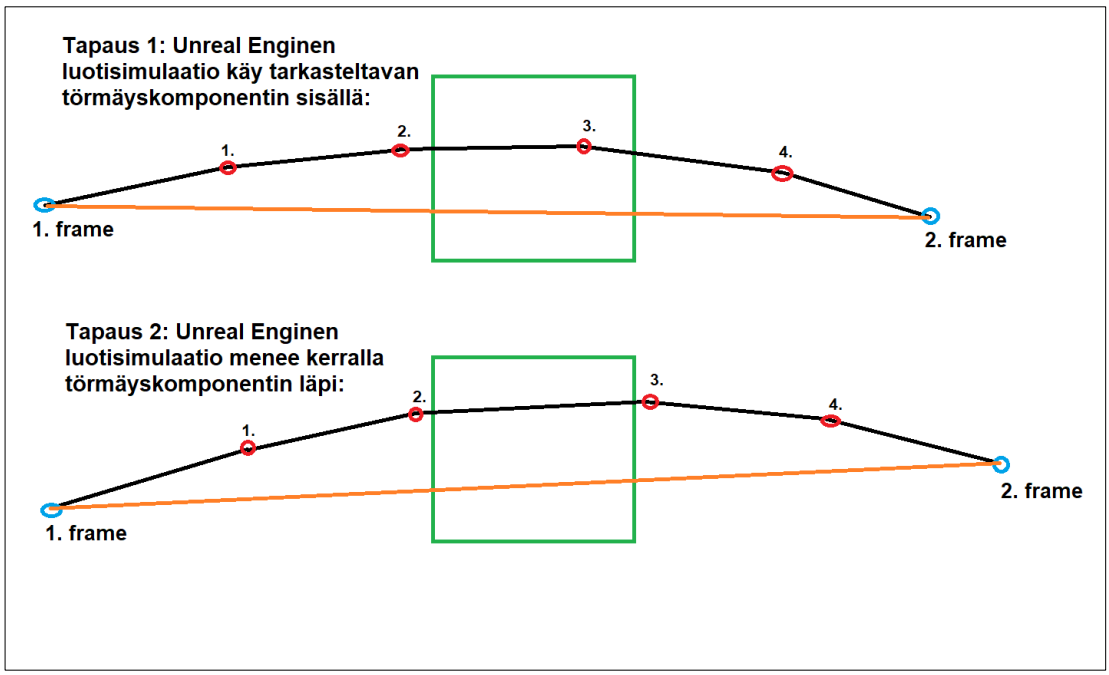
Unreal Enginessä voi säätää jokaisen osumatarkastelun kaksipuoliseksi, eli tässä tapauksessa Begin Overlap -tapahtuma tapahtuisi myös silloin, kun

normaalisti tapahtuisi End Overlap -tapahtuma. Tällä lähestymistavalla saadaan myös luodin poistumiskohdasta tarkka sijainti, jos tarkastellaan, kuinka monesti yhteen komponenttiin on luoti osunut. Ensimmäinen Begin Overlap -tapahtuma on luodin osumakohta, ja toinen Begin Overlap -tapahtuma samaan komponenttiin on luodin poistumiskohta. Vaikka tämä lähestymistapa vaikuttaa täydelliseltä, on siinäkin ongelma: jos luoti on niin nopea tai törmätty komponentti on niin pieni, että Unreal Enginen omissa laskelmissa luoti ei koskaan käy esineen sisällä (kuva 21 tapaus 2). Tässä tapauksessa Unreal Engine ei aktivoi toista Begin Overlap -tapahtumaa luodin poistuessa törmäystarkastelusta, vaikka kaksipuolisella törmäystarkastelulla tämän pitäisi tapahtua. Minulle ei ole tiedossa onko kyseinen toiminnallisuus ohjelmointivirhe vai haluttu toiminnallisuus Unreal Enginen puolelta, mutta tätä tilannetta se ainakin vaikeuttaa huomattavasti.

Koska Unreal Enginen luotien törmäystarkastelu kuitenkin aina tuottaa ensimmäisen Begin Overlap -tapahtuman, vaikka luoti matkaisi kilometrejä yhden ruudunpäivityksen aikana, voi luodin poistumissijainnin löytämiseksi hyödyntää *Predict Projectile Path* -komentoa. Predict Projectile Path -komento laskee luodin ennakoidun liikeradan, palauttaen törmäystarkastelutapahtuman oletetusta osumasijainnista. Ensimmäisen Begin Overlap -tapahtuman aikana voi käyttää Predict Projectile Path -komentoa, joka sitten palauttaisi sijainnin kaksipuolisesta törmäystarkastelusta, johon luoti tulee laskujen mukaan osumaan, jolloin löytyisi myös luodin poistumiskohta. Ongelmaksi tällä kertaa muodostuu Predict Projectile Path -komennon rajoite: Predict Projectile Path -komento ei voi ottaa syötteekseen hakeutuvien luotien maalitettua komponenttia, eli jos impale-erikoisefekti on toiminnassa yhtä aikaa homing-erikoisefektin kanssa, ei Predict Projectile Path -komento laske oikeaa poistumissijaintia luodille.

Sweep Result -rakenne sisältää myös tiedon mistä pisteestä mihin pisteeseen Unreal Engine on törmäystarkastelua tehnyt. Jos hyödynnetään tämän törmäystarkastelun loppupistettä, eli luodin tarkinta mahdollista sen hetkistä laskettua sijaintia, ja tehdään sieltä *Line Trace* kohti luodin osumakohtaa. Line Trace on Unreal Enginen ominaisuus, jolla kehittäjä voi itse tehdä törmäystarkastelun aloituspisteen lopetuspisteen väliltä. Tarkastelemalla Line Tracen aiheuttamaa

törmäystarkastelun tulosta saadaan luodin tarkka poistumiskohta, koska tarkastelun alkupiste ei ole luodin sijainti ruudunpäivityshetkellä, vaan luodin tarkin mahdollinen sijainti alkuperäisen osumahetken jälkeen luotisimulaation aikana (kuva 21 Tapaus 2: luodin sijainti 3). Ongelma tässä on tietenkin taas se, jos luoti on tarpeeksi hidas tai törmätty komponentti tarpeeksi suuri, jää luodin sijainti törmäystarkastelun keskelle, jolloin Line Trace palauttaisi alkuperäisen osumakohdan, koska käytössä on kaksipuoleiset törmäystarkastelut. Unreal Engineltä täytyykin luodin osuessa kysyä, onko törmäystarkastelun loppupiste osutun komponentin sisällä vai ei. Jos luoti on tarkastelussa osutun komponentin sisällä (kuva 21 tapaus 1), ei tehdä Line Tracea taaksepäin kohti alkuperäistä osumaa, koska tiedetään, että luoti tulee vielä osumaan kaksipuoleiseen törmäystarkasteluun, jolloin sieltä saadaan sitten vaadittu tieto luodin poistumiskohdaksi. Jos luoti on tarkastelun aikana jo osutun komponentin ulkopuolella tarkoittaa se sitä, että kaksipuolisen törmäystarkastelun toista osumaa ei koskaan tule tapahtumaan. Tässä tilanteessa tulee tehdä Line Trace kohti alkuperäistä osumapistettä. Kun tämä Line Trace osuu jälleen samaan komponenttiin mihin luoti aiemmin törmäsi, on osumapiste tällöin luodin poistumiskohta. Näin saadaan tässäkin tapauksessa tarkka poistumiskohta luodille ja impale-erikoisefektin laskutoimitukselle.



Kuva 21. Unreal Enginen luodin visuaalinen reitti (framet 1 ja 2) ja ruudunpäivityksien välillä simuloitu reitti (numerot 1-4).

13.2 Physics Constraint -komponentti

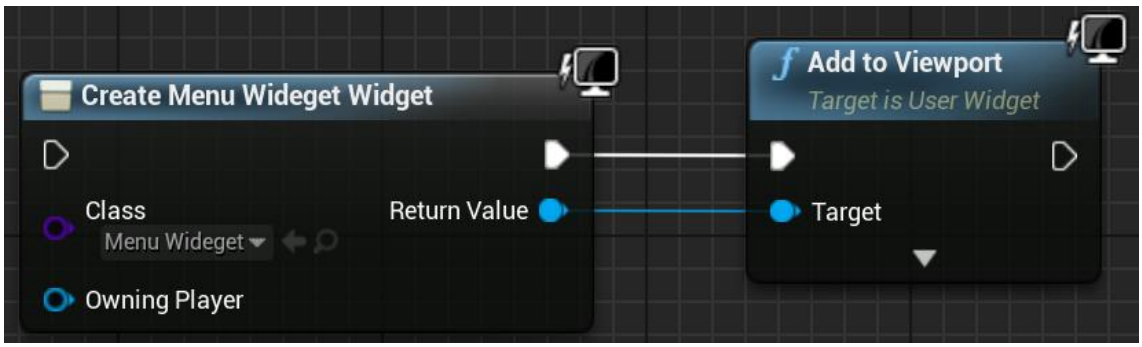
Physics Constraint -komponentti mainittiin luvussa 5.5, ja sitä käytetään esimerkkiprojektissa aseiden ja pelaajan väliseen liitokseen. Projektin toteutuksen aikana physics constraint -komponentti kuitenkin ilmeisesti joidenkin muutosten jälkeen meni rikki. Hajoamisen jälkeen physics constraint -komponentti ei enää tuottanut haluttua fysiikoiden avulla liittämistä. Sen sijaan toiminnallisuus näytti siltä, että physics constraint -komponentti olisi heti liitoksen jälkeen irronnut. Lokitiedostojen mukaan mitään irtoamista ei kuitenkaan tapahtunut. Tällaisessa physics constraint -komponentin hajoamistilanteessa ainoa toimiva ratkaisu oli vanhan physics constraint -komponentin poistaminen ja uuden lisääminen samoilla asetuksilla.

Tämä physics constraint -komponentin kummallinen toiminnallisuus hidasti työntekoa, koska Unreal Enginen puolelta ei tullut mitään virheilmoitusta liittyen physics constraint -komponentin vialliseen toimintaan. Ratkaisin ongelman ainoastaan satunnaisesti testaamalla erilaisia ratkaisuja, ja physics constraint -komponentin uudelleen luominen osoittautui ainoaksi toimivaksi ratkaisuksi.

13.3 Käyttöliittymäelementit VR-ympäristössä

HUD-elementtien tekeminen VR ympäristöön Unreal Enginellä on myös hieman ongelmallista tietyiltä osin. HUD on lyhenne sanoista *heads-up display,* joka videopeleissä tarkoittaa pelaajan ruudulla olevia graafisia elementtejä, jotka antavat pelaajalle informaatiota pelitilanteesta. Tavalliseen, ilman VR-laseja pelattavaan videopeliin, on Unreal Enginessä helppo tehdä HUD-elementtejä luomalla UI-elementtiblueprinttejä eli widgettejä, ja laittamalla ne käyttäjän ruudulle eli viewporttin (kuva 22). UI-elementit ovat käyttöliittymäelementtejä, joiden avulla pelaajalle välitetään informaatiota pelitilanteesta tai pelaaja voi olla vuorovaikutuksessa käyttöliittymäelementin kanssa, ja saavuttaa näin jonkin halutun toiminnon. Esimerkiksi lähes kaikissa videopeleissä peli aloitetaan käyttöliittymän avulla. Unreal Enginessä kuvan 22 mukaisesti pelaajan ruudulle lisätyt UI-

elementit eivät tietenkään jää piiloon pelimaailmassa olevien esineiden taakse, jolloin saavutetaan haluttu HUD toiminnallisuus.



Kuva 22. Widgetin lisääminen käyttäjän viewporttiin.

VR-laseilla asiat hieman muuttuvat, koska VR-lasit vaativat molemmille silmille hieman erilaisen visuaalisen näkökentän, että VR-lasit saavuttavat virtuaaliympäristöefektinsä. UI-elementtien kannalta tämä merkitsee sitä, että kuvan 22 kaltaista järjestelyä ei voi käyttää, koska tällä tavoin lisätyt UI-elementit näyttävät visuaalisesti menevän VR-laseilla aivan vääriin paikkoihin tai vain näkyviin toiselle silmälle. Unreal Enginessä on muutama työkalu, joita käyttämällä UI-elementtejä saadaan myös VR-lasien ruuduille näkyviin. UI-elementin voi luoda objektiksi pelimaailmaan, jolloin se käyttäytyy kuin mikä tahansa muukin esine maailmassa. Tällaisen elementin voi laittaa kiinni pelaajan VR-laseihin, jolloin se seuraa käyttäjän pään liikkeitä, käyttäytyen tavallisen HUD:in tavoin. Mutta tällainen VR-laseihin kiinnitetty elementti voi jäädä piiloon maailman esineiden taakse, koska se ei ole käyttäjän ruudulla vaan maailmassa. Unreal Enginen materiaaliasetuksilla voi kuitenkin säätää UI:n näkymään myös maailmassa olevien esineiden takaa, mutta tätä hyödyntäessäni muut esineet kuitenkin vaikuttivat vielä UI:n ulkonäköön hieman.

Unreal Enginessä on myös *stereo layer* -komponentti, joka on tarkoitettu VR-ympäristöjen HUD toteutuksiin, mutta itse en sitä saanut toiminaan ollenkaan. Käsittääkseni stereo layer -komponentti on Unreal Enginen toteutuksessa vielä hieman keskeneräinen.

13.4 Välilehdet

Unreal Engine 4.24 versiossa on ohjelmointivirhe, jonka takia projektin käynnistyksen yhteydessä viimekerralla auki olleet välilehdet eivät automaattisesti avautuneet uudestaan. Vasta 05.05.2020 julkaistussa Unreal Enginen 4.25 versiossa tämä on korjattu.

Välilehtien uudelleen aukeaminen on minulle tärkeä ominaisuus, koska sen avulla pääsen helposti käsiksi viimekerralla kesken jääneeseen koodiin, sekä saan nopean yleiskäsityksen koodin sen hetkisestä tilanteesta. Toiminnallisuuden puuttuminen hidasti projektin etenemistä, koska minun täytyi aina avata kaikki tarvittavat välilehdet uudelleen, ja en aina ollut varma, mitkä kaikki välilehdet viimeksi olivat auki.

14 Johtopäätökset

Toimiakseen kunnolla virtuaalinen ase tarvitsee vähintään tulitusnopeuden, luodin lähtönopeuden ja -suunnan, rekyylin, hajonnan ja lippaan koon. Vaikka puoliautomaattiset ja kerran laukeavat aseet eivät välttämättä tarvitse tulitusnopeutta toimiakseen, estää tulitusnopeus käyttäjää ampumasta näillä tulitusmekanismeilla liian nopeasti. Vastaavasti virtuaalinen ase ei tarvitse hajontaa, mutta ilman hajontaa luodit olisivat täydellisen tarkkoja. Täydellisen tarkat luodit eivät vastaa oikean maailman aseita, eikä sellaista toiminnallisuutta tässäkään asegeneraattorissa haluttu nähdä. Vahinkoarvoa ase ei toiminnallisuudeltaan välttämättä tarvitse, koska aseilla voidaan tehdä myös esimerkiksi pelkkää tarkkuusammuntaa tai mahdolliset viholliset voivat tuhoutua yhdestä luodin osumasta. Vahinkoarvon käyttö riippuukin täysin projektin muista elementeistä. Kriittisen osuman todennäköisyys, kriittisen vahingon kerroin, panssarin läpäisy, lataamisaika ja panoksen koko ovat arvoja, jotka eivät ole aseen toiminnallisuudelle pakollisia. Vaikka nämä arvot eivät ole pakollisia, antavat ne silti huomattavasti työkaluja satunnaisen aseen luomiseen. Lataamisaika on hieman hassu ominaisuus, koska oikean

maailman aseissa ei ole ennalta määritettyä lataamisaikaa, vaan lataamisajan nopeus riippuu käyttäjän nopeudesta. Asegeneraattorissa nämä aseen lataamiseen liittyvät käyttäjältä vaaditut toiminnot, ovat paljon yksinkertaisempia kuin oikean maailman aseissa. Lataamista voi hidastaa lataamisajan tuomalla animaatiolla, jos se on tarpeellista.

Visuaalisesti asetta voi pilkkoa vielä pienemmäksi, mitä tässä asegeneraattorissa on visuaalisia elementtejä. Tällä määrällä visuaalisia elementtejä saadaan tätä asegeneraattoria varten tarpeeksi erilaisia visuaalisia variaatioita. Mitä enemmän visuaalisia elementtejä generaattoriin tekee, sitä työläämpi ja hankalampi sitä on suunnitella ja toteuttaa. Tämän asegeneraattorin visuaalinen puoli tuntui tällä visuaalisten elementtien määrällä olevan hyvin tasapainossa monimuotoisuuden ja työn määrän kanssa.

Virtuaalinen ase on pilkottu myös toiminnallisesti piippuun, runkoon, lippaaseen ja olkatukeen. Toiminnallisesti aseen voi myös pilkkoa pienemmiksi osiksi, mutta pienemmille osille olisi vaikea keksiä mihin aseen arvoon niiden tulisi vaikuttaa. Nyt jokaisella aseen osalla on selkeä toiminnallisuus, ja ne ovat selkeästi toisistaan eroavia aseen elementtejä.

Erikoisefektien avulla virtuaaliset aseet saavat lisää variaatioita ja toiminnallisuuksia. Mestariase on suunniteltu ottamaan vastaan siihen vaikuttavia arvoja useista eri lähteistä, joten erikoisefektien vaikutuksia aseen perustoiminnallisuuteen on helppo säätää. Erikoisefektit eivät ole myöskään aseen toimimisen kannalta vaadittuja elementtejä, ja ne on helppo jättää pois, jos ne eivät ole tiettyyn tilanteeseen haluttuja.

CSV-taulukot toimivat erinomaisesti projektiin vaadittuun arvojen muokkaamiseen koodin ulkopuolelta. Koodista käsin taulukoiden tietotyypin muuttaminen oli hieman työlästä, koska aluksi taulukoiden arvoja käsiteltiin numeroina, ja ne otettiin Unreal Enginen puolelle käyttöön sellaisenaan. Mutta toiminnallisuudeksi haluttiin arvolla olevan jokin haarukka, minkä väliltä lopullinen arvo satunnaisesti valittaisiin. Tämän takia taulukoita piti käsitellä tekstinä, että samaan

sarakkeeseen saataisiin haarukan pienin ja suurin mahdollinen arvo. Tämä vaati hiukan rakenteellisia muutoksia Unreal Enginen puolella, ja tekstin muuntaminen vaati hiukan lisäfunktioita tilanteen mukaan. Myös Excel hankasi aluksi vastaan, koska siinä tiettyjä merkkejä käsitellään automaattisesti. Tämän takia jotkin CSV-taulukon arvoista muuttui Excelissä kirjoitetuiksi kuukausiksi, eikä tätä ominaisuutta saanut pois päältä Excelin asetuksista. Ratkaisuna käytettiin sellaista merkkiä erottamaan pienin ja suurin arvo, jota Excel ei käsitellyt automaattisesti.

Asegeneraattoria jatkokehitetään kehitystyökaluksi Unreal Engine Marketplaceen tai siitä kehitetään videopeli KollektiWe Oy:n toimesta. Asegeneraattorin toiminnallisuuksia voidaan myös pilkkoa omiksi työkaluiksi, ja näitä työkaluja voi laittaa Unreal Enginen Marketplace -kauppapaikkaan myyntiin omina kokonaisuuksinaan. Selkeitä pilkottavia kohteita ovat esimerkiksi impale-erikoisefektin toiminnallisuus, visuaalinen generointi ja luotien ennakkoon rakentaminen.

Lähteet

Bogost, I. 2008. Persuasive Games: Texture. Gamasutra. https://www.gamasutra.com/view/feature/132053/persuasive_games_texture.php?page=2. 12.5.2020.

Carnes, B. 2019. What is a CSV File and How to Open the CSV File Format. FreeCodeCamp. https://www.freecodecamp.org/news/what-is-a-csv-file-and-how-to-open-the-csv-file-format/. 15.4.2020.

KollektiWe Oy. 2020. GunGen 1.0 Preview. https://youtu.be/3j_PfRxVAGY. 15.5.2020.

Rouse, M. 2016. 3D-mesh. Techtarget. https://whatis.techtarget.com/definition/3D-mesh. 10.4.2020.

Smith, S. 2020. M16A2 and M16A4 5.56mm Rifles: Features and Background. The Balance Careers. https://www.thebalancecareers.com/m16a2-and-m16a4-5-56mm-rifles-features-and-background-3357220. 08.05.2020.

Sobieck, B. 2015. What's the Difference? Machine Gun vs. Submachine Gun. Benjamin Sobieck. https://crimefictionbook.com/2015/10/01/whats-the-difference-machine-gun-vs-submachine-gun/. 8.5.2020.

Stroustrup, B. 2020. Bjarne Stroustrup's FAQ. Stroustrup. http://www.stroustrup.com/bs_faq.html#difference. 8.5.2020.

Sutocren. 2016. This is Unreal. Hotgates. https://hotgates.eu/this-is-unreal/. 10.4.2020.

Unreal Engine. 2019. Why was my submission declined. Epic Games. https://marketplacehelp.epicgames.com/s/article/Why-was-my-submission-declined?language=en_US. 6.5.2020.

Unreal Engine. 2020a. Collision Overview. Epic Games. https://docs.unrealengine.com/en-US/Engine/Physics/Collision/Overview/index.html. 14.5.2020.

Unreal Engine. 2020b. Actors. Epic Games. https://docs.unrealengine.com/en-US/Programming/UnrealArchitecture/Actors/index.html. 21.4.2020.

Unreal Engine. 2020c. Actor Lifecycle. Epic Games. https://docs.unrealengine.com/en-US/Programming/UnrealArchitecture/Actors/ActorLifecycle/index.html. 6.5.2020.

Unreal Engine. 2020d. Setting Up and Using Sockets With Static Meshes. Epic Games. https://docs.unrealengine.com/en-US/Engine/Content/Types/StaticMeshes/HowTo/Sockets/index.html. 26.04.2020.

Unreal Engine. 2020e. Unreal Object Handling. Epic Games. https://docs.unrealengine.com/en-US/Programming/UnrealArchitecture/Objects/Optimizations/index.html. 06.05.2020.

Unreal Engine. 2020f. ELogVerbosity::Type. Epic Games. https://docs.unrealengine.com/en-US/API/Runtime/Core/Logging/ELogVerbosity__Type/index.html. 4.5.2020.

Unreal Engine. 2020g. Marketplace Distribution Agreement. Epic Games. https://www.unrealengine.com/en-US/marketplace-distribution-agreement. 6.5.2020.