TAMPEREEN TEKNILLINEN YLIOPISTO

OLLI MÄNTYLÄ

SUORITUSKYKY JA UNREAL ENGINE 4

Diplomityö

Tarkastaja: prof. Tommi Mikkonen
Tarkastaja ja aihe hyväksytty Tieto- ja sähkötekniikan tiedekuntaneuvoston kokouksessa 8.6.2016.

TIIVISTELMÄ

TAMPEREEN TEKNILLINEN YLIOPISTO
Tietotekniikan koulutusohjelma
MÄNTYLÄ, OLLI: Suorituskyky ja Unreal Engine 4
Diplomityö, 61 sivua, 3 liitesivua
Joulukuu 2016
Pääaine: Ohjelmistotuotanto
Tarkastaja: professori Tommi Mikkonen

Avainsanat: Suorituskyky, profilointi, optimointi, pelimoottori, Unreal Engine 4

Nykyaikana pelinkehityksessä voidaan hyödyntää usein erilaisia pelimoottoreita. Pelimoottorit ovat pelien luontiin tarkoitettuja ohjelmistoja, joihin voi sisältyä sekä ajonaikaisia komponentteja että kehitystyökaluja. Pelimoottorien tarkoituksena on mahdollistaa peleissä käytettyjen ohjelmakomponenttien uudelleenkäyttö uutta peliä luodessa. Koska pelin kehitys tapahtuu pelimoottorin valmiita komponentteja ja mahdollisia työkaluja hyödyntäen, vaikuttavat pelimoottorin sisältämät ominaisuudet usein paljon pelinkehittäjän mahdollisuuksiin.

Pelinkehityksessä suorituskyky on usein keskeisessä asemassa. Jo pelin kehitysvaiheessa on otettava huomioon suunniteltu kohdealusta sekä laitteistovaatimukset, jotta pelistä ei tule missään vaiheessa laskennan kannalta liian raskasta. Myös pelimoottorin valinnalla on kuormittavuuden kannalta merkitystä. Pelimoottorin tarjoamat ominaisuudet voivat myös osaltaan helpottaa optimointia ja auttaa löytämään sopivan kuormitustason eri kohdealustoille. Toisaalta ominaisuuksien puute voi tehdä optimoinnista tai erilaisten kohdealustojen tukemisesta hankalaa.

Tässä diplomityössä perehdytään Unreal Engine 4 -pelimoottoriin sekä sen tarjoamiin ominaisuuksiin ja pelinkehitystyökaluihin. Tarkoituksena on tutustua pääasiassa pelimoottorin sisältämiin profilointityökaluihin sekä tutkia pelimoottorin tarjoamia mahdollisuuksia pelien optimointiin. Lisäksi profilointityökaluja ja optimointikeinoja hyödynnetään optimoimalla työhön valittu esimerkkiprojekti testilaitteistoilla paremmin suoriutuvaksi. Esimerkkiprojektin optimointi antaa mahdollisuuden testata profilointia ja optimointia pelimoottorin työkaluilla käytännössä. Samassa yhteydessä käsitellään myös joitakin profilointiin liittyviä käytännön ongelmia, kuten suorituskertojen toistettavuutta.

Esimerkkiprojektin tapauksessa optimointikohteet löytyivät profilointidatan avulla kohtalaisen helposti. Myös suoriutumiselle asetetut tavoitteet olivat saavutettavissa yksinkertaisten muutosten kautta. Optimointia auttoivat pelimoottorista löytyvät ominaisuudet. Tehdyt muutokset eivät ole kuitenkaan ainoat mahdolliset optimointitavat, sillä tilanteisiin voidaan usein soveltaa erilaisia ratkaisuja. Pelimoottorin tarjoamat työkalut vaikuttavat profilointiin ja optimointiin sopivilta ja monipuolisilta. Profilointidatan luotettavuutta on hankala vahvistaa ja profiloinnin toistomäärien tarvetta hankala arvioida, mutta profilointidatassa ei kuitenkaan näkynyt merkittäviä vaihteluita suorituskertojen välillä.

ABSTRACT

TAMPERE UNIVERSITY OF TECHNOLOGY
Master's Degree Programme in Information Technology
MÄNTYLÄ, OLLI: Performance and Unreal Engine 4
Master of Science Thesis, 61 pages, 3 Appendix pages
December 2016
Major: Software engineering
Examiner: Professor Tommi Mikkonen

Keywords: Performance, profiling, optimization, game engine, Unreal Engine 4

Nowadays, different kinds of game engines can be utilized in game development. A game engine is a piece of software that is used for creating games, and it may include both runtime components and development tools. The purpose of a game engine is to enable reusing software components used in existing games to create a new one. Since the runtime components and the possible development tools of the game engine are used in the development process, the features offered by the game engine often affect the options available for the developer.

Performance is a typical key aspect in game development. One has to take target platform and hardware requirements into consideration as early as in the development, so that at no point the game is too demanding for the hardware. Choosing a suitable game engine is important when considering performance. The features offered by the game engine can support straightforward optimization and help to find a suitable level of processing for different platforms. By contrast, lack of features can make optimization and supporting different platforms more difficult.

In this thesis, features and development tools of the game engine Unreal Engine 4 are studied. The focus is mainly on the performance profiling tools and on examining various methods that the game engine offers for optimizing performance. In addition, the profiling tools and optimization methods are utilized to improve the performance of a selected example project on a test hardware. The example project gives us the opportunity to examine profiling tools and optimization methods in practice. At the same time, it presents us with some practical issues such as repeating the tests.

In the example project's case, the optimization targets were found relatively easily using the profiling data. The performance goals that were set for the project can also be met using simple modifications. The features found in the game engine helped in the optimization process. However, those modifications are not the only way to reach the goal, since it is usually possible to apply different kind of solutions. It is also difficult to confirm the reliability of the profiling data and estimate the sufficient amount of test runs, but the profiling data did not show any significant changes between runs.

ALKUSANAT

Tämän diplomityön aihe syntyi halusta perehtyä nykyaikaiseen pelimoottoriin ja sen käyttöön. Aiheen valintaa ohjasi myös kiinnostus suorituskyvyn optimointiin sekä mahdolliset uudet tekniikat, joita tässä yhteydessä voisi mahdollisesti hyödyntää. Työ on tehty itsenäisesti.

Kiitän erityisesti ohjaajaani professori Tommi Mikkosta, jolta sain työstä ohjauksen kannalta hyvää palautetta ja uusia näkökulmia. Kiitos kuuluu myös kaikille opiskelukavereille sekä kerhoille ja killoille, joiden kanssa olen opiskeluaikanani ollut tekemisissä!

Tampereella, 30.10.2016

Olli Mäntylä

SISÄLLYS

KUVALUETTELO

TERMIT JA NIIDEN MÄÄRITELMÄT

Aktori	On mikä tahansa objekti, joka voidaan sijoittaa pelin kenttään.
Blueprint	Unreal Engine 4:n käyttämä visuaalisen ohjelmoinnin järjestelmä.
Blueprint Editor	On Unreal Engine 4:n työkalu blueprintien muokkaukseen.
Epic Games Launcher	Unreal Engine 4:n työkalu pelimoottoriversioiden asentamiseen, projektien hallintaan sekä kauppapaikan sisällön selaamiseen ja lataamiseen.
Frames Per Second, FPS	Ruudulle piirrettyjen kuvien määrä sekunnissa.
GitHub	Git-versionhallintatyökaluun perustuva yhteisöllinen ohjelmien kehitys- ja hallintapalvelu, https://github.com.
Graafi (*graph*)	Graafi tarkoittaa tässä työssä useimpiin blueprinteihin sisältyvää graafia, joka mahdollistaa logiikan ohjelmoimisen.
Level Editor	On Unreal Engine 4:n keskeinen työkalu pelin kenttien muokkaamiseen.
Linux	Myös *GNU/Linux* [1]. Avoimen lähdekoodin käyttöjärjestelmä.
Näyttöalue (*viewport*)	Alue grafiikan renderöintiin.
Profiler	On Unreal Engine 4:n graafinen suorituskyvyn profilointityökalu.
Profilointi	Tarkoittaa suorituskyvystä puhuttaessa laskentatehoa käyttävien kohteiden analysointia.
Renderöinti (*rendering*)	On digitaalisen mallin generointia piirrettäväksi digitaaliseksi kuvaksi
Tekstuuri (*texture*)	Tekstuuri on kuva, jota käytetään esimerkiksi jonkin materiaalin pintana. Materiaalia puolestaan voidaan käyttää esimerkiksi jonkin objektin pintamateriaalina.
Unreal Engine 4, UE4	Epic Gamesin luoma pelimoottori.
Varjostin, sävytin (*shader*)	On tietokonegrafiikassa grafiikan, esimerkiksi värien tai varjojen, laskentaan tarkoitettu yleensä näytönohjaimen suorittama ohjelma.
Visuaalinen ohjelmointi	On ohjelmointia käyttäen visuaalisia työkaluja tekstipohjaisen ohjelmoinnin sijaan.

1 JOHDANTO

Peleissä suorituskyvyllä ja optimoinnilla on usein suuri merkitys pelin pelattavuuden ja hyvän toimivuuden kannalta. Siksi on myös oleellista päättää, millaiset laitteistovaatimukset aikoo projektilleen asettaa. Peliä suorittavan kohdealustan täyden laskentatehon hyödyntämiseksi on myös huolehdittava sen sopivasta ja tasapuolisesta kuormittamisesta. Laskennan pullonkaulojen löytämiseksi voidaan hyödyntää profilointityökaluja. Profilointityökalut voivat kuulua esimerkiksi pelissä käytetyn pelimoottorin työkaluvalikoimaan. Pelimoottorit ovat ohjelmistoja, joita hyödynnetään pelien luonnissa. Niihin voi sisältyä ajonaikaisia komponentteja sekä erilaisia kehitystyökaluja. Profiloinnista saatujen tuloksien pohjalta päästään tutkimaan erilaisia optimointimahdollisuuksia pullonkaulojen poistamiseksi ja sopivan tasapainon löytämiseksi. Hyvä ja tasainen suoriutuminen tarkoittaa usein myös parempaa käyttäjäkokemusta, mikäli onnistutaan löytämään parempi tasapaino ruutujen piirtotaajuuden sekä kuvanlaadun välille.

Tässä työssä käsitellään Unreal Engine 4 -pelimoottorin käyttöä pelinkehityksessä sekä sitä käyttävien sovellusten suorituskyvyn profilointia ja optimointia suorituskyvyn parantamiseksi. Selkeyden vuoksi työssä keskitytään ensisijaisesti pelimoottorin käyttöön pelinkehityksen näkökulmasta, jossa peleillä tarkoitetaan nimenomaan ja ainoastaan *videopelejä*. Unreal Engine 4:n käyttö ei kuitenkaan periaatteessa rajoitu ainoastaan pelien kehittämiseen, joten tämän työn sisältö on hyödynnettävissä myös muuntyyppisissä Unreal Engine 4:ää käyttävissä projekteissa. Tällaisia ovat pelien lisäksi esimerkiksi 3D-elokuvat, simuloinnit ja erilaiset visualisaatiot tai audiovisuaaliset esitykset ja animaatiot.

Suurin osa tässä opinnäytetyössä käytetyistä lähteistä perustuu *Epic Games*in ylläpitämään Unreal Engine 4:n dokumentaation tuoreimpaan versioon. Dokumentaatio on vapaasti saatavilla ja selattavissa web-selaimella. Dokumentaatiossa tapahtuneiden muutosten jäljittämiseksi voi halutessaan hyödyntää dokumentaation sivua *New and Updated Resources* [2].

Ensimmäiseksi tässä työssä selvennetään sanan *pelimoottori* tarkoitusta luvussa 2. Sanan merkitys ei ole aina ollut täysin vakiintunut ja sillä on saatettu tarkoittaa hieman erilaisia asioita, joten samalla saadaan tarkennettua käsitteen merkitystä. Luvussa käydään läpi myös pelimoottoreihin liittyviä ominaisuuksia ja pelimoottorien arkkitehtuuria.

Luvussa 3 esitellään lyhyesti Unreal Engine 4 -pelimoottoria sekä siihen liittyviä työkaluja ja ominaisuuksia. Myös lisenssi- ja käyttöehtoja käydään läpi, sillä rekisteröity-

minen ja ehtojen hyväksyminen on edellytys pelimoottorin käyttämiselle. Käyttöehdot ja lisensointi ovat myös yksi tekijä, joka erottaa eri pelimoottoreita toisistaan.

Luvussa 4 käydään läpi joidenkin pelimoottoriin liittyvien työkalujen käyttöä. Erilaisia työkaluja on suuri määrä ja kaikkien läpikäynti ei ole tarpeellista, joten luvussa keskitytään vain keskeisiin työkaluihin.

Luvut 5 ja 6 esittelevät tämän jälkeen pelimoottorilla ohjelmoinnin perusasioita. Tässä vaiheessa lukijalla pitäisi olla riittävän hyvä kuva pelimoottorin käyttöliittymän käytöstä sekä pohja mahdollisten muutoksien tekemiseksi projekteihin optimointitarkoituksessa.

Perusteiden läpikäynnin jälkeen edetään työn ydinasian kannalta merkittävämpään aiheeseen eli profilointiin. Profilointia käsittelevä luku 7 kuvaa tarkemmin profiloinnin tarkoitusta ja roolia suorituskyvyn optimoinnissa. Luvussa myös esitellään pelimoottorin tarjoamia profilointityökaluja sekä perehdytään niiden käyttöön tarkemmin.

Profiloinnin jälkeen pitäisi olla tiedossa optimointia vaativat kohteet, joten luvussa 8 edetään käsittelemään erilaisia optimointimahdollisuuksia. Potentiaalisia optimointikohteita voi olla projektissa käytetyistä ominaisuuksista riippuen hyvin suuri määrä, joten työssä pyritään käsittelemään pääasiassa yleisimpiä keinoja ja kohteita, mutta

Luvussa 9 sovelletaan opittuja asioita käytännössä ja valitaan pelimoottorille valmiiksi saatavilla olevaa esimerkkiprojekti tutkittavaksi. Luvussa käydään läpi työssä käytetyt testausympäristöt sekä profiloinnin kannalta oleellisten sovellusten asetukset ja niiden yksityiskohtaisempi käyttö. Tavoitteena on esimerkkiprojektin suorituskyvyn parantaminen työssä käytetyillä laitteilla. Luvussa esitetään konkreettisia esimerkkejä esimerkkiprojektin profiloinnista ja yksinkertaisten optimointikeinojen hyödyntämisestä käytännössä. Luvun lopussa arvioidaan lyhyesti tuloksia sekä niiden luotettavuutta.

Luku 10 sisältää yhteenvedon ja siinä kerrataan lyhyesti työn sisältöä ja pohditaan luvussa 9 suoritettuja mittauksia ja saatuja tuloksia. Luvussa käydään läpi myös joitain työssä kohdattuja haasteita sekä pohditaan muita mahdollisia pelimoottoriin liittyviä tutkimuskohteita.

2 MIKÄ ON PELIMOOTTORI?

Tässä luvussa selvitetään pelimoottorin käsitettä sekä pohjustetaan käsitettä lyhyesti historialla ja alalla toimineiden henkilöiden käsityksillä termin muotoutumisesta. Tämän jälkeen käydään läpi nykyajan pelimoottoreihin liittyviä ominaisuuksia ja arkkitehtuuria karkealla tasolla, mutta kuitenkin tarpeeksi tarkasti riittävän yleiskuvan antamiseksi. Viimeisenä käsitellään suorituskyvyn merkitystä peleissä ja pelimoottorin mahdollista roolia pelin suorituskyvyssä sekä sen parantamisessa.

2.1 Yleistä

Pitkään pelinkehityksen parissa työskennelleen ja niiden arkkitehtuurista kirjoittaneen Jason Gregoryn mukaan käsite ”pelimoottori” alkoi muotoutua noin 1990-luvun puolivälissä. Noihin aikoihin suosittu *id Software*n peli *Doom* oli yksi peleistä, jonka ohjelmistoarkkitehtuurissa korostui ydinkomponenttien modulaarisuus. Näihin komponentteihin kuuluivat muun muassa 3D-mallinnusjärjestelmä, törmäyksentunnistus ja pelin äänijärjestelmä. Modulaarisuus teki lähdekoodin uudelleenkäytöstä helpompaa, jolloin kehittäjät pystyivät luomaan uusia pelejä pienemmällä vaivalla hyödyntäen olemassa olevia komponentteja pelien ytimessä. Myös pelimoottorien lisenssien myynti alkoi olla varteenotettava tulonlähde. [3, s. 1-3][4, s. xiii-12]

Gregoryn mukaan pelin ja pelimoottorin välinen raja on usein häilyvä, ja pelimoottorin modulaarisuus ja yleiskäyttöisyys riippuu pitkälti luodusta toteutuksesta. Gregory itse ehdottaa termin ”pelimoottori” käyttämistä nimenomaan ohjelmistosta, jonka laajentaminen ja uudelleenkäytettävyys uusien pelien perustana onnistuu ilman suuria muutoksia kyseisen ohjelmiston lähdekoodiin. On kuitenkin otettava huomioon, että erilaiset pelimoottorit on usein suunniteltu jollain tasolla tiettyihin käyttötarkoituksiin tai tietynlaisille laitteille, joten ne eivät kuitenkaan sovellu tai suoriudu hyvin aivan kaikissa ympäristöissä. [4, s. 11-13]

Käsitettä ”pelimoottori” on käytetty myös pienempikokoisesta pelimoottorista, joka on vastannut kuitenkin ainoastaan grafiikasta [3, s. 2]. Koska nykyaikainen pelimoottori voi muodostua useista ohjelmistokomponenteista, käytetään tässä työssä tällaisesta ohjelmistosta tai komponentista selkeyden vuoksi sanaa *grafiikkamoottori*.

2.2 Ominaisuudet ja arkkitehtuuri

Pelimoottori sisältää yleensä kehitysvaiheeseen tarkoitetut kehitystyökalut sekä ajonaikaiset ohjelmistokomponentit [4, s. 31-32]. Muiden uudelleenkäytettävyyteen tähtäävien ohjelmistojen tapaan pelimoottorin arkkitehtuuri on usein kerroksittainen ja modulaarinen. Rakenteessa ja ydinkomponenteissa on karkealla tasolla havaittavissa samankaltaisuuksia eri pelimoottorien välillä. Muiden ohjelmistojen tapaan pelimoottorin toiminta rakentuu pohjimmiltaan käyttöjärjestelmän rajapintojen ja käyttöjärjestelmän käyttämien laitteisto-ohjaimien päälle. Seuraavilla tasoilla ovat karkeasti ottaen erilaisista ydintoiminnoista, kuten muistin- ja tiedostonhallinnasta vastaavat kirjastot ja rajapinnat, matalan tason grafiikkakirjastot, fysiikkamoottori ja matematiikkakirjastot. Vasta näiden kerroksien päälle muodostuu varsinainen kehittäjälle näkyvä pelimoottori ja sen ohjelmointirajapinnat.[4, s. 3-4] [4, s. 28-49]

2.2.1 Ajonaikaiset komponentit

Ajonaikaisista komponenteista pelaajalle näkyvin osa pelistä on pelimoottorin ruudulle tuottama grafiikka. Grafiikan hallinnasta vastaa grafiikkamoottori. Grafiikkamoottorin ominaisuuksiin voivat kuulua esimerkiksi geometristen muotojen piirtäminen sekä varjojen ja heijastusten laskenta. Grafiikkamoottori itsessään on voitu jakaa useisiin komponentteihin, joista yksi esimerkiksi vastaa visuaalisista tehosteista. Gregoryn mukaan suurin osa grafiikkamoottoreista hyödyntää pohjalla matalamman tason kirjastoja ja ohjelmointirajanpintoja grafiikkaohjaimen kanssa keskustelemiseen. Tällä tavoin luotu korkeampi abstraktiotaso auttaa piilottamaan laitteistoon ja grafiikan laskentaan liittyviä yksityiskohtia ohjelmoinnin yksinkertaistamiseksi [3, s. 173-174]. Alkuun tämä tarkoitti luopumista ominaisuuksista joita grafiikkakirjasto ei tukenut (niin kutsuttu *fixed-function pipeline*), mutta myöhemmin kehittynyt varjostin (tai sävytin) -ohjelmointi (engl. *shader programming*) on mahdollistanut myös itse määriteltyjen ohjelmien ajamisen näytönohjaimella ja on noussut nykyään merkittävään rooliin [5, s. xxiv-5]. Laajasti käytössä olevia 3D-grafiikkakirjastoja ovat esimerkiksi *OpenGL* ja Microsoftin *DirectX*. [5, s. xxiv-5][4, s. 31-40]

Toinen merkittävä kokonaisuus pelimoottoreissa on fysiikkamoottori. Kaupallisen fysiikkamoottorin kehittäneen ja sen ohjelmoinnista kirjan kirjoittaneen Ian Millingtonin mukaan juuri fysiikkamoottori on yksi aiempina vuosina merkittävyyttään kasvattanut pelimoottorin komponentti [5, s. 2]. Fysiikkamoottori erikoistuu fyysisten tapahtumien, kuten esimerkiksi törmäyksien, mallintamiseen. Erilaisia loogisia osa-alueita varten on fysiikkamoottoriin voitu grafiikkamoottorin tavoin tehdä omat komponenttinsa. Esimerkkejä fysiikkamoottoreista ovat *Havok* ja *PhysX*. [4, s. 32][6, s. 1-5]

Muita keskeisiä pelimoottorin ominaisuuksia voivat olla muun muassa komponentit hahmojen animointiin, työkalut tekoälyn helpompaan luomiseen, äänijärjestelmä ja mo-

ninpeliominaisuudet paikallisen moninpelin tai verkkopelaamisen mahdollistamiseksi. [4, s. 28-49]

2.2.2 Kehitystyökalut

Pelimoottoriin kuuluvat usein ajonaikaisten komponenttien lisäksi kehitystyökalut, joiden tarjonta ja monipuolisuus riippuu käytetystä pelimoottorista. Pelit ovat käytännössä multimediasovelluksia, joten materiaalien (engl. *asset*) luomiseen tarvitaan avuksi yleensä muita sovelluksia. Nämä sovellukset ovat usein erikoistuneet pääasiassa tietyntyyppisen sisällön luomiseen, mutta ovat yleiskäyttöisiä eivätkä pelkästään peleihin liittyvän materiaalin luontiin. Esimerkkejä sovelluksista ovat muun muassa seuraavat:

- Kuvankäsittelyohjelmat kuvien ja tekstuurien luontiin.
- 3D-mallinnusohjelmat 3D-mallien ja/tai luurankoanimaatioiden (engl. *skeletal animation*) luontiin. Luurankoanimaatioissa 3D-mallin liikkeiden mallinnuksessa ja animoinnissa käytetään apuna ikään kuin luurankomaista rakennetta mallin sisällä. Tällöin voidaan määritellä esimerkiksi nivelkohtia ja miltä mallin pitäisi näyttää luurangon osien ollessa erilaisissa asennoissa.
- Äänenkäsittelyohjelmat, pelin äänien muokkaamiseen.

Itse pelimoottorin tarjonnasta voi löytyä työkaluja myös erikoisempiin ja pelikeskeisempiin tarkoituksiin, kuten maaston ja muun pelimaailman luomiseen tai pelin suorituskyvyn profilointiin. [4, s. 49-56]

2.3 Syitä erilaisille pelimoottoreille

Yksittäinen pelimoottori on usein suunniteltu tietynlaisten pelien luontiin ja tietyille kohdealustoille. Pelimoottori voi olla suunniteltu esimerkiksi erityisesti 2D- tai 3D-pelejä varten ja alustaksi tarkoitettu esimerkiksi mobiililaitteet tai pelikonsolit. Tietynlaisiin pelityyppeihin erikoistuneet pelimoottorit voivat puolestaan olla tarkoitettu esimerkiksi strategiapelien tai autopelien luontiin. [4, s. 11-14]

Pelimoottorien monimuotoisuuden taustalla on useita syitä. Millington on käsitellyt kirjassaan yleiskäyttöisen fysiikkamoottorin hyödyntämisen etuja ja haittoja, mikä voi osaltaan auttaa ymmärtämään näitä syitä paremmin. Esimerkiksi yksinkertaista fysiikkaa vaativalle pelille voi oman toteutuksen ohjelmointi fysiikkamallinnusta varten olla usein riittävä ratkaisutapa. Vaatimusten kasvaessa fysiikan monimuotoisuus ja monimutkaisuus tekee mallinnuksen ohjelmoinnista työlästä ja hankalaa, jolloin toteutuksen muotoilu modulaarisemmaksi fysiikkamoottoriksi voi olla hyödyllisempää uudelleenkäytettävyyden vuoksi. Tällaisessa tilanteessa voi olla myös helpompaa ja nopeampaa käyttää jotakin valmista fysiikkamoottoria. [4, s. 11-25][6, s.1-5]

Yleiskäyttöisen fysiikkamoottorin yleisin heikkous on Millingtonin mukaan kuitenkin nopeus. Fysiikkamoottori ei voi olla toiminnaltaan yhtä erikoistunut, vaan saattaa suorittaa pelin kannalta tarpeettoman monimutkaisia laskuoperaatioita. Toinen heikkous

voi olla vaaditun fysiikkamallinnuksen yksinkertaisuus: voi olla helpompaa toteuttaa yksinkertainen simulaatio itse kuin yrittää konfiguroida valmista fysiikkamoottoria toimimaan halutulla tavalla. Vastaavat syyt pätevät myös erikoistuneempaan pelimoottoriin: reaaliaikaisten strategiapelien luontiin (*real-time strategy, RTS*) suunniteltu pelimoottori on todennäköisesti tarkoitukseen paremmin optimoitu kuin yleisesti 3D-peleihin suunniteltu pelimoottori. Lisäksi mahdollinen pelimoottorin tarjoama kehitystyökalu ja sen käyttöliittymä on todennäköisesti tarkoitukseen sopivampi ja yksinkertaisempi. Kolmantena syynä voidaan pitää pelinkehittäjän resurssien säästöä: omaa fysiikkamallinnusta luodessa modulaarisen fysiikkamoottorin ohjelmoinnin voi nähdä vaativan turhaan ylimääräistä vaivaa. Tämä voi olla merkittävä seikka, jos kyseessä on esimerkiksi harrastelijaohjelmoijan peliprojekti. [4, s. 11-25][6, s.1-5]

Yllä mainitut perustelut tuskin kattavat kaikkia syitä, jotka ovat voineet johtaa erilaisiin arkkitehtuureihin tai päätöksiin jonkin komponentin laajuudesta ja erikoistumisen asteesta. Pyrkimyksenä on kuitenkin aada asiasta riittävä hyvä yleiskäsitys.

2.4 Suorituskyvyn merkitys peleissä

Useimmat 2D- ja 3D-pelit ovat reaaliaikaisia. Myös vuoropohjaiset pelit sisältävät usein ainakin reaaliaikaisesti toimivan käyttöliittymän. Reaaliaikajärjestelmiä yhdistävä tekijä on suoriutumiselle asetetut aikamääreet. Tämän vuoksi suorituskyvyllä on usein merkittävä rooli pelin toiminnan kannalta, vaikkakin pelien tapauksessa simuloinnin huono suoriutuminen todennäköisesti ei ole kenellekään vaaraksi. [4, s. 9-11]

Hyvänä esimerkkinä peleihin liittyvistä suoritusvaatimuksista on kuvan päivitystaajuus. Kuvaa piirtävän laitteiston ja sitä käyttävän grafiikkamoottorin ohjelmistokomponentteineen tulee pystyä piirtämään riittävän monta kuvaa sekunnissa saadakseen aikaan vaikutelman sulavasta liikkeestä. Minimivaatimus päivitystaajuudelle voi olla esimerkiksi 24 kuvaa sekunnissa. Vastaavia vähimmäisvaatimuksia voi löytyä myös muista pelistä löytyvistä järjestelmistä. Tällaisia voivat olla esimerkiksi fysiikkamallinnus ja tekoäly. Fysiikkamoottorilla voi olla sen vaatima vähimmäispäivitystaajuus, jotta sen suorittama mallinnus pysyy tarkkana ja toimii oikein. Vastaavasti tekoälyn täytyy suoriutua riittävän hyvin tehdäkseen pelin kannalta hyviä päätöksiä riittävän lyhyessä ajassa. [7] [4, s. 9-11]

Kehittäjän täytyy tarvittaessa pystyä selvittämään optimointia vaativat kohteet pitääkseen suoritusajat aiemmin mainituissa rajoissa. Apuna voidaan käyttää erilaisia profilointityökaluja, joilla saadaan analysoitua ja eriteltyä eri asioiden laskentaan kulunut aika.

3 UNREAL ENGINE 4:N ESITTELY

Unreal Engine 4 (UE4) on Epic Gamesin luoma monipuolinen pelimoottori. UE4 soveltuu muun muassa 2D- ja 3D-pelien tekemiseen niin pelikonsoleille kuin mobiililaitteillekin. Käyttäjäkuntaa ei ole erityisesti rajattu, vaan UE4 on suunnattu niin opiskelijoille, itsenäisille pelinkehittäjille kuin ammattilaisillekin. Pelimoottorin C++-lähdekoodi on haettavissa maksutta. [8]

Tässä luvussa esitellään lyhyesti Unreal Engine 4 -pelimoottoriin liittyviä työkaluja ja ominaisuuksia sekä käydään läpi tuetut alustat. Lopuksi käsitellään lisensointiin liittyviä ehtoja ja rajoituksia.

3.1 Keskeiset ominaisuudet

UE4:n kanssa käytettävä päätyökalu on graafisella käyttöliittymällä varustettu *Unreal Editor*. UE4:ään sisältyy suuri joukko ominaisuuksia, joita käyttäjä voi hyödyntää esimerkiksi pelien, 3D-elokuvien, simulaattoreiden ja visualisaatioiden luonnissa. Pelimoottorin keskeisiin ominaisuuksiin kuuluvat muun muassa:

- Grafiikkamoottori, joka hyödyntää useita DirectX:n tai OpenGL:n renderöintiominaisuuksia alustasta riippuen.
- Fysiikkamallinnus. Fysiikkamoottorina toimii PhysX.
- Äänijärjestelmä.
- Tekoälyjärjestelmä.

Itse Unreal Editoriin sisältyy lisäksi erilaisia työkaluja helpottamaan pelinkehitykseen liittyvien resurssien luontia. Näihin kuuluvat esimerkiksi grafiikan mallintamisen avuksi tarkoitetut työkalut maaston, kasvillisuuden, materiaalien ja visuaalisten tehosteiden luontiin. Yksi merkittävimmistä ominaisuuksista on myös visuaalinen ohjelmointi niin kutsuttujen *blueprint*ien avulla. Myös tämän työn kannalta oleelliset suorituskyvyn analysointiin käytettävät profilointityökalut ovat UE4:ssa valmiiksi saatavilla. Kehitystä ei ole myöskään rajoitettu pelkästään 3D-sovelluksiin. Myös 2D- sekä 2D/3D-hybridipelien luominen onnistuu Unreal Enginestä löytyvällä *Paper 2D* -työkalulla, joka on tarkoitettu 2D-grafiikan hallintaan ja animointiin [9]. Mainitut ominaisuudet eivät muodosta kattavaa listaa, mutta antavat paremman kokonaiskuvan pelimoottorin monipuolisuudesta ja sen ominaisuuksista. [10][11][12]

3.2 Graafiset työkalut

Unreal Editoriin kuuluu useita graafisia työkaluja eri tarkoituksiin. Täydellinen lista graafisista työkaluista on lueteltu taulukossa 1 [13]. Taulukko antaa hyvän käsityksen Unreal Engineen kuuluvasta kattavasta työkalukokoelmasta. Näiden työkalujen lisäksi UE4:n kanssa käytettävä oleellinen sovellus on *Epic Games Launcher*, joka toimii pelimoottorin graafisena asennustyökaluna, projektien hallintatyökaluna sekä työkaluna kauppapaikan sisällön selaamiseen ja lataamiseen. [13][14]

Taulukko 1: Unreal Editoriin kuuluvat graafiset työkalut ja käyttötarkoituksen kuvaus

Työkalun nimi	Kuvaus
Level Editor	On keskeinen työkalu pelin kenttien muokkaamiseen.
Blueprint Editor	On työkalu blueprintien muokkaukseen.
Material Editor	On tarkoitettu visuaalisen pintamateriaalin luontiin ja muokkaukseen.
Behavior Tree Editor	Auttaa tekoälyn ohjelmoinnissa.
Persona Editor	Sisältää kokoelman työkaluja objektien animaatioiden ja luurankojen muokkaamiseen.
Cascade Editor	On työkalu hiukkaspohjaisten tehosteiden (particle effects) muokkaukseen
UMG UI Editor	On työkalu käyttöliittymäelementtien, kuten valikoiden, luontiin.
Matinee Editor	On työkalu dynaamisten pelitapahtumien luontiin ja objektien animointiin.
Sound Cue Editor	Käytetään pelin äänien ja äänitehosteiden ohjauksessa.
Paper2D Sprite Editor	On työkalu 2D-pohjaisen sprite-grafiikan muokkaukseen.
Paper2D Flipbook Editor	On työkalu 2D-pohjaisen sprite-animaatioiden muokkaukseen.
Physics Asset Tool Editor	On tarkoitettu fysiikkamallinnettavien objektien luontiin luurankojen avulla.
Statish Mesh Editor	On työkalu staattisten objektien luontiin.
Media Player Editor	Auttaa mediatiedostojen lisäämisessä projektiin.

3.3 Alustatuki

Unreal Engine 4 mahdollistaa sovellusten luonnin useille eri alustoille [15]. Tuettuja alustoja ja käyttöjärjestelmiä ovat:

- Microsoft Windows, Mac OS X, Linux ja SteamOS
- Mobiililaitepuolelta Android
- Virtual Reality -alustat, muun muassa HTC Vive (SteamVR), Morpheus, Oculus Rift ja Gear VR
- Pelikonsoleista PlayStation 4 ja Xbox One
- HTML5 (web-sovellukset).

Unreal Editor puolestaan on virallisesti tuettu Microsoft Windowsilla ja Mac OS X:llä sekä pelikonsoleista Xbox Onella ja PlayStation 4:llä maksutta [15]. Linux-käyttöjärjestelmän tuki on UE:n wiki-sivun mukaan pääosin toimiva [16].

3.4 Lisenssi- ja käyttöehdot

Unreal Engine 4:n sekä sen lähdekoodin käyttö edellyttää loppukäyttäjän lisenssisopimuksen, EULA:n (*End User License Agreement*), tuntemisen ja hyväksymisen. Lähdekoodin osalta UE4 on periaatteessa avoin ja lähdekoodi on haettavissa maksutta, mutta avoimuudesta huolimatta käyttöä ja sovellusten lisensointia rajoittaa aiemmin mainittu EULA. Epic Gamesin mukaan ehdot kuitenkin pyrkivät rajoittamaan käyttöä mahdollisimman vähän. [15]

Käyttöehdot eivät salli UE:n lähdekoodin, työkalujen tai *Marketplace*:n sisällön levitystä sellaisenaan käyttäjille, jotka eivät ole itse hyväksyneet käyttöehtoja ja joilla ei siten ole oikeuksia kyseiseen materiaaliin. Käyttöehtojen vuoksi UE4:llä tehtyjen sovellusten lisensointiin liittyy rajoite: se ei ole yhteensopiva esimerkiksi lisenssien *GNU General Public License* (GPL [17]) tai *Creative Commons Attribution-ShareAlike License* kanssa, sillä kyseiset lisenssit vaativat sovelluksen lähdekoodin ja muun materiaalin julkistamista. Lisenssi *Lesser GPL* (LGPL) on mahdollinen jos sovellukseen liitetty LGPL-lisensoitu sisältö liitetään ohjelmaan vain jaettuna kirjastona. Työsuhteet ja sopimukset pelin kehitystarkoituksessa muodostavat ehtoihin poikkeuksen. Myös esimerkkikoodin ja -ohjelmien alilisensointi ja levitys on luvallista. Itse pelimoottorin lähdekoodia voi liittää keskustelutarkoituksessa julkisille foorumeille korkeintaan 30 rivin osissa. [18][19]

Avoimen lähdekoodin projekteja varten käyttäjä voi lisensoida oman lähdekoodinsa esimerkiksi *MIT License* -lisenssillä, joka on yhteensopiva UE4:n kanssa, sillä lisenssi ei aseta ehtoja sovelluksen muiden osien, kuten käytetyn pelimoottorin, lisensoinnin suhteen [19][20]. Muita yhteensopivia lisenssejä ovat esimerkiksi *Apache License*, *BSD License* ja *Microsoft Public License* [19].

4 UNREAL ENGINE 4:N TYÖKALUJEN PERUSTEITA

Unreal Enginen tarjoamien työkalujen ansiosta käyttäjä pystyy luomaan monenlaista sisältöä graafisia käyttöliittymiä hyödyntäen. Pelimoottorilla kehittäminen ei siten välttämättä vaadi lähdekoodin kirjoittamista tekstimuotoon. Tämä luku käy lyhyesti läpi Unreal Engine 4:n asennusprosessin sekä esittelee uuden projektin luonnin ja Unreal Editorin käyttöliittymän käytön perusteita. Lopuksi käydään läpi vaiheet projektin testaamiseksi erilaisilla kohdelaitteilla sekä projektin paketoimiseksi valmiiksi ja itsenäiseksi ohjelmakokonaisuudeksi valitulle alustalle.

4.1 Unreal Engine 4:n asentaminen

Asentaminen vaatii käyttäjältä rekisteröitymisen ja käyttöehtojen hyväksymisen. Tämän jälkeen käyttäjä saa tarvittavat oikeudet asennusohjelman lataamiseksi ja tarvittaessa myös pelimoottorin lähdekoodin lataamiseksi. Microsoft Windows -käyttöjärjestelmällä asennusohjelman avulla asennetaan ensin sovellus *Epic Games Launcher*, jonka avulla käyttäjä voi hallita asennettuja pelimoottorin versioita sekä luotuja projekteja. Työkalulla käytetään lisäksi *Marketplace* -kauppapaikkaa, jonka kautta voidaan ostaa ja ladata uutta sisältöä kehitystä varten. Yksityiskohtaiset asennusohjeet ovat nähtävissä Unreal Enginen dokumentaatiossa. [14]

Tämän työn kirjoittamisen aikana valmiita binääripaketteja Linux-käyttöjärjestelmälle ei ollut saatavilla. Myöskään Epic Games Launcher -sovellusta ei ollut vielä saatavilla [16][21]. Asentaminen suoritettiin siten kääntämällä pelimoottori lähdekoodista, joka on saatavilla GitHub-palvelussa. Version 4.10 lähdekoodin hakeminen, kääntäminen ja Unreal Editorin käynnistäminen tapahtuu komentorivillä ohjelmassa 1 esitetyillä komennoilla:

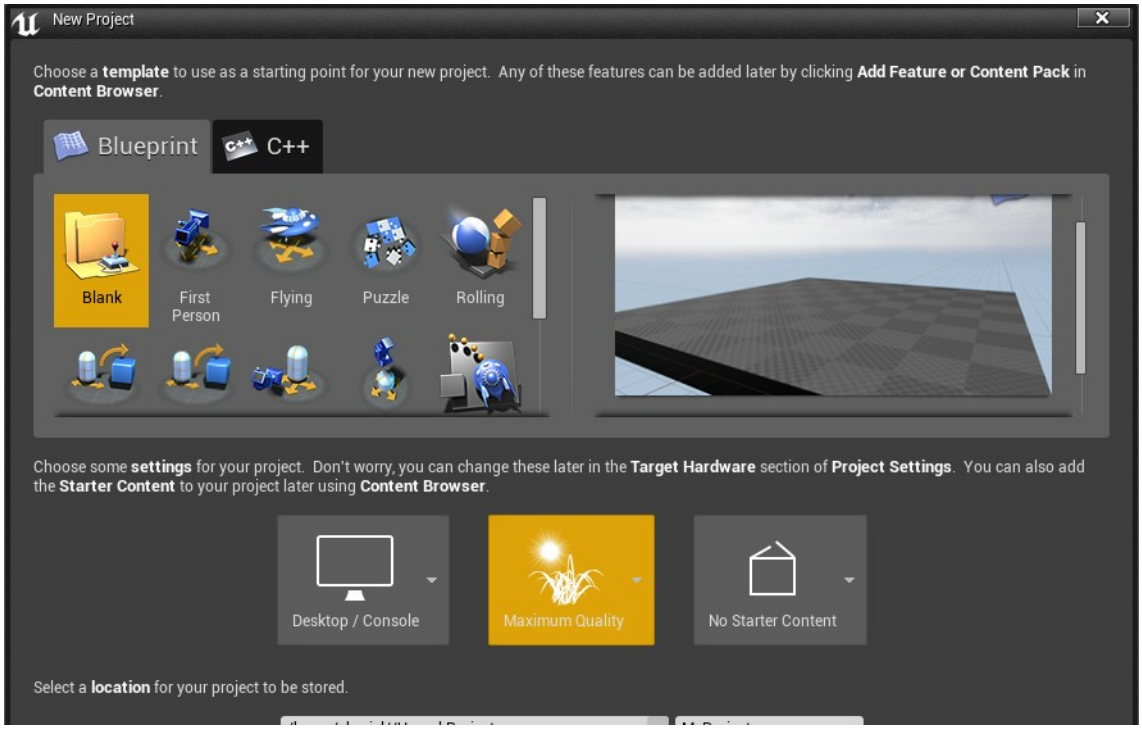
```
1.  git clone -b 4.10 https://github.com/EpicGames/UnrealEngine.git
2.  cd UnrealEngine/
3.  ./Setup.sh
4.  ./GenerateProjectFiles.sh
5.  make UE4Editor UE4Game UnrealPak CrashReportClient ShaderCompileWorker
    UnrealLightmass
6.  cd Engine/Binaries/Linux/
7.  ./UE4Editor
```

Ohjelma 1: Komennot pelimoottorin kääntämiseksi Linux-käyttöjärjestelmällä

Ajantasaiset ohjeet kääntämiseen löytyvät Unreal Engine 4:n dokumentaatiosta [22] sekä UE:n wiki-sivuilta [23]. Dokumentaatio varoittaa, että Linux-tukea muille jakeluille tai versioille kuin kehitys- ja tukitiimin käyttämälle uusimmalle Ubuntu Linux -käyttöjärjestelmälle ei välttämättä voida tarjota [22]. Viitteitä uuden sisällön ostamiseksi ja lataamiseksi muuta kautta kuin virallista Epic Games Launcher -sovellusta käyttämällä ei löytynyt. Työtä tehdessä testattiin, että sisältöä ja projekteja on mahdollista siirtää Linux-järjestelmän puolelle itse, mutta kaikki sisältö ei ole virallisesti yhteensopivaa.

4.2 Uuden projektin luonti

Ohjelmointi UE4:ssä alkaa uuden projektin luomisella, ellei kyseessä ole olemassa olevan projektin muokkaus. Uuden projektin luonti sekä olemassa olevan projektin muokkaus voidaan aloittaa käyttäen joko Epic Games Launcher -sovellusta tai käynnistämällä Unreal Editor -työkalu suoraan. Unreal Editorin käynnistämisen jälkeen käyttäjälle esitetään graafinen dialogi, josta hän voi valita aiemmin luomansa projektin tai luoda uuden *New Project* -välilehdeltä. Uutta projektia luodessa käyttäjä voi valita useista valmiista mallipohjista projektin tyypin mukaan. Valinnat uutta projektia luodessa ovat nähtävillä kuvassa 1.



Kuva 1: Näkymä uutta projektia luodessa (Linux-versio)

Mallin pohjalta tyhjään projektiin luodaan pelityypille tyypillisiä elementtejä ja asetuksia. Ikkuna antaa myös mahdollisuuden valita valmiiksi luotavat elementit joko blueprinteinä tai C++:na ja kohdealustan. Ikkunassa on myös erikseen huomautettu, että

nämä valinnat eivät rajoita projektissa käytettäviä tai siihen myöhemmin lisättäviä ominaisuuksia ja ovat muutettavissa jälkikäteen.

4.3 Level Editor

Projektin auettua käyttäjä saa ensimmäisenä eteensä *Level Editor* -työkalun käyttöliittymän, joka on pääasiallinen työkalu pelin kenttien muokkaamiseen. ”Kenttä” (engl. *level*) on tässä tapauksessa Unreal Editorin yleisesti käyttämä nimitys englanninkieliselle termille *scene*, joka voi olla joillekin tutumpi termi. Kenttiin sijoitettavissa olevia objekteja kutsutaan puolestaan aktoreiksi (engl. *actor*). [24] [25]

Käyttöliittymän näkymä on järjestelty erilaisiin suorakulmion mallisiin välilehdellisiin ikkunoihin, jotka kuvaavat muun muassa erilaisia kenttään lisättäviä objekteja, työkaluja tai kentästä jo löytyviä objekteja ja niiden parametreja. Aivan käyttöliittymän yläreunassa on välilehtipalkki, jossa näkyy muokattavana oleva kenttä. Ikkunoiden sisältöä kuvaavan nimen voi nähdä ikkunan näyttämässä välilehdessä. Muita käyttöliittymään kuuluvia ikkunoita ovat:

- *Modes*, käsittää erilaisia valittavia muokkaustiloja, joiden kautta käyttäjä pääsee käsiksi erikoistuneempiin työkaluihin esimerkiksi maaperän tai kasvillisuuden muokkaamiseksi.
- *Content Browser*, nimensä mukaisesti sisältöselain, jonka avulla käyttäjä voi selata muun muassa kenttiä tai kentissä käytettävää sisältöä.
- *Toolbar*, päätyökalupalkki (nähtävissä myös kuvassa 3).
- *Viewport*, näyttöalue, kentästä renderöity interaktiivinen näkymä.
- *World Outliner*, esittää hierarkisen näkymän kenttään kuuluvista aktoreista. Ikkuna on nähtävissä kuvassa 2.
- *Details*, näyttää näyttöalueen nykyistä valintaa koskevat tiedot, valinnat ja toiminnot.

Selkeyden vuoksi ikkunoiden nimet ovat tässä listassa englanniksi. Kuvan käyttöliittymästä kokonaisuudessaan voi nähdä liitteessä 1, jossa myös kyseiset ikkunat ovat nähtävissä. Esimerkkinä ikkunoiden ulkoasusta ja rakenteesta toimii *World Outliner*, joka on esitetty kuvassa 2. [25]

Kuva 2: Kenttään kuuluvia objekteja kuvaava World Outliner

Käyttöliittymässä keskiössä sijaitsevan pelin testaamiseen tarkoitetun näyttöalueen (*viewport*) yläpuolella olevassa työkalupalkissa puolestaan on rivi tärkeimpiä ja yleisimmin käytettyjä toimintopainikkeita. Näihin kuuluvat tallennus-, asetus- sekä ohjelman kääntämisessä ja testaamisessa käytetyt *build*, *play* ja *launch* -painikkeet [25]. Palkki on nähtävillä kuvassa 3.

Kuva 3: Level Editorin työkalupalkki painikkeineen

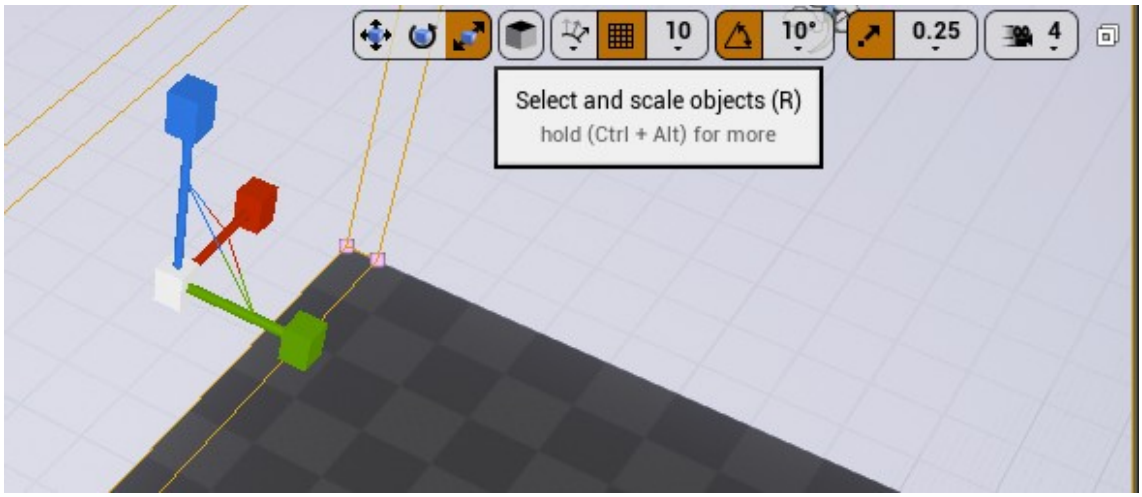
Käyttöliittymän personoimiseksi voi näiden suorakulmaisten alueiden kokoa ja sijaintia muokata vapaasti raahaamalla osoittimella välilehteä, jossa näkyy sisältöä kuvaava nimi. Päävalikon *Window*-valikon kautta voidaan hallita näytettäviä ikkunoita.

4.4 Aktorien hallinnan ja muokkauksen perusteita

UE4:ssä aktorilla tarkoitetaan mitä tahansa objektia, joka voidaan sijoittaa pelin kenttään. Aktori on yleinen luokka, joka tukee 3D-maailmaan liittyviä muunnoksia, kuten liikkumista ja pyörimistä. Aktoreiden ja muun sisällön lisääminen pelikentälle onnistuu suoraviivaisesti käyttämällä graafista käyttöliittymää ja raahaamalla haluttuja objekteja hiirellä näyttöalueelle. Objekteja voidaan lisätä kenttään *Content Browser* -ikkunasta tai *Modes* -ikkunasta *Places*-tilan ollessa valittuna. Tässä tilassa käyttäjä voi hyödyntää joitakin yksinkertaisia valmiiksi luotuja aktoreita. Aktoreiden luonti onnistuu myös ohjelmallisesti blueprintejä tai C++:aa käyttäen funktiota `SpawnActor()`. [26][24]

Kenttään sijoittamisen jälkeen voidaan kyseisten instanssien ominaisuuksia hallita usealla tavalla. Kuvasta 4 nähdään näyttöalueen oikean yläkulman painikkeet, joista vasemmanpuoleisin ryhmä sisältää toimintoja näyttöalueelta valittujen aktoreiden sijain-

nin, kulman ja koon manipulointiin. Graafisten työkalujen lisäksi attribuutteja voidaan hallita myös säätämällä arvoja *details*-ikkunasta tai muuttamalla niitä ohjelmallisesti. [26]



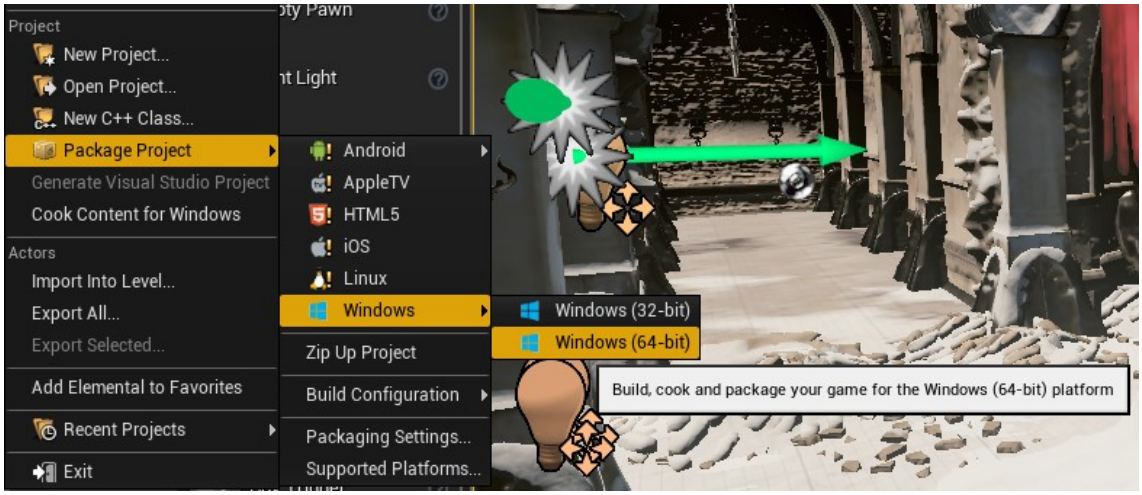
Kuva 4: Näyttöalueen yläreunan työkalut aktoreiden manipuloimiseksi ja näyttöalueen käyttäytymisen hallitsemiseksi.

Kuvassa näkyy myös valittu aktori, joka on tässä tapauksessa läpinäkyvä seinä, sekä koordinaattiakselien suuntaiset kahvat. Kahvojen avulla käyttäjä voi hallita esimerkiksi aktorin kokoa tai sijaintia sekä kääntää sitä.

4.5 Projektin testaaminen ja paketointi

UE4:n käyttöliittymä antaa mahdollisuuden kokeilla peliprojektia heti *play*-painikkeen avulla, jolla suoritusta voi joko simuloida suoraan Editorissa (niin kutsuttu *Play In Editor* -ominaisuus) tai käynnistää pelin itsenäisenä sovelluksena [27]. Painike sijaitsee päätyökalupalkissa ja on nähtävissä kuvassa 3. Useille laitteille ohjelmoitaessa kehittäjä pystyy kokeilemaan miltä ohjelma näyttää esimerkiksi mobiililaitteella myös käytännössä. Tähän tarkoitukseen käyttöliittymässä on *Launch*-painike, jolla projekti ja sillä hetkellä muokkauksessa oleva kenttä valmistetaan (dokumentaatio käyttää termiä *cooking*) kohdealustaa varten. Tällöin projektin lähdekoodi käännetään ja pelin tarvitsema sisältö muunnetaan alustalle sopivaan muotoon, jonka jälkeen ohjelma suoritetaan. Tällä tavoin testaaminen on nopeampaa, sillä se ei vaadi koko pelin täydellistä paketointia. [28][29]

Käyttäjille levitettävää ohjelman versiota varten koko projekti lopulta käännetään ja paketoidaan valmiiksi paketiksi valitulle alustalle. Paketoinnin voi tehdä graafisen käyttöliittymän kautta valitsemalla valikosta *File* → *Package Project* ja valitsemalla kohdejärjestelmän. Saman valikon kautta valitaan myös käännösasetukset valikosta *File* → *Package Project* → *Build Configuration*. Vaihtoehtoja ovat esimerkiksi *Development* ja *Shipping*. Yksityiskohtaisemmat asetukset löytyvät valikosta *Packaging Settings*.... Valikot ovat nähtävillä kuvassa 5.



Kuva 5: ElementalDemo-projektin paketointiasetukset Microsoft Windows 7:llä (UE4:n versio 4.12.5).

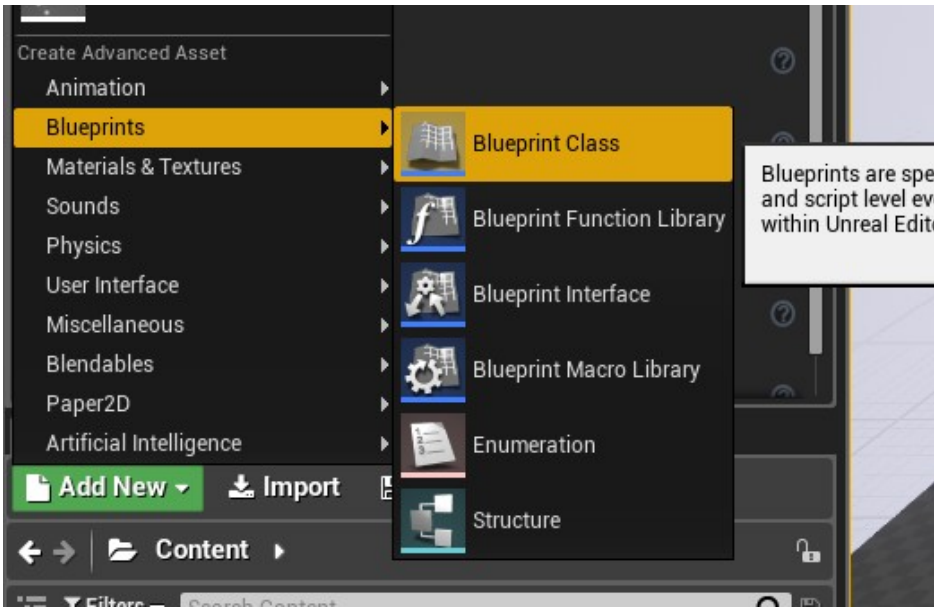
Paketoinnin jälkeen kaikki ohjelman suoritukseen vaadittava sisältö ja kentät on koostettu valmiiksi kokonaisuudeksi ohjelman itsenäistä suoritusta varten. On hyvä huomata, että vasta luodulla pelillä ei ole valmiiksi valittuna oletuskenttää, joten vasta-paketoitua peliä ajettaessa saattaa eteensä saada vain mustan ruudun. Oletuskentän voi valita valikon *Edit → Project settings… → Maps & Modes* kautta. [30]

5 VISUAALINEN OHJELMOINTI BLUEPRINTEILLÄ

Blueprintit ovat Unreal Engine 4:n lähestymistapa visuaaliseen ohjelmointiin, jolloin kehittäjä voi luoda suoritettavaa logiikkaa graafisten työkalujen avulla tekstipohjaisen ohjelmoinnin sijaan [31]. Visuaalinen ohjelmointi tapahtuu yhdistelemällä keskenään erilaisia solmuja (*node*), tapahtumia (*event*), funktioita sekä muuttujia. Käyttömahdollisuuksia on monia kuten esimerkiksi pelaajan tai aktoreiden käyttäytymisen ohjaus, pelivalikoiden toiminta sekä pelin etenemisen ja välianimaatioiden kontrollointi. Tässä luvussa käydään läpi erilaisia blueprint-tyyppejä ja niiden käyttötarkoituksia sekä lopuksi blueprinteillä ohjelmointia käyttäen Blueprint Editor -työkalua.

5.1 Blueprintien lisääminen

Blueprintejä voidaan luoda käyttämällä Content Browserin *Add New* -painiketta [32]. Painikkeesta avautuvan valikon käyttöä havainnollistaa kuva 6.



Kuva 6: Content Browser ja Add New -valikko, jonka kautta voidaan valita useita Blueprint-tyyppejä.

Lisäämiseen voi liittyä myös jonkinlainen ohjattu prosessi. Esimerkiksi tyyppiä Blueprint Class luotaessa aukeaa luonnin yhteydessä dialogi, jossa valitaan luokalle haluttu kantaluokka. [33]

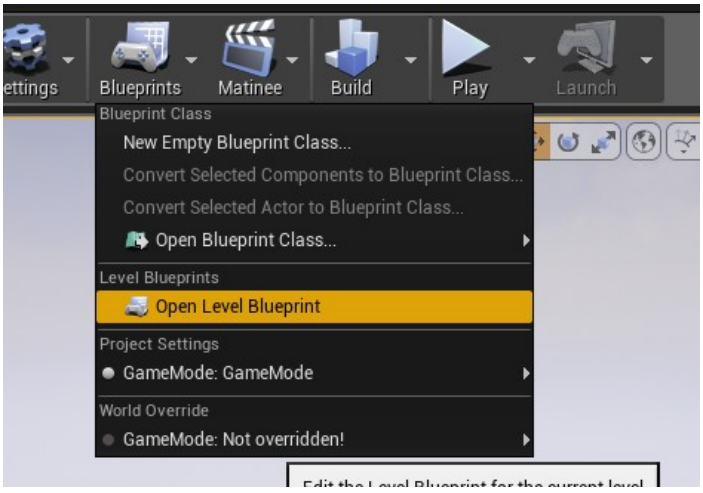
5.2 Graafit ja blueprint-tyypit

Blueprintejä on useaa eri tyyppiä, joista tavallisimmin käytetyt ovat *Level Blueprint* ja *Blueprint Class*. Muita tyyppejä ovat *Data-Only Blueprint*, *Blueprint Interface* sekä *Blueprint Macro Library*. Useimmat blueprint-tyypit sisältävät ainakin yhden muokattavan *graafin* (engl. *graph*). Graafia muokkaamalla blueprintiin ohjelmoidaan toiminnallisuutta. Suorituksen kulkua ohjataan kytkemällä muun muassa funktiokutsuista ja tapahtumista muodostuvia solmuja toisiinsa kaarilla. [31][34][35]

Myös graafityyppejä on erilaisia ja blueprint-tyypit voivat sisältää kulloisellekin blueprint-tyypille ominaisia blueprint-graafityyppejä (*Blueprint Graph Types*). Hyvä esimerkki on niin kutsuttu *Construction Script.* Construction Script on ainoastaan tyypissä Blueprint Class käytettävä graafityyppi, joka suoritetaan ainoastaan Blueprint Classin luontivaiheessa. Se siis muistuttaa oliokielien rakentaja-funktiota. [36]

5.2.1 Level Blueprint

Level Blueprint on blueprint-tyyppi, jota käytetään ohjaamaan pelin kentän ja siihen kuuluvien objektien tai animaatioiden käyttäytymistä laajemmassa kokonaisuudessa [31][34]. Jokaisella kentällä on oma blueprintinsä. Level Blueprintin muokkaus aloitetaan valitsemalla *Open Level Blueprint* työkalupalkin *Blueprints*-valikon kautta. Valikko on nähtävillä kuvassa 7.



Kuva 7: Valikko Level Blueprintin avaamiseen

Level Blueprintiä käytetään käsittelemään kentänlaajuisia tapahtumia tai kenttään kuuluviin aktoreihin (*actor*) liittyviä tapahtumia. Tapahtumia käyttämällä pelissä voidaan laukaista toimintoja funktiokutsujen avulla [37]. Aktori on mikä tahansa liikuteltava objekti, joka voidaan sijoittaa pelin kenttään [24].

5.2.2 Muut luokat ja tyypit

Muut mainitut blueprint-tyypit ovat nähtävissä myös kuvan 6 valikossa. Muut tyypit ja niiden käyttötarkoitukset ovat:

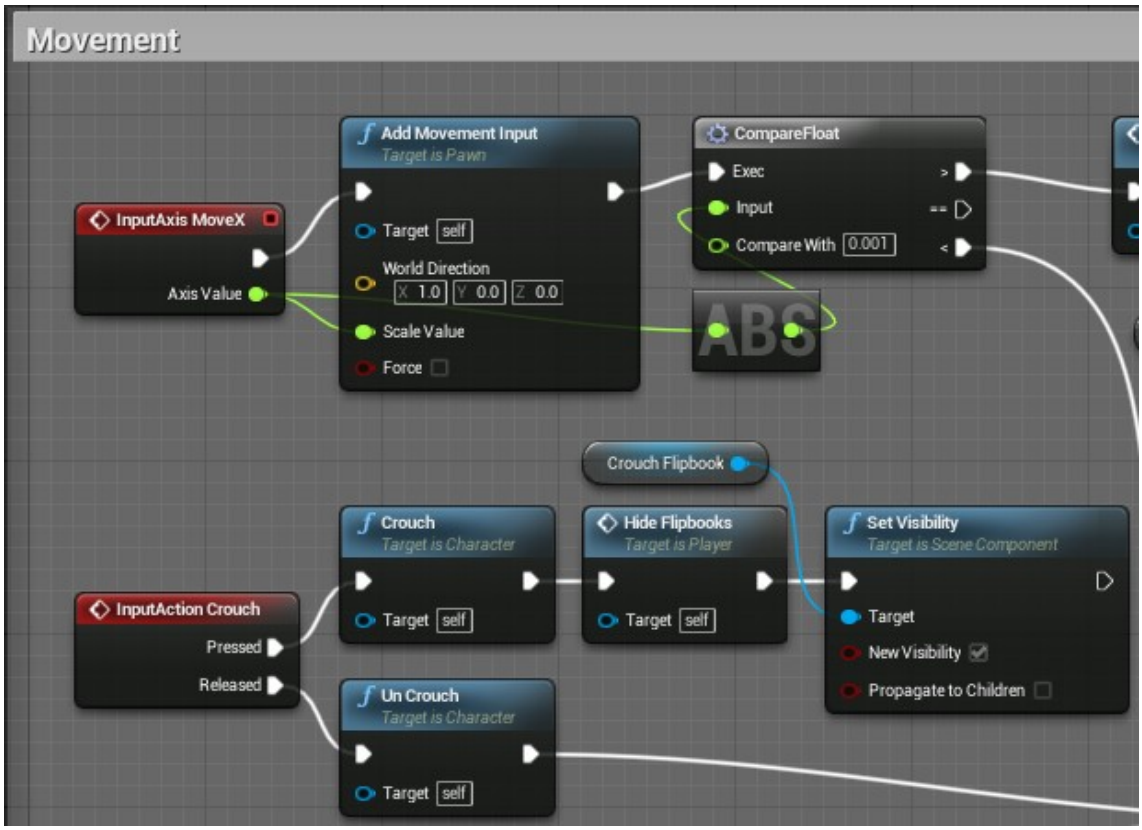
- *Blueprint Class* soveltuu hyvin interaktiivisten objektien, kuten esimerkiksi ovien tai painikkeiden, luontiin [31]. Yksi tyypille ominainen graafityyppi on *Construction Script*. Construction Script on nimensä mukaisesti blueprint-graafi, joka ajetaan Blueprint Class -tyypin luontivaiheessa eli esimerkiksi luotaessa objektia pelin kenttään pelin alkaessa. [36][34]
- *Data-Only Blueprint* on Blueprint Class, jonka tarkoituksena on tarjota kevyt tyyppi kantaluokasta muokatun version luomiseen. Se voi sisältää täten ainoastaan kantaluokkansa koodin, muuttujat ja komponentit. [33][34]
- *Blueprint Interface* sisältää kokoelman funktiorajapintoja joka voidaan liittää muihin blueprinteihin, jolloin kyseiset blueprintit sisältävät tämän jälkeen aina nämä funktiot. Itse toteutus on tehtävä näihin blueprinteihin. [34]
- *Blueprint Macro Library* on säiliö makroja tai itsenäisiä graafikokonaisuuksia varten. Sen tarkoituksena on yksinkertaistaa toistuvasti käytettyjen blueprint-graafien käyttöä. [34]

5.3 Blueprint Editor

Blueprint Editor on työkalu blueprintien muokkaamiseen [13]. Halutun blueprintin saa avattua muokattavaksi esimerkiksi Content Browser -ikkunan kautta kaksoisklikkaamalla haluttua kohdetta. Poikkeuksen muodostaa Level Blueprint, jonka muokkaaminen aloitetaan työkalupalkin *Blueprints*-valikon kautta.

Blueprint-graafien muokkaus tapahtuu Blueprint Editoriin kuuluvan *Graph Editor* -työkalun avulla. Blueprintiin kuuluvien muuttujien ja graafien muokkaus on käytännössä blueprintin, ja siitä mahdollisesti luodun aktorin, ohjelmointia. Työkalun ulkoasua havainnollistaa kuva 8. Kuva koko Blueprint Editorin käyttöliittymästä on nähtävissä liitteessä 2. [31][33][38]

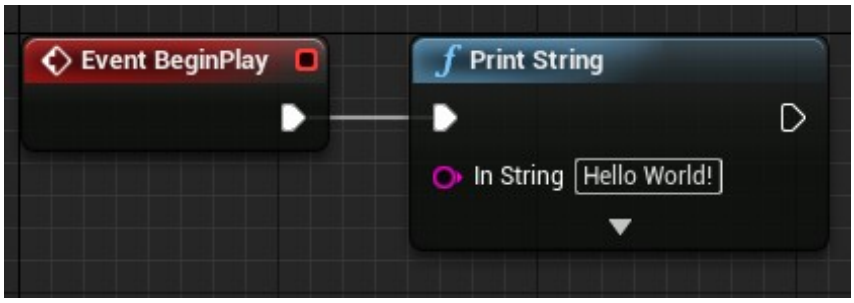
Muokattavissa olevat ominaisuudet riippuvat blueprintin tyypistä. Esimerkiksi yleisesti käytetystä Blueprint Class -tyypistä voidaan luoda uusia aktoreita peliin ja muokata tapahtumagraafin avulla sen sisältämiä parametreja sekä reagointia erilaisiin tapahtumiin. Toisaalta tyypit *Blueprint Macro* ja *Blueprint Function* sisältävät ainoastaan graafin, ja ne ovat tarkoitettu kutsuttavaksi ainoastaan muiden blueprint-graafien sisältä. Graafia muokatessa uusien solmujen luontiin voi käyttää oikealla hiirenpainikkeella avattavaa valikkoa. [31][33][38]



Kuva 8: Esimerkkinä Graph Editor ja itse toteutettu Blueprintin tapahtumagraafi, joka säätelee pelaajan liikkumista

5.4 Funktiokutsut ja tapahtumat

Yleisin blueprintien sisältämä graafityyppi on tapahtumagraafi (*Event Graph)*. Sen tarkoituksena on ottaa vastaan tapahtumia (*events*) ja funktiokutsuja ja reagoida niihin kehittäjän määrittelemällä tavalla. Näitä tapahtumia käytetään usein interaktiivisen toiminnallisuuden lisäämiseen. Kuva 9 näyttää kuinka tulostetaan pelin alkaessa merkkijono ”Hello World!” kytkemällä `BeginPlay`-tapahtumaan funktio `Print String`. [38]



Kuva 9: BeginPlay-tapahtumaan kytketty funktio, joka tulostaa merkkijonon "Hello World!"

Liitteen 2 esittämä kuva Blueprint Editorista näyttää myös tapahtumagraafin sekä siitä löytyvät aktorin oletustapahtumat `BeginPlay` ja `Tick`. Ne ovat pelin kannalta keskeisiä tapahtumia, joita kutsutaan jokaisella ruudun piirtokerralla [39].

6 UE4:N OHJELMOINTI C++:LLA

Visuaalisen ohjelmoinnin lisäksi UE4:n kanssa voidaan käyttää ohjelmointiin myös C++:aa. Kehittäjän ei ole kuitenkaan pakko valita pelkästään toista. On kyllä mahdollista luoda kokonainen peli kirjoittamatta riviäkään tekstimuotoista lähdekoodia tai vastaavasti käyttäen pelkästään C++:aa, mutta dokumentaation mukaan paras tapa pelimoottorin hyödyntämiseen on kuitenkin käyttää molempia tapoja yhdessä. Tällöin tekniikoiden ominaisuudet vahvuuksineen täydentävät toisiaan. [40][41][42]

Tämä luku keskittyy esittelemään C++-ohjelmointia UE4:n kanssa. Varsinaisia C++-ohjelmoinnin perusteita ei luvussa kuitenkaan käydä läpi vaan perusoletus on, että lukijalla on riittävän hyvä yleiskäsitys ohjelmoinnista ymmärtääkseen annetut esimerkkitapaukset. Luvussa käsitellään ensin mahdollisuuksia erilaisten kehitysympäristöjen käyttöön sekä tarkastellaan UE4:n tarjoamia ohjelmointityökaluja. Lopuksi käsitellään itse ohjelmoinnin aloittamista luokan luomisella ja tekemällä siihen yksinkertaisia muutoksia.

6.1 Ohjelmointityökalut ja -ympäristöt

Unreal Engine 4 ei tarjoa valmista kehitysympäristöä C++-lähdekoodin kirjoittamiseen, vaan kehittäjä valitsee työkalun itse [40]. Pelimoottori on kuitenkin alun perin suunniteltu kehitettäväksi Microsoft Visual Studio 2015 -kehitysympäristöllä ja integroituu siihen suoraan [43]. Dokumentaatio käyttää myös ohjeissaan ja esimerkeissään pääasiassa Visual Studiota. Dokumentaatiosta löytyy kuitenkin joiltain sivuilta ohjeita myös Mac OS X -käyttöjärjestelmällä ajettavalle Xcode -kehitysympäristölle [44].

On myös mahdollista käyttää muuta kehitysympäristöä ja Unreal Editoria rinnakkain ongelmitta. Kehitysympäristössä muokattu lähdekoodi käännetään jaetuiksi kirjastoiksi (*shared library*). Muutosten ja uudelleenkääntämisen jälkeen kyseiset kirjastot päivittyvät, jolloin Unreal Editor tunnistaa päivittyneet tiedostot ja ottaa ne käyttöön automaattisesti. Ominaisuutta kutsutaan dokumentaatiossa nimellä *Hot reloading* [40]. Vaihtoehtoisesti kääntäminen voidaan suorittaa Unreal Editorissa päätyökalupalkin *Compile*-painikkeen avulla. Tämän vuoksi varsinaisen lähdekoodin muokkaamisen voi periaatteessa tehdä millä tahansa työkalulla, eikä esteitä esimerkiksi Linux-käyttöjärjestelmässä kehittämiselle ei ole. [40]

Varsinaista C++-lähdekoodin kirjoittamista on pyritty helpottamaan erilaisilla UE4:n tarjoamilla työkaluilla ja ominaisuuksilla. Esimerkiksi luokkien ohjattuun luomiseen on

graafinen työkalu *Class Wizard*, jolla pohjan luominen luokille on yksinkertaista. Ohjelmointia yksinkertaistavat myös erilaiset valmiit C++-makrot. [40]

Lähdekoodin kirjoittamista varten dokumentaatio sisältää sivun, jossa määritellään Epic Gamesin käyttämät, ja pelimoottorin lähdekoodissa käytetyt, käytännöt ja standardit. Ohjeita ei ole välttämätöntä noudattaa omassa lähdekoodissaan, mutta mahdollisesti yhteisölle julkistettavaksi tarkoitetun koodin kanssa se edesauttaa yhdenmukaisuutta ja selkeyttä sekä yhteensopivuutta eri alustojen välillä. [45]

6.2 Erilaisten kääntämistapojen kokeilua Linux-ympäristössä

Dokumentaatiosta ei löydy varsinaisia ohjeita muille kehitysympäristöille kuin aiemmin mainitulle ja virallisesti tuetulle Visual Studiolle sekä Xcodelle. Luodun C++-projektin hakemistosta löytyy kuitenkin tiedostoja, jotka näyttävät määrittelytiedostoilta erilaisia kehitysympäristöjä varten. Näitä ovat esimerkkiprojektin tapauksessa *TestProjectCpp.kdev4 (Kdevelop* [46]*), TestProjectCpp.pro (Qmake* ja *Qt Creator* [47]) sekä *Makefile* (yleisesti käytetylle *make*-työkalulle) ja *CMakeLists.txt (cmake [48])*.

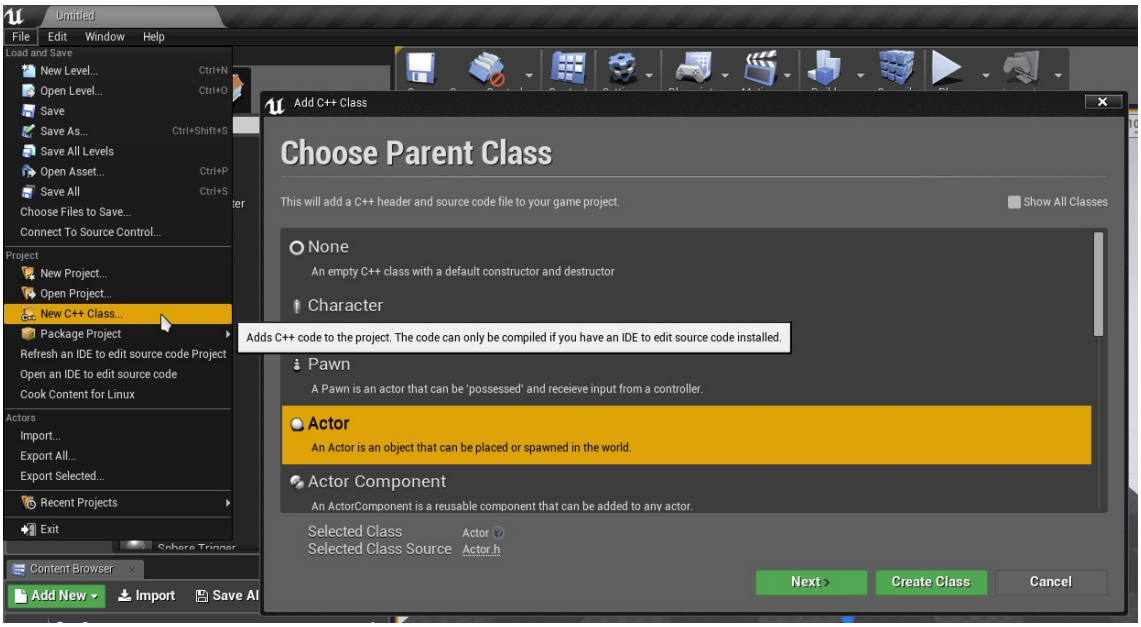
Qmaken projektitiedosto ei kuitenkaan vaikuttanut toimivan sellaisenaan Qt Creatorin kanssa oikein, sillä kääntäminen ei mennyt virheittä läpi ja näytti lisäksi kääntävän tarpeettoman oloisesti myös itse pelimoottoriin liittyviä lähdekooditiedostoja.

Hakemistosta löytyvä *Makefile* ei myöskään toiminut odotetunlaisesti. Lupaavannäköisestä *Makefilen* sisällöstä huolimatta komennot ”*make*” tai ”*make TestProjectCpp*” eivät kääntäneet projektiin kuuluvia lähdekooditiedostoja vaan vaikutti pikemminkin samaan tapaan aloittavan myös UE4-pelimoottorin komponenttien kääntämisen.

Kokeillut keinot eivät siis tuottaneet haluttua tulosta tai tarvittavaa kirjastotiedostoa. Näyttäisikin olevan yksinkertaisinta käyttää valitsemaansa kehitysympäristöä vain lähdekoodin muokkaamiseen ja suorittaa varsinainen kääntäminen Unreal Editorin *Compile*-toiminnolla. Unreal Editorilla kääntäminen tuottaa oikeanlaisen tuloksen ja esimerkkiprojektin alihakemistoon *Binaries/Linux/* ilmestyy kääntämisessä tuotetut .so-päätteiset kirjastotiedostot. Ohjelman tulosteiden perusteella kääntämisessä hyödynnetään joitakin komentosarjatiedostoja sekä *UnrealBuildTool*-työkalua mistä myös dokumentaatio on maininnut [49]. Kääntämisprosessi onkin siis luultavasti oletettua monimutkaisempi eikä siihen tässä työssä tämän tarkemmin syvennytä.

6.3 Uuden luokan luominen

Uuden C++-luokan lisääminen tapahtuu yksinkertaisimmillaan Unreal Editorin työkalun *C++ Class Wizard* avulla. Työkalun löytää navigoimalla päävalikosta *File → New C++ Class...* tai käyttämällä kuvassa 6 nähtävää Content Browserin *Add New* -valikkoa. Aukeavasta dialogista valitaan uudelle luokalle haluttu kantaluokka. Kuvassa 10 näkyy *C++ Class Wizard*, jossa kantaluokaksi on valittu *Actor*-luokka. [50][40][51]



Kuva 10: Valikko uuden C++-luokan luomiseen ja perittävän luokan valinta valikosta avautuneessa dialogissa.

Uuden luokan lähdekooditiedostot sijoitetaan oletuksena projektin Sources-alihakemistoon, mutta sijainti on vapaasti päätettävissä luokan määrittämisen seuraavassa vaiheessa. Oletuspolkua käytettäessä luokan `MyTestActor` lisäämisen jälkeen Sources-alihakemiston rakenne näyttää seuraavalta:

```
Source/
|-- TestProjectCpp
|    |-- MyTestActor.cpp
|    |-- MyTestActor.h
|    |-- TestProjectCpp.Build.cs
|    |-- TestProjectCpp.cpp
|    |-- TestProjectCppGameMode.cpp
|    |-- TestProjectCppGameMode.h
|    `-- TestProjectCpp.h
|-- TestProjectCppEditor.Target.cs
`-- TestProjectCpp.Target.cs
```

6.4 Luokan lähdekoodi

Valmiiksi generoitu luokka sisältää vain välttämättömät funktiomääritykset eli tässä tapauksessa `BeginPlay()` ja `Tick(float DeltaSeconds)`, jotka ylikuormittavat valitun kantaluokan `Actor` jäsenfunktiot. Ohjelmat 2 ja 3 näyttävät tämän automaattisesti luodun lähdekoodin kokonaisuudessaan. Funktiota `BeginPlay()` kutsutaan, kun kyseinen objekti on pelin käynnistyttyä luotu maailmaan ja valmiina toimimaan. Funktiota `Tick(float DeltaSeconds)` kutsutaan tämän jälkeen jokaisella ruudun piirtokerralla parametrilla, joka kertoo edellisen ruudun piirrosta kuluneen ajan sekunteina. [31][40]

Luokan `TestActor` otsikkotiedosto *TestActorCpp.h* nähdään ohjelmassa 2. Siinä nähdään luokan `TestActor` rajapinta ja kantaluokan `Actor` jäsenfunktioiden ylikuormitus. Rivin 6 makro `UCLASS()` sekä rivin 9 makro `GENERATED_BODY()` ovat pakollisia luokan määritykseen Unreal Enginea varten [31][51].

```
1.  // Fill out your copyright notice in the Description page of Project
    Settings.
2.  #pragma once
3.  #include "GameFramework/Actor.h"
4.  #include "TestActor.generated.h"
5.
6.  UCLASS()
7.  class TESTPROJECTCPP_API ATestActor : public AActor
8.  {
9.          GENERATED_BODY()
10.
11. public:
12.         // Sets default values for this actor's properties
13.         ATestActor();
14.         // Called when the game starts or when spawned
15.         virtual void BeginPlay() override;
16.
17.         // Called every frame
18.         virtual void Tick( float DeltaSeconds ) override;
19. };
```

Ohjelma 2: Ohjelmassa 3 esitetyn tiedoston TestActorCpp.cpp otsikkotiedosto TestActorCpp.h.

Itse lähdekooditiedoston sisältö on esitetty ohjelmassa 3.

```
1.  // Fill out your copyright notice in the Description page of Project
    Settings.
2.  #include "TestProjectCpp.h"
3.  #include "TestActor.h"
4.
5.  // Sets default values
6.  ATestActor::ATestActor()
7.  {
8.          // Set this actor to call Tick() every frame.  You can turn this off
    to improve performance if you don't need it.
9.          PrimaryActorTick.bCanEverTick = true;
10. }
11.
12. // Called when the game starts or when spawned
13. void ATestActor::BeginPlay()
14. {
15.         Super::BeginPlay();
16. }
17.
18. // Called every frame
19. void ATestActor::Tick( float DeltaTime )
20. {
21.         Super::Tick( DeltaTime );
22. }
23.
```

Ohjelma 3: Unreal Editorin tiedostoon TestActorCpp.cpp generoima lähdekoodi luokasta `TestActor`

Esimerkissä käytetty `AActor`-luokka on yksi neljästä pääluokasta, joiden pohjalta pystytään luomaan suurin osa pelissä käytetyistä luokista. Kolme muuta pääluokkaa ovat `UObject`, `UActor`, `UActorComponent`, ja `UStruct`. Valmiiden luokkien käyttö ei ole välttämätöntä, mutta ilman niitä luokka ei tule hyödyntämään pelimoottorin ominaisuuksia. [40]

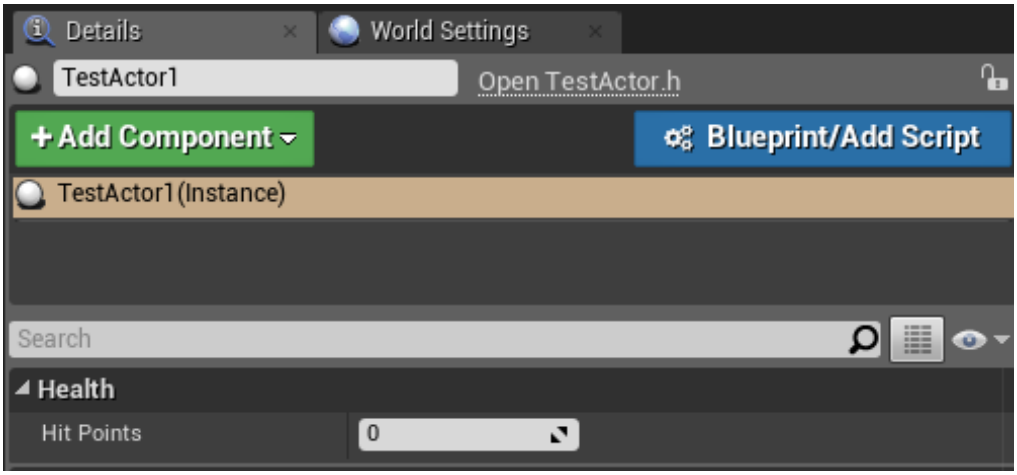
6.5 Luokan laajentaminen

Luokan määrittely vaatii makrojen `UCLASS()` ja `GENERATED_BODY()` käyttöä, jotta luokka saadaan näkyviin Unreal Editorin käyttöliittymään. Samaan tapaan luokkaan lisättävät attribuutit vaativat makron `UPROPERTY()` käyttöä. Makron avulla voidaan määrittää esimerkiksi luku- ja kirjoitusoikeuksia blueprintejä varten tai kategoria, jonka alla kyseinen attribuutti Unreal Editorissa näytetään. [40]

Esimerkkiluokan `TestActor` tapauksessa luokalle voitaisiin lisätä esimerkiksi attribuutti `HitPoints`. Tätä varten ohjelmassa 2 näkyvään otsikkotiedostoon lisätään rivin 18 jälkeen seuraavat määritykset:

```
19.  private:
20.      UPROPERTY(EditAnywhere, Category="Health")
21.      int32 HitPoints;
```

Kääntämisen jälkeen attribuutti ilmestyy luokan `TestActor` ominaisuuksiin. Tämän voimme nähdä kuvassa 11.



Kuva 11: Luokan `TestActor` instanssi, jonka ominaisuuksissa nähdään attribuutti `HitPoints` kategoriassa Health

Tämä mahdollistaa C++-lähdekoodina kirjoitettujen luokkien käytön graafisilla työkaluilla. Myös luokan jäsenfunktioiden paljastamisessa käytetään samaa ideaa, mutta funktioiden kanssa käytettävä makro on `UFUNCTION()`. Jos esimerkiksi halutaan antaa

mahdollisuus kutsua jäsenfunktiota myös blueprintien kautta, on käytettävä makroa parametreilla `UFUNCTION(BlueprintCallabe, Category="Health").` [40]

7 PROFILOINTI

Tässä luvussa perehdytään profilointiin ja sen merkitykseen sekä esitellään Unreal Engineen kuuluvia profilointityökaluja. Tämän jälkeen keskitytään yksityiskohtaisemmin CPU- ja GPU-profilointiin. Lopuksi havainnollistetaan tämän työn kannalta keskeisimmän profilointityökalun, *Profilerin*, käyttöä.

Unreal Engine sisältää lukuisia ominaisuuksia, joista jokaisella on omanlaisensa vaikutus suorituskykyyn. Suorituskyvyn optimoimiseksi ja pullonkaulojen löytämiseksi täytyy ensin löytää kohteet, jotka ovat merkittäviä laskentatehon kannalta. Tässä luvussa käsiteltävät UE:n profilointityökalut on tehty juuri tämänlaista suorituskyvyn profilointia varten. [7][52]

7.1 Mitä on profilointi?

Nykyaikaisissa laitteissa on monia rinnakkain ajautuvia yksiköitä, joita myös pelit hyödyntävät monella tasolla. On oleellista tunnistaa tekijät, jotka aiheuttavat suoritukseen pullonkauloja tai ovat kohdelaitteistolle tarpeettoman raskaita. Väärien asioiden optimointi ei hyödytä suorituskykyä halutulla tavalla minkä lisäksi turha optimointi voi johtaa uusiin virheisiin tai suorituskykyongelmiin muilla osa-alueilla. Yhden osa-alueen optimoinnin jälkeen profilointi kannattaa suorittaa uudestaan, sillä se voi paljastaa uusia pullonkauloja, jotka eivät aiemmin olleet nähtävissä. [7][52]

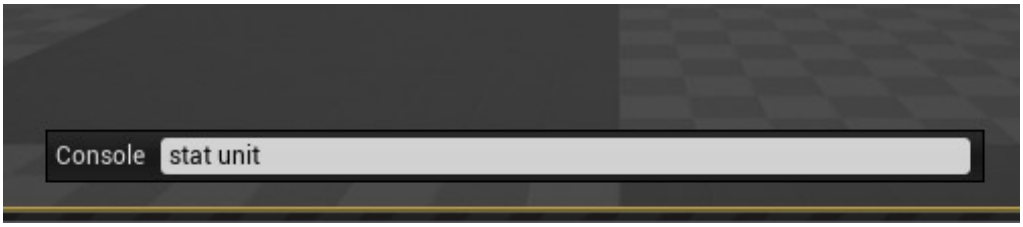
Tarkassa profiloinnissa mittayksikkönä kannattaa käyttää millisekunteja. Näytölle sekunnin aikana piirrettävien kuvien määrää kuvaava yksikkö *frames per second (*lyhyesti *FPS)* ei ole profiloinnin kannalta yhtä käytännöllinen, vaikka muuntaminen yksiköiden välillä onkin periaatteessa yksinkertaista. Tämä johtuu siitä, että profiloinnissa mitataan yksittäisten ominaisuuksien laskentaan käytettyä aikaa ja ajan yksikkönä käytetään tässä tapauksessa millisekunteja. Lisäksi optimoinnin tuomat säästöt laskenta-ajoissa eivät näy samassa suhteessa suoraan lopullisessa FPS-luvussa. [52]

7.2 Konsolin ja profilointityökalujen käyttö

Unreal Engine 4 sisältää useita erilaisia työkaluja suorituskyvyn profilointia varten. Niiden avulla pystytään analysoimaan sovelluksen suoriutumista yksityiskohtaisesti sekä löytämään suorituksesta mahdolliset pullonkaulat ja raskaat operaatiot. Suorituskykyä analysoitaessa kannattaa ensimmäiseksi selvittää rajoittaako suorituskykyä keskusprosessori (*CPU*) vai näytönohjain (*GPU*). Dokumentaation antamien yleisten ohjeiden

mukaan profilointi kannattaa suorittaa mahdollisimman toistettavasti ja irrallaan mahdollisista häiriötekijöistä. Profiloinnin tarkkuutta voidaan arvioida toistamalla suoritus useita kertoja ja vertailemalla saatuja arvoja. [7][11]

Yksi UE4:n profilointityökaluista on nimeltään *Profiler*. Profiler-työkalun lisäksi voidaan käyttää myös esimerkiksi niin kutsuttuja *stat*-komentoja, joista yksi on *stat unit*. Konsoli (engl. *console*) voidaan avata Unreal Editorissa tai itse pelissä, ja konsolikomentojen syöttämisen voi aloittaa konsolin avaamisen jälkeen. Konsolin avaamiseen käytettävä näppäinyhdistelmä voidaan asettaa navigoimalla päävalikon kautta *Edit…* → *Editor Preferences…* → *Keyboard shortcuts*. Toiminnon nimi on *Open Console Command Box*. Avattu konsoli on nähtävillä kuvassa 12. [7]



Kuva 12: Konsoliin on kirjoitettu komento ”stat unit”. Konsoli aukeaa näyttöalueen alareunaan.

Stat-komentoja ja -näkymiä on useanlaisia ja ne tarkoitettu nimenomaan suorituskyvyn profilointia varten. UE4:n dokumentaatio sisältää komennoista kattavan listan kuvauksineen [53]. Komennon syöttämisen jälkeen näytöllä on nähtävissä arvoja liittyen erilaisten toimintojen käyttämästä laskenta-ajasta. Arvojen tulostaminen lokitiedostoon on myös mahdollista. Esimerkki näyttöalueen oikeaan reunaan tulostuvista *stat unit* -arvoista nähdään kuvassa 13. [7][53]

Kuva 13: Stat unit *-arvot näyttöalueen oikeassa reunassa.*

Arvoista ”*Frame*” tarkoittaa yhden kuvan piirtämiseen kulunutta aikaa. ”*Game*” tarkoittaa CPU:n suorittaman pelisäikeen vaatimaa aikaa (*CPU Game Thread*), ”*Draw*” CPU:n suorittaman piirtosäikeen aikaa (*CPU Render Thread*) ja ”*GPU*” nimensä mukaisesti näytönohjaimen käyttämää aikaa. Näiden lukujen pohjalta voidaan arvioida pullonkaulana toimivaa suoritusyksikköä senhetkisessä tilanteessa. Esimerkkikuvan tapauksessa pullonkaulan muodostaa *Game*-säie. [7][52]

7.3 CPU-profilointi

Jos pullonkaulan muodostava yksikkö ohjelman suorituskyvyssä on CPU, on UE4:n dokumentaation mukaan syy todennäköisesti liian suuressa määrässä piirtokutsuja. Piirtokutsujen määrää voidaan vähentää esimerkiksi yhdistämällä useita erillisiä piirrettäviä objekteja yhdeksi. Objektien piirtämiseen liittyvä CPU:n kuormitus koostuu todellisuudessa useasta tekijästä. Tämä kohta käsittelee keinoja tunnistaa eniten CPU:n laskentatehoa vaativia osa-alueita ja sitä kautta merkittävimpien optimointikohteiden löytämistä. [54]

Stat unit -ominaisuuden tapaan CPU-profilointiin voidaan käyttää muitakin samankaltaisia stat-komentoja. Esimerkiksi komentoa ”stat scenerendering” käytetään esittämään profilointitietoa CPU:n piirtosäikeestä. Ominaisuuden käyttöä havainnollistaa kuva 14. Kuvassa näkyvässä taulukossa on eritelty muun muassa valoihin ja varjoihin liittyvät laskenta-ajat. [54]

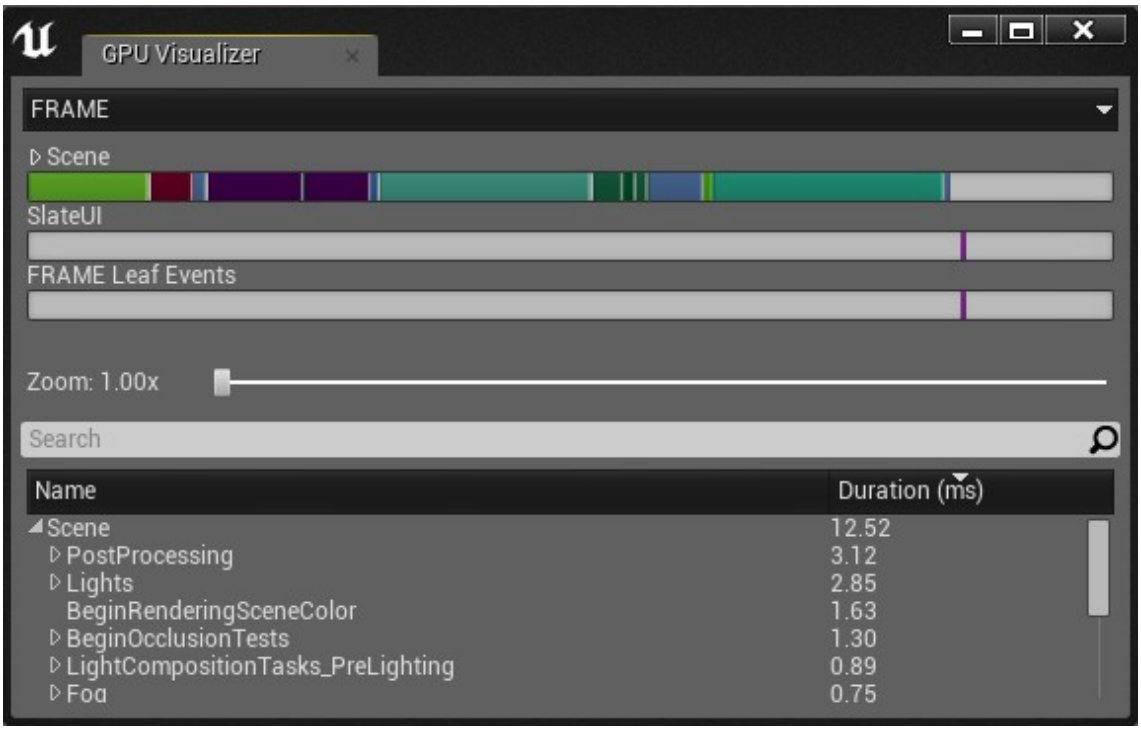
Kuva 14: stat scenerendering *-ominaisuuden näyttämät profilointitiedot.* stat unit *-ominaisuuden tapaan tiedot näkyvät näyttöalueella.*

Komento ”stat game” esittää vastaavanlaisia profilointitietoja *Game*-säikeestä. Ominaisuutta havainnollistaa kuva 15, jossa nähdään esimerkiksi aktorien luontiin (*Spawn Actor Time*) ja blueprintien prosessointiin (*Blueprint Time*) käytetty aika vastaavanlaisessa taulukossa eriteltynä. Taulukosta voi saada jo hyvän kuvan sopivista optimointikohteista. Tarvittaessa komento ”stat dumpframe” mahdollistaa lisäksi ajankäytön yksityiskohtaisemman tarkastelun funktiotasolla. [54]

Kuva 15: Aiempia stat-komentoja vastaavat stat game -profilointitiedot

7.4 GPU-profilointi

Näytönohjain käyttää useita rinnakkaisia yksiköitä laskennassa ja on yleistä, että eri yksiköt ovat pullonkaulana eri osissa piirrettävän kuvan laskentaa. GPU:n profiloinnissa voidaan hyödyntää komentoa ”ProfileGPU”, jonka avulla voidaan analysoida GPU:n eri osa-alueisiin käyttämää laskenta-aikaa. Työkalun ulkonäköä havainnollistaa kuva 16. Työkalua voidaan käyttää myös ilman graafista käyttöliittymää. [55]



Kuva 16: Kuva GPU Visualizer -työkalusta. Esimerkissä nähdään rivit Duration-sarakkeen mukaan laskevasti järjestettynä, joten eniten aikaa vievät osa-alueet ovat ylimpänä.

Dokumentaation mukaan erilaiset näytönohjaimen optimoinnit voivat aiheuttaa epätarkkuuksia profilointiin. Joillain ajureilla voi esimerkiksi olla tapana suorittaa optimointia vasta joidenkin sekuntien jälkeen jonkin varjostimen käytöstä. Tämän vuoksi profiloinnin arvoja tulkitessa kannattaa säilyttää sopiva kriittisyys ja varmuuden vuoksi kokeilla profilointia uudestaan esimerkiksi pidemmän odotusajan jälkeen. [55]

Profiloinnin tukena voi olla hyödyllistä testata peliä myös tarvittaessa eri resoluutioilla. Tämä auttaa havaitsemaan pikselien määrään liittyviä rajoitteita, kun laskentavaatimukset kasvavat resoluution kasvaessa. Tällaisissa tilanteissa pullonkaulat johtuvat usein näytönohjaimen laskentayksikön tai muistiväylän riittämättömyydestä. [55]

7.5 Profiler-työkalun käyttö

Profiler on suorituskyvyn analysointiin tehty ja aiemmissa kohdissa läpikäytyjä työkaluja monipuolisempi työkalu. Työkalua voidaan käyttää kahdella eri tavalla:

1. Dokumentaation käyttämä käsite *Live Connection* on tapa analysoida ohjelman suorituskykyä suoran yhteyden kautta tallentamatta dataa ensin tiedostoihin.
2. Aiemmin kaapatun profilointidatan lataaminen Profiler-työkalussa.

Työkalun saa auki Unreal Editorissa *Live Connection* -tilassa navigoimalla valikkopalkista *Window* → *Developer Tools* → *Session Frontend* ja valitsemalla *Profiler*-välilehden. Avaaminen onnistuu myös ilman Unreal Editoria käynnistämällä Unreal Editorin kanssa samassa hakemistossa sijaitsevan *Unreal Frontend* -työkalun suoraan. Tällöin peli täytyy käynnistää käyttäen parametria ”-messaging”. [11][56]

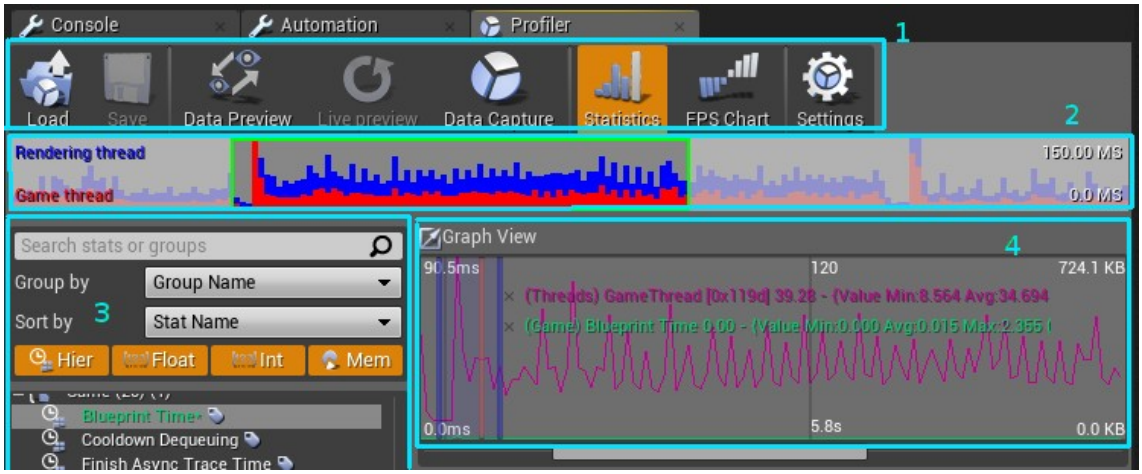
Yhteyden muodostuttua käyttöliittymästä valitaan sinne ilmestynyt uusi istunto (engl. *session)*, minkä jälkeen voidaan aloittaa profilointidatan kerääminen. *Live Connection* -tilassa tämä onnistuu Profilerin työkalupalkin *Data Preview* -toiminnolla. Tällöin kerätty tieto näkyy käyttöliittymässä suoraan, eikä sitä tallenneta tiedostoon. Toimintoa *Data Capture* sen sijaan käytetään datan keräämiseksi tiedostoon, jonka jälkeen tiedoston voi avata työkalulla tarkasteltavaksi myöhemmin käynnistämättä peliä. Pelimoottorin versiossa 4.12 istunnon valinta on käyttöliittymän vasemmassa reunassa. Valikko on nähtävillä kuvassa 17. [11][56]

Kuva 17: Istunnon valinta Unreal Frontend -työkalussa.

Kuva 18 sisältää kuvan profilointidatasta Profiler-työkalun esittämänä. Profilointidata on kaapattu käyttämällä *Data Preview* -toimintoa. Kuvan esittämä alue sijaitsee Unreal Frontend-työkalun käyttöliittymän ylemmällä puoliskolla. Kuvassa on seuraavat alueet:

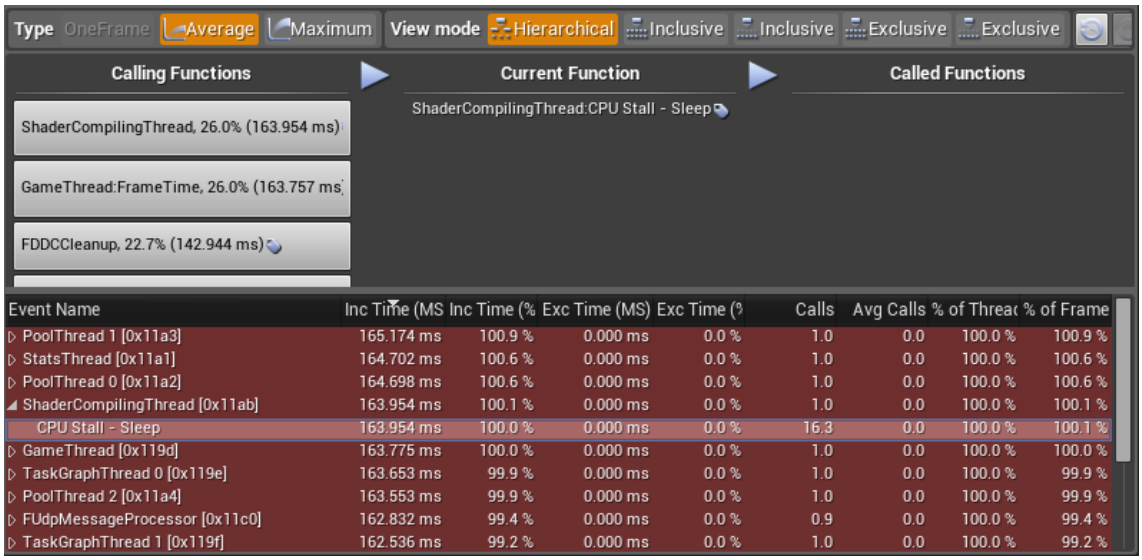
1. *Main Toolbar*, työkalupalkki.
2. *Data Graph Full* -ikkuna, jossa on karkealla tasolla nähtävillä Rendering ja Game -säikeiden profilointidata koko ajalta.
3. Vasemmassa alareunassa osittain näkyvä *Filter and Presets* -ikkuna. Tämän avulla voidaan valita *Data Graph* -ikkunassa näytettäviä kuvaajia ja tietoja.
4. *Data Graph* -ikkuna, joka näyttää valituista tiedoista generoidut kuvaajat.

Profiler-työkalun käyttöliittymä on nähtävillä kokonaisuudessaan liitteessä 3. [11]



Kuva 18: Profilointidataa havainnollistettuna ajan suhteen Profiler-työkalun esittämissä kuvaajissa.

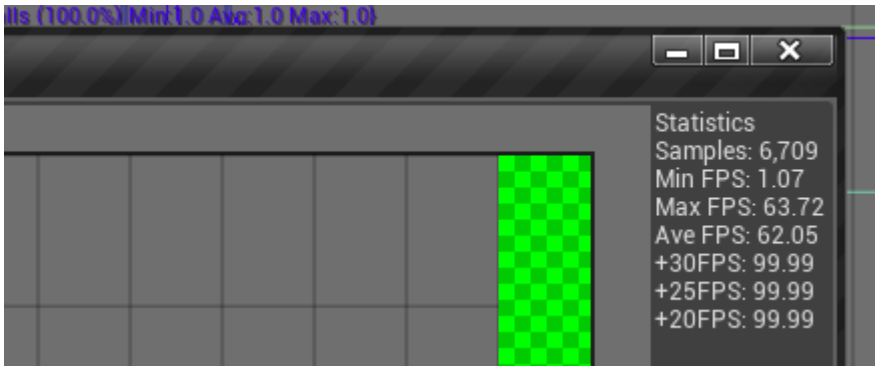
Data graph -kuvaajasta on mahdollista valita tarkasteltava alue tarkempaa tutkintaa varten. Valinnan jälkeen alueeseen liittyvät tiedot näkyvät kuvaajan alapuolella *Event Graph* -ikkunassa, joka on nähtävillä kuvassa 19. Näkymän avulla on mahdollista tarkastella aikaa vieviä kokonaisuuksia tarkemmin myös tapahtumatasolla. Myös tästä ikkunasta voidaan raahata hiiren avulla haluttuja suodattimia *Data Graph* -ikkunaan.



Kuva 19: Profiler-työkalun Event Graph -ikkuna

Lopullisissa profilointivertailuissa käytetään tässä työssä datatiedostoja, jotka on kerätty suoraan ohjelman ajosta ja ilman Unreal Editorin käynnistämistä. Tämä on myös dokumentaation suosittelema tapa. Tällöin saadaan minimoitua ylimääräiset häiriötekijät lopullisesta profilointidatasta [7]. Dokumentaation mukaan profiloinnin datatiedostot tallentuvat pelimoottorin hakemistoon "*...\UE4\Engine\Programs\UnrealFrontend\Saved\Profiling\UnrealStats\Received*", mutta profilointeja tehdessä huomattiin niiden kuitenkin tallentuneen sen sijaan paketoidun pelin alihakemistoon *ProjektinNimi\Saved\Profiling\UnrealStats*. [11][56]

Tarkkojen profilointitietojen lisäksi työssä käytetään dataa FPS-arvon keskiarvoista sekä minimiarvosta ja maksimiarvosta antamaan yleiskuvaa ohjelman suoriutumisesta. Arvot ovat nähtävissä työkalupalkin *FPS Chart* -painikkeen kautta, joka aukaisee eteen *FPS Histogram* -ikkunan. Luvut näyttävä ikkunan alue on nähtävissä kuvassa 20.



Kuva 20: FPS Histogram -ikkunan näyttämät arvot

Konsolikomentojen antaminen, ja siten myös stat-työkalujen käyttö, onnistuu myös Unreal FrontEnd -työkalun kautta. Komentojen antaminen vaatii profiloinnin tapaan

valmiiksi muodostetun yhteyden pelin kanssa, minkä jälkeen konsoli on käytettävissä kuvassa 18 näkyvän ”Console”-välilehden kautta. [56]

7.6 Palomuuriasetukset Profiler-työkalulle

Jos *live connection* -tapaa käyttäessä yhteyden muodostus työkalun ja pelin välille ei onnistu, kannattaa tarkistaa ettei palomuuri estä pelin ja työkalun välistä verkkoliikennettä. Tämä voi olla tarpeellista, vaikka profilointi suoritetaankin paikallisesti eikä verkon yli. Palomuurin tulee sallia liikennöinti *UDP*-verkkoprotokollalla *multicast*-osoitteeseen 230.0.0.1 ja porttiin 6666. Tämän voi nähdä esimerkiksi palomuurilokista ja pelin tulostamista käynnistysviesteistä, joista yksi on ”*UdpMessaging: Initializing bridge on interface 0.0.0.0:0 to multicast group 230.0.0.1:6666*”. Unreal Enginen wiki-sivuston mukaan tämä verkkoliikenne on tarpeellista Linux-versiossa myös valaistuksen laskentaa varten [57]. [56]

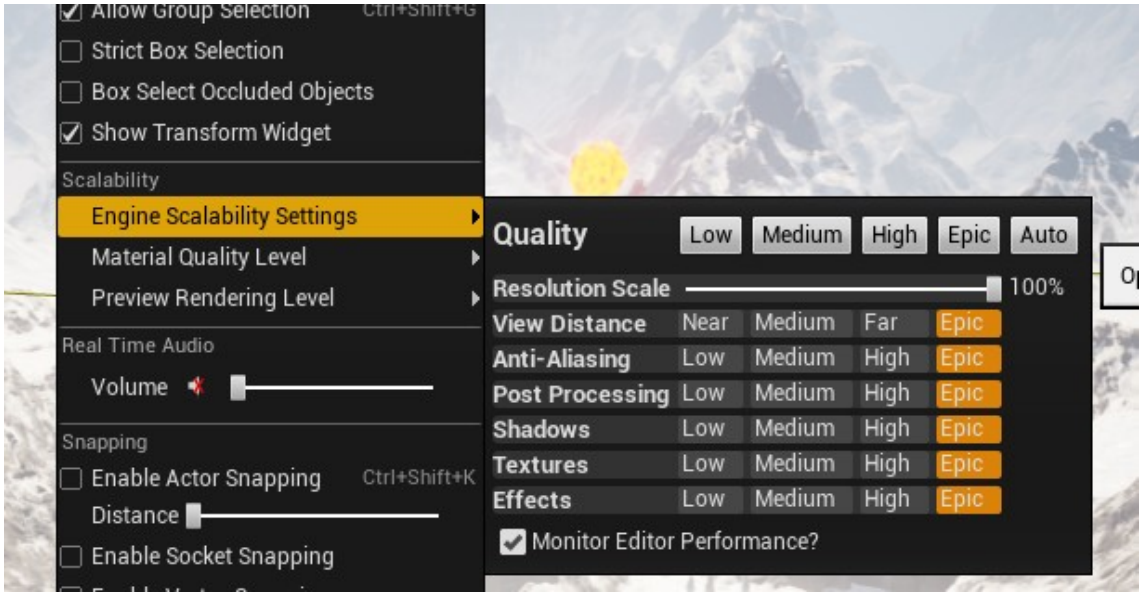
8 SUORITUSKYVYN OPTIMOINTI

Tässä luvussa keskitytään optimoinnin yleisiin periaatteisiin UE4:n kanssa ja käydään optimointitapoja- ja kohteita läpi yleisellä tasolla. Optimointikohteita ja optimointitapoja on pelimoottorin ominaisuuksien määrästä johtuen runsaasti eikä jokaisen yksityiskohtainen läpikäynti ole tämän työn kannalta oleellista. Yksityiskohtaisemmat tarpeet riippuvat projektista ja profiloinnin paljastamista pullonkauloista ja joihinkin keinoihin syvennytään tarkemmin testiprojektien optimoinnissa.

Luvussa esitellään ensiksi yksinkertainen tapa vaikuttaa pelin kuormittavuuteen skaalautuvuusasetuksia käyttämällä. Tämän jälkeen perehdytään jo suunnitteluvaiheessa hyödynnettäviin yleisiin ohjeisiin, jota seuraa CPU-optimoinnille omistettu lyhyt luku. Loput luvuista käsittelevät mahdollisuuksia laskennan tehostamiseksi pääasiassa näytönohjaimen kannalta hyödyntämällä tuoreita UE4:n ominaisuuksia.

8.1 Skaalautuvuusasetukset

Skaalautuvuusasetusten avulla voidaan hallita helposti eri ominaisuuksien laatua ja sitä kautta niiden kuormittavuutta. Unreal Editorissa asetukset ovat työkalupalkin *Settings*-valikossa *Scalability*-kategoriassa. Valikosta ja sen tarjoamat valinnat ovat nähtävissä kuvassa 21. [58]



Kuva 21: Skaalautuvuusasetusten valikko

Asetukset on valittavissa myös konsolista asetettavien muuttujien kautta eli niitä ei tarvitse lukita etukäteen. Asetukset ja niiden vaikutukset ovat seuraavat:

- *Resolution scale (sg.ResolutionQuality)* on ominaisuus, jolla kuva voidaan renderöidä matalammalla resoluutiolla ja skaalata sitten ruudun kokoiseksi. Kuvan skaalaus isommaksi vaatii jonkin verran laskentaa, mutta pienemmän renderöintiresoluution vuoksi se on yleensä silti kannattavaa. Asetus ei vaikuta 2D-käyttöliittymägrafiikkaan, jota dokumentaatio perustelee todennäköisen hyödyn pienuudella suhteessa laadun heikkenemiseen.
- *View distance (sg.ViewDistanceQuality)* määrää etäisyyden kamerasta, jonka jälkeen objekteja aletaan leikkaamaan pois näkyvistä. Etäisyys muodostuu kertoimesta, sekä objekteille määritetystä toivotusta leikkausetäisyydestä, joten kaikki objektit eivät leikkaudu samalla etäisyydellä.
- *Anti-Aliasing (sg.AntiAliasingQuality)* eli reunanpehmennys on kuvanparannustekniikka. Vähentämällä sen käyttöä voidaan vähentää näytönohjaimen kuormitusta.
- *Post Processing (sg.PostProcessQuality)* -arvon avulla voidaan säätää useiden eri kuvanparannustekniikoiden käyttöä. Kuvanparannus vaatii laskentatehoa, joten arvoja säätämällä voidaan vaikuttaa pelin kuormittavuuteen.
- *Shadows (sg.ShadowQuality)* -asetuksella voidaan lyhyesti sanottuna säätää varjoihin liittyviä laatuasetuksia.
- *Textures (sg.TextureQuality)* -asetuksella hallitaan tekstuureihin liittyviä laatuasetuksia. Matalammalla laadulla voidaan vaikuttaa erityisesti näytönohjaimen muistinkäyttöön.
- *Effects (sg.EffectsQuality)* vaikuttaa erilaisiin tehosteisiin, joihin kuuluvat esimerkiksi aktoreihin renderöitävien yksityiskohtien minimimäärän määrittävä arvo (UE4:ssä niin kutsuttu *Detail Mode*).

Sulkuihin on merkitty vastaavan konsolimuuttujan nimi. Listatut asetukset kontrolloivat itse asiassa useiden eri ominaisuuksien arvoja. Tarkemmat tiedot näitä ominaisuuksia hallitsevista muuttujista ja niille asetetuista arvoista on hallittavissa pelimoottorin asennushakemiston tiedostossa *Engine\Config\BaseScalability.ini*. Skaalausasetusryhmien muuttamat arvot ovat säädettävissä myös projektikohtaisesti muokkaamalla projektin hakemiston tiedostoa *Config\DefaultScalability.ini*. Kaikkia arvoja voidaan säätää muuttujien kautta myös ajonaikaisesti esimerkiksi konsolia käyttäen. [58]

Settings → Material Quality Level -valikosta voidaan hallita lisäksi materiaalien laatua. Vaihtoehtoja ovat *Low, Medium* ja *High*. Tämä vaatii materiaalilta tuen kyseiselle asetukselle, eli materiaaliin on oltava määriteltynä, millä tavoin se käyttäytyy eri asetusten ollessa voimassa. [58]

8.2 Yleisiä suunnitteluohjeita

Pelin kuormittavuuteen voidaan vaikuttaa jo hyvissä ajoin ottamalla huomioon joitain perusasioita kappaleiden mallinnuksessa ja kenttäsuunnittelussa. UE4:n dokumentaatio listaa joitain yleisiä ohjesääntöjä suunnittelijoita varten. Kehittäjille tarkoitettuja mallinnusta ja materiaaleja koskevia ohjeita ovat muun muassa [59]:

- Minimoi yksittäisten elementtien määrä objektissa. Yhdistele malleja saadaksesi tarpeeksi pienen määrän kolmioita jokaiseen elementtiin (esimerkiksi yli 300).
- Läpinäkymättömät materiaalit kuormittavat vähiten ja mahdollistavat piiloon jääneiden pikseleiden leikkauksen pois laskennasta. Läpinäkyvät materiaalit kuormittavat eniten.
- Pienempikokoisten tekstuuriformaattien käyttö nopeuttaa materiaalien laskentaa.
- Monimutkaiset varjostinohjelmat materiaaleissa lisäävät kuormitusta, joten niiden käyttö tulisi minimoida materiaaleja optimoidessa. Lisäksi valaisematon (*unlit*) -varjostinmalli, jonka väreihin valot eivät vaikuta, kuormittaa kaikista vähiten, valaistu (*lit*) -varjostinmalli on yleisin ja muut mallit kuormittavat näitä malleja enemmän.
- Pienemmässä skaalassa nähtävistä tekstuureista ei kannata ikinä ottaa pois *Mip-map*-optimointia, sillä sen seurauksena tekstuurien välimuistitusta ei saada hyödynnettyä yhtä tehokkaasti.
- Tesselointi on kallis operaatio, jonka käyttöä pitäisi välttää. Tesselaatiolla (engl. *tessellation*) tarkoitetaan tekniikkaa, jossa pinta voidaan laskennallisesti jakaa tarpeen mukaan useampiin osiin yksityiskohtien lisäämiseksi. Valmiiksi etukäteen tesseloidun mallin käyttö on yleensä tehokkaampaa [60].

Kenttäsuunnitteluun liittyvät ohjeet koskevat puolestaan suurelta osin valojen käyttöä. Näitä ohjeita ovat esimerkiksi [59]:

- Pyri rajoittamaan valonlähteiden määrää.
- Hyödynnä kevyempiä valaistusasetuksia. Valojen liikkuvuustyypit kevyemmästä raskaampaan ovat: täysin staattinen valo (*static*), paikallaan olevat valot (*stationary*) ja täysin dynaamiset valot (*moveable*) [61]. Lisäksi erityisesti alue-valonlähteet kuormittavat hieman muita enemmän. Valonlähteistä laskennallisesti raskaimpia ovat pistemäiset valot, suunnattu valo hieman kevyempää ja kohdevalot puolestaan kevyimpiä.
- Pyri rajoittamaan valon kantamaa ja valon peittämän alueen määrää.
- Poista varjojen laskenta käytöstä valo- tai objektikohtaisesti aina kun se on mahdollista.

Lisäksi kenttäsuunnittelussa tulisi ottaa huomioon objektien leikkautuminen, missä esimerkiksi isoilla objekteilla voidaan peittää näkökenttää ja siten parantaa suorituskykyä.

8.3 CPU-optimointi

UE4:n dokumentaation mukaan CPU:n muodostaessa pullonkaulan kyse on todennäköisesti liian suuresta piirtokutsujen määrästä. Kutsujen määrää voidaan vähentää esimerkiksi yhdistelemällä useita objekteja yhdeksi. Myös monimutkaisemmat materiaalit vaativat enemmän laskentaa. Kutsujen vaatima laskenta-aika muodostuu useista eri operaatioista, kuten GPU:lle lähetettävien kutsujen valmistelusta sekä arvojen tarkistamisista. [54]

Yksi *stat SceneRendering* -työkalulla nähtävä arvo on 3D-objekteihin liittyvä piirtokutsujen määrä *Mesh draw calls*. Näiden kutsujen määrää voidaan tarvittaessa vähentää muun muassa:

- vähentämällä tai yhdistämällä objekteja
- vähentämällä elementtien määrää esimerkiksi vähentämällä tai yhdistämällä materiaaleja
- rajaamalla näkyvyysaluetta

Jos CPU:n liiallinen kuormitus ei johdu piirtokutsuista, vaan *Game*-säikeestä, voi raskaus johtua esimerkiksi blueprinteistä, fysiikkamallinnuksesta, tekoälystä tai muistinvarausoperaatioista. Myös hiukkasjärjestelmä (engl. *particle system*) voi olla syynä *Game*-säikeen vaatimaan laskenta-aikaan. UE4:ssä käytetty hiukkasjärjestelmä on tapa luoda esimerkiksi savuun, kipinöihin ja tuleen liittyviä tehosteita. Raskaan hiukkasjärjestelmän keventämiseksi on useita keinoja. Voidaan esimerkiksi vähentää hiukkasten määriä, niiden elinaikaa, sekä poistaa käytöstä niihin liittyviä raskaita operaatioita kuten törmäyslaskenta. Tarkempi kuormittavuuden syy selviää profiloinnin avulla. [54][62][63]

8.4 Fysiikkamallinnus GPU:lla

Fysiikkamoottorin tehtävänä on huolehtia fysiikkamallinnuksesta ja siihen liittyvästä laskennasta. Työssä käytetty UE4:n versio käyttää fysiikkamallinnuksessa PhysX 3.3 -fysiikkamoottoria [12]. PhysX pystyy periaatteessa hyödyntämään laskennassa CPU:n lisäksi myös näytönohjainta, mutta ominaisuus vaatii toimiakseen ainakin NVIDIA:n CUDA-teknologialla varustetun näytönohjaimen. CUDA on teknologiayhtiö NVIDIA:n tekniikka rinnakkaisen laskennan toteuttamiseksi NVIDIA:n näytönohjaimilla [64]. CUDA sekä sen kaltaiset teknologiat mahdollistavat näytönohjaimen ohjelmoinnin, jolloin näytönohjaimen rooli muuttuu puhtaasta grafiikkasuorittimesta (*graphics processing unit, GPU)* yleiskäyttöisemmäksi GPGPU-laskentayksiköksi (*General Purpose GPU*) [64]. UE4 ei kuitenkaan ilmeisesti käytä GPU-kiihdytystä PhysX:n kanssa paitsi *GPU Particles* -ominaisuuden kanssa, joten UE4:ssä fysiikkamallinnus kuormittaa pääasiassa CPU:ta. [65][66][67]

Uusia mahdollisuuksia GPU:n hyödyntämisessä fysiikkamallinnuksessa voidaan saada UE4:n ja NVIDIAn FleX-integraation kautta. FleX kykenee mallintamaan esimerkik-

si nesteitä, kankaita ja kiinteitä kappaleita sekä niiden keskinäisen vuorovaikutuksen. UE4:n Flex-integraation lähdekoodi on saatavilla GitHub-palvelussa, mutta projekti on rajattu UE4:n lähdekoodin tavoin vain rekisteröityneille käyttäjille. FleXin käyttöä fysiikkamallinnuslaskennan optimoimiseksi ei kuitenkaan tämän työn puitteissa testattu. [68][69][70]

8.5 Uuden Vulkan-grafiikkakirjaston hyödyntäminen

Vulkan on vuonna 2015 julkaistu avoin grafiikkakirjasto, joka pyrkii muun muassa modernisoimaan ja tehostamaan GPU-laskentaa. Vulkan voi siten olla tulevaisuudessa mahdollisesti suorituskykyisempi vaihtoehto nykyisin laajasti käytössä oleville OpenGL:lle ja DirectX:lle. Kohdelaitteistona ovat sekä mobiililaitteisto että tehokkaammat pöytätietokoneiden näytönohjaimet. Parempi laskentateho on tarkoitus saavuttaa muun muassa vähentämällä CPU:n tarvetta grafiikkalaskennassa välttämällä tarpeettomia CPU:lla tehtäviä kutsuja sekä yhdistämällä useita kutsuja yhteen. Vulkan pyrkii myös hyödyntämään paremmin useita CPU-ytimiä samanaikaisesti rinnakkaistamalla laskentaa. [71]

Unreal Engine 4 tukee Vulkan-rajapintaa ja sen käyttäminen on periaatteessa jo mahdollista. Käyttö vaatii tuen näytönohjaimen ajureilta ja tuen pitäisi löytyä nykyään sekä AMD:n että Nvidian tuoreimmista ajureista. Epic Wiki -sivuston mukaan Vulkan käyttää tällä hetkellä kuitenkin vasta mobiililaitteille tarkoitettuja renderöintiominaisuuksia. Vulkan-ominaisuudet eivät välttämättä ole myöskään kaikin puolin viimeisteltyjä. Tässä työssä ei tehdä testausta mobiililaitteistolla, joten Vulkan-kirjaston väitettyjä hyötyjä ei ole työssä testattu. [71][72]

9 ESIMERKKIPROJEKTIEN OPTIMOINTI KÄYTÄNNÖSSÄ

Tässä luvussa testataan profilointityökaluja ja joitain optimointikeinoja käytännössä. Ensiksi esitellään testausympäristöön kuuluvat laitteistot ja ohjelmistot, minkä jälkeen-jälkeen perehdytään testaustapoihin ja käytettyihin asetuksiin. Lopuksi profiloidaan esimerkkiprojektia ja kokeillaan sopivia optimointitapoja suorituksen parantamiseksi.

9.1 Kehitys- ja testausympäristöt

Alustana Unreal Engine 4:lle tässä työssä toimii yksi pöytätietokone ja yksi kannettava tietokone. Pelimoottoria testattiin pöytätietokoneella seuraavilla käyttöjärjestelmillä:

1. Linux Mint 17.3 (linux-ytimen versiolla 4.2.0-36-generic). Työpöytäympäristönä KDE 4.14.2 [73].
2. Microsoft Windows 7 Professional N (Service Pack 1)

Pelimoottoria testatessa Unreal Enginen tarjoama Linux-käyttöjärjestelmän tuki osoittautui työn kannalta lopulta vielä liian rajoittuneeksi. Merkittävimpänä tekijöinä UE:n kauppapaikan Linux-version puuttuminen sekä virallisen tuen puuttuminen useilta kauppapaikasta ladattavilta sisällöiltä. Kauppapaikasta ladatun sisällön siirto Linux-puolelle onnistui käsinkin, mutta kävi työlääksi ottaen huomioon että sisältö oli jo siinä vaiheessa Windows-ympäristössä käyttövalmiina. Testiprojekteja kokeiltiin siten ainoastaan Windows-ympäristössä. Molemmilla tietokoneilla käyttöjärjestelmän versio on Microsoft Windows 7 Professional N (Service Pack 1).

Tietokoneiden tarkemmat laitteistotiedot ovat oleellisia tulosten toistettavuuden sekä yleisen vertailtavuuden kannalta. Pöytätietokoneen tärkeimmät komponentit on esitetty taulukossa 2. Kannettavan tietokoneen laitteistotiedot on vastaavasti esitetty taulukossa 3.

Taulukko 2: Pöytätietokoneen keskeiset laitteistotiedot

Näytön resoluutio (L x K)	1920 x 1200 pikseliä
Prosessori	Intel Core i5-6600
Keskusmuisti	16 Gt dual-channel (2 kpl 8 Gt moduulia) DDR4 2133 MHz, CL14
Näytönohjain	ASUS Turbo GeForce GTX 1060 (6 Gt VRAM, Nvidian grafiikkapiiri)
Massamuistilaite	Samsung EVO 850 SSD

Taulukko 3: Kannettavan tietokoneen keskeiset laitteistotiedot

Näytön resoluutio (L x K)	1920 x 1080 pikseliä
Prosessori	Intel Core i7-2670QM
Keskusmuisti	8 Gt (2 kpl 4 Gt moduulia) DDR3 1333 Mhz
Näytönohjain	AMD FirePro™ M5950 (1 Gt VRAM)
Massamuistilaite	Kingston V+200 120 Gt SSD

9.2 Mittaustavat

UE4:n dokumentaation ohjeiden mukaan profilointi ja testaus tulisi tehdä niin lähellä kohdealustaa kuin mahdollista. Lisäksi profilointiin vaikuttavien häiriötekijöiden määrä tulisi minimoida. Esimerkiksi Unreal Editor voi itse vaikuttaa mittaustuloksiin ja suljetaan ennen profilointia luotettavampien tulosten saamiseksi. Joissain tapauksissa voi olla hyödyllistä jopa muokata pelin lähdekoodia toistettavaan testaukseen sopivammaksi esimerkiksi poistamalla satunnaislukugeneraattorien vaikutuksen.

9.2.1 Käännös- ja ajoasetukset

Työssä testattavat projektit käännetään aina ensin valmiiksi paketiksi ennen suoritusta, jotta Unreal Editorin aiheuttamalta mahdolliselta ylimääräiseltä kuormitukselta ja muilta vaikutuksilta voidaan välttyä. Testaus aloitetaan luomalla ensin Elemental Demo -sisällöstä projekti *Create Project* -napin avulla. Yksi tapa tehdä tämä on kuvassa 24 näkyvän sivun kautta. Ensimmäiseksi luotua projektia ei muokata lainkaan, vaan projektin muokkaamista varten luodaan toinen projekti. Näin saadaan säilytettyä muokkaamaton projekti vertailukelpoisena.

Projektit paketoidaan *Win64*-alustalle. Projekteista käännetään kehitysversio (*Development Build*), jotta profilointityökalut olisivat käytettävissä. Tämä ei ole suoritustehon kannalta parhain asetus, mutta ei todennäköisesti muodosta itsessään merkittäviä pullonkauloja laskentaan. Optimaalisin käännösvalinta olisi julkaistavaa versiota varten tarkoitettu *Shipping*, jossa ylimääräiset kehitys- ja testausominaisuudet on karsittu pois. [49]

Dokumentaation suositusten mukaisesti pystytahdistusta (*vertical sync*) ei peliä ajettaessa käytetä. Pystytahdistuksen tilan voi tarkistaa konsolista tarkistamalla muuttujan *r.VSync* arvon. [52]

9.2.2 Profiloinnin suoritus

Luotettavamman datan saamiseksi profilointidatan keruuseen käytetään Profiler-työkalua käynnistämällä Unreal FrontEnd -työkalu itsenäisesti. Kerätty profilointidata tallennetaan tiedostoon myöhempää arviointia varten käyttäen työkalun ominaisuutta *Data Capture*. Tiedon kerääminen käynnistetään käsin *Data Capture* -nappia painamalla. Tässä vaiheessa pelin istunto täytyy olla valittuna. [52]

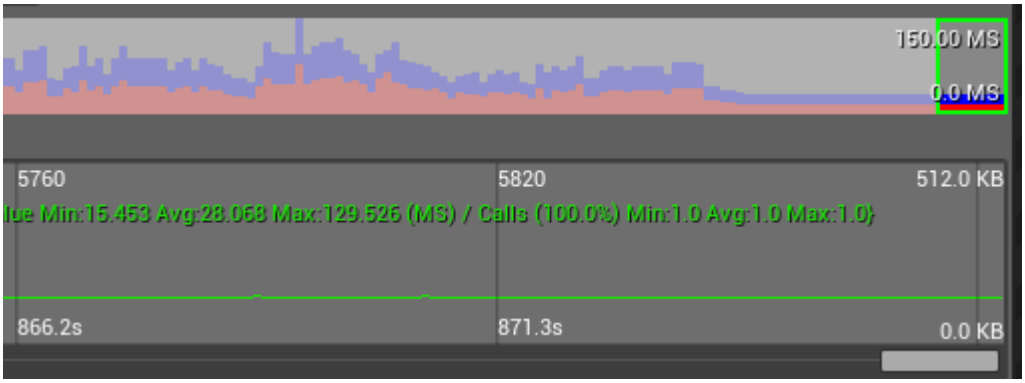
Pelin ja Unreal FrontEnd -työkalun välistä kommunikaatiota varten peli suoritetaan käyttäen parametria ”`-messaging`”. Tämä onnistuu suorittamalla peli esimerkiksi komentoriviltä tai sopivalla pikakuvakkeella. Testatessa osoittautui hyödylliseksi suorittaa sovellusta ennalta määritetty aika, jotta profilointidataa saadaan kerättyä suunnilleen saman verran. Tähän käytettiin seuraavanlaisia komentojonotiedostoa:

```
1.    start C:\Users\TestUser\Documents\PackagedElementalDemo\WindowsNoEditor
      \ElementalDemo.exe -messaging
2.    timeout /t 180
3.    taskkill /im "ElementalDemo.exe" /t
```

Ohjelma 4: Komentojonotiedosto ”start_elementaldemo.bat”, joka käynnistää sovelluksen sekä lopettaa sen määritetyn ajan jälkeen

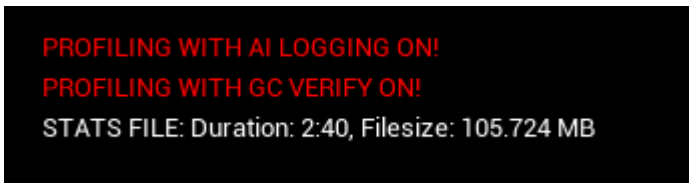
Rivi 1 käynnistää paketoidun ohjelmabinäärin käyttäen parametria ”`-messaging`”. Rivillä 2 odotetaan 180 sekuntia, jonka jälkeen rivin 3 komento lopettaa ”*ElementalDemo.exe*”-nimisen prosessin ja kaikki sen lapsiprosessit. Projektin suoritukseen kokeiltiin myös Unreal Frontend -työkalun ”Project Launcher”-välilehteä, mutta projektin paketointi sitä käyttäen epäonnistui virheen vuoksi joka kerta.

Testien suorituskerroista katsotaan FPS-lukeman minimiarvo, maksimiarvo sekä keskiarvo yleisen suorituskyvyn arvioinniksi. Profilointinäytteiden kesto mitataan profiloinnin aikana ruudulla näytetystä ajasta. Dokumentaation mukaan Profiler-työkalun Data Graph -ikkunan alareunan pitäisi esittää aika alkaen nollasta, mutta arvot eivät ole odotetunlaiset [74]. Kuvassa 22 nähdään työkalun esittämät arvot sekunteina. Arvot näyttävät näytteen loppupuolella ”866.2s” sekä ”871.3s”, mutta todellisuudessa aikaa on kulunut selvästi vähemmän.



Kuva 22: Data Graph -ikkuna. Yläreunassa piirretyn ruudun numero ja alareunassa aika.

Aika katsotaan siten profiloinnin aikana ruudun vasemmalla puolella nähtävistä lisätiedoista, joista nähdään esimerkki kuvassa 23.

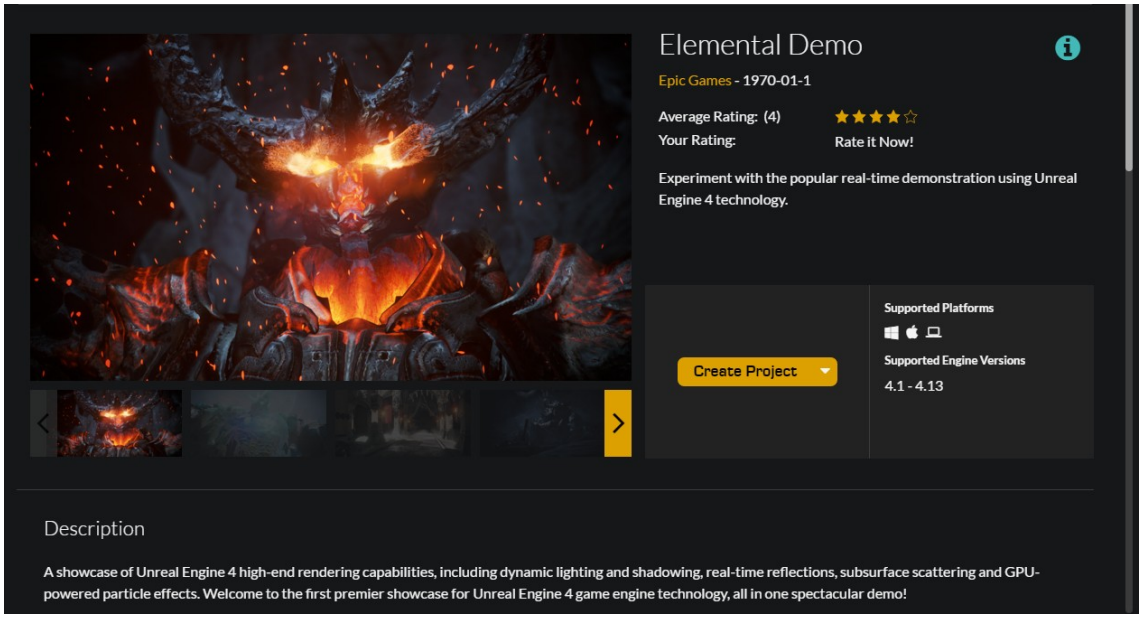


Kuva 23: Ruudussa näkyvät lisätiedot profiloinnin ollessa käynnissä pelimoottorin versiolla 4.12.5.

Ajastimen tarkkuus vahvistettiin vertaamalla lukemia matkapuhelimen sekuntikellon näyttämään aikaan. Ajan seuraaminen silmämääräisesti tuo epätarkkuutta, mutta tulosten kannalta esimerkiksi noin 1-2 sekunnin epätarkkuus ei ole merkittävä. Lisäksi profiloinnin käynnistäminen käsin ja ohjelman käynnistyminen eri nopeudella eri kertoina aiheuttavat vastaavia epätarkkuuksia näytteen pituuteen sekä profiloinnin aloituskohtaan. Tällä ei ole merkitystä käsiteltäessä rajattuja alueita näytteestä, mutta se voi vaikuttaa esimerkiksi koko näytteen FPS-keskiarvoon.

9.3 Testattava ohjelma: Elemental Demo

Testiprojektina profiloinnin ja optimointikeinojen havainnollistamiseksi käytetään kauppapaikan *Learn*-osiosta saatavilla olevaa *Elemental Demo* -animaatiota. Se on Epic Gamesin julkaisema pelimoottorin ominaisuuksia demonstroiva esitys, joka on vapaasti ladattavissa UE4:n kauppapaikalta. Kyseessä ei siis ole varsinaisesti peli, sillä käyttäjä ei esimerkiksi itse ohjaa mitään hahmoa. Tämä on kuitenkin myös etu testattavuuden kannalta. Kuvassa 24 nähdään kauppapaikan sivu, josta tämän testauksessa käytetyn sisällön voi ladata.



Kuva 24: Elemental Demo -projektin sivu UE:n kauppapaikalla

Sisällön nimi- ja versiotiedot ovat nähtävillä taulukossa 4. Kuten kuvasta 24 nähdään niin versionumeroa tai julkaisupäivää ei ole valitettavasti saatavissa. Kauppapaikan sivu näyttää päivämäärän väärin päiväksi ”1970-01-1”. Myöskään sisällöstä luodut projektit eivät projektin tiedoissa näytä sisältöön liittyvää versionumeroa, vaan projektin versiona näkyy ”1.0.0”. Tämän vuoksi on hankala jäljittää sisällössä tapahtuvia mahdollisia muutoksia.

Taulukko 4: Testiohjelman nimi- ja versiotiedot

Nimi	Elemental Demo
Versio	Ei saatavilla
Tuetut pelimoottorin versiot	4.1-4.13 (4.13 on uusin pelimoottorin versio)
Testauksessa käytetty pelimoottorin versio	4.12.5

Luotujen *ElementalDemo*-projektien skaalautuvuusasetukset tarkistettiin konsolin kautta. Oletusarvot ovat nähtävillä taulukossa 5. Nämä ovat siis projektien voimassa olevat arvot ellei niitä erikseen muokata. Arvoista nähdään niiden olevan skaalautuvuusasetusten maksimiarvot [58].

Taulukko 5: Skaalausasetukset muokkaamattomassa projektissa

Muuttuja	Alkuperäinen arvo
sg.ResolutionQuality	100
sg.ViewDistanceQuality	3
sg.AntiAliasingQuality	3
sg.PostProcessQuality	3
sg.ShadowQuality	3
sg.TextureQuality	3
sg.EffectsQuality	3

9.4 Profilointi ja optimointi pöytäkoneella

Profilointi päätettiin tehdä ensin 4 kertaa, jonka aikana saatiin käsitys FPS-lukemien vakaudesta. Maksimiarvosta nähdään vaihtelua tapahtuvan hyvinkin paljon, mutta keskiarvo sekä minimiarvo pysyvät vakaampina.

Laitteisto suoriutuu animaation toistamisesta ongelmitta ja liike on sulavaa. Tämä näkyy taulukon 6 esittämissä FPS-arvoissa: keskiarvo on jopa yli 60. Näytön virkistystaajuuden ollessa 60 hertsiä, ei näyttö sitä korkeampaa FPS-taajuutta pysty myöskään esittämään. Minimiarvoista kuitenkin nähdään, että paikoittain suoriutuminen on ollut heikkoa.

Taulukko 6: Muokkaamattoman projektin FPS-arvot eri suorituskertoina

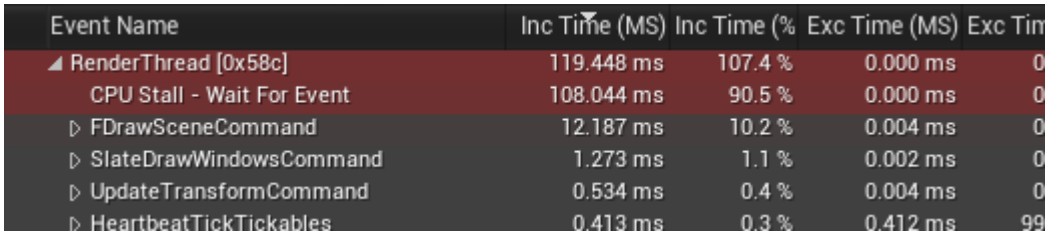
Ajokerta	*1*	*2*	*3*	*4*
Kesto (sekuntia)	171	171	172	170
FPS: minimiarvo	15,46	14,83	14,23	15,27
FPS: maksimiarvo	129,71	79,83	67,65	69,88
FPS: keskiarvo	62,07	62,06	62,06	62,06

Tavoitteeksi valitaan tässä tilanteessa pitää FPS-lukema aina yli 30:n, jolloin taataan animaatioon kauttaaltaan sopivantasoinen sulavuus. Tätä varten etsitään ensin optimointia vaativat kohdat Profiler-työkalun luoman graafin avulla. Graafi on nähtävillä kuvassa 25.

Kuva 25: Profiler-työkalun näyttämä karkea kuvaaja säikeiden käyttämistä ajoista. Kuvan värikylläisyyttä on lisätty graafia peittävän harmaan kerroksen häivyttämiseksi.

Samanlainen tilanne on nähtävissä jokaisen ajokerran graafissa. Tavoitteen saavuttamiseksi projektista on siten optimoitava yksi alue. Skaalautuvuusasetukset päätetään siten pitää oletusarvoissaan laadun pitämiseksi parhaalla mahdollisella tasolla.

Tutkitaan seuraavaksi *Event Graph* -ikkunan avulla eri säikeisiin ja ominaisuuksiin käytettyä laskenta-aikaa. Valitaan kuvaajasta tarkasteltavaksi alue siten, että kuvassa 25 näkyvä huippukohta sisältyy valintaan. Tämän jälkeen valitaan näytettävien arvojen tyypiksi ”*Maximum*”, jolloin saadaan valitulta ajanjaksolta valittua tapahtumien laskentaan käytetyt maksimiajat. Listassa nähdään runsaasti kohteita, joiden suoritukseen on sarakkeen *Inclusive Time* mukaan kulunut paljon aikaa. Tarkemmin tarkasteltaessa kuitenkin huomataan, että suurin osa ajasta on mennyt yksinkertaisesti odottamiseen. Tällaisia odotusaikaa laskevia kohteita löytyy eri säikeistä useita ja niiden nimessä on yleensä sana ”stall” tai ”idle”. Kuvassa 26 nähdään esimerkki *RenderThread*-säikeestä, jossa suurin osa ajasta on kulunut tapahtumaan ”*CPU Stall – Wait For Event*”, joka ei ole etsitty pullonkaula.



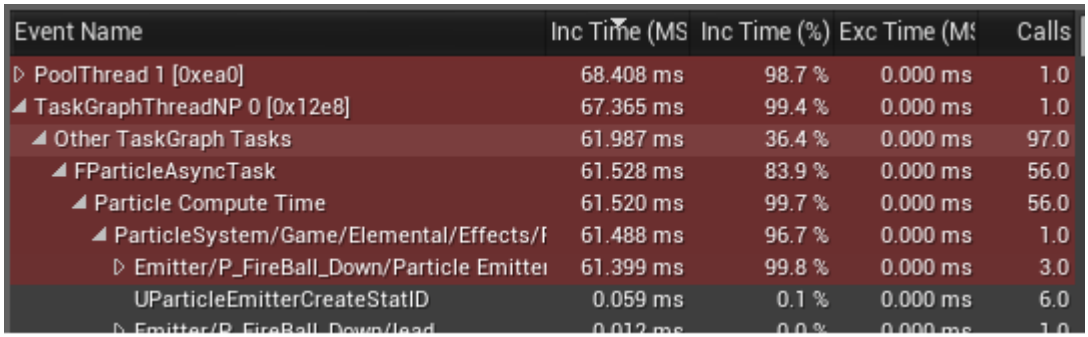
Kuva 26: RenderThread-säikeen laskenta-aikojen maksimiarvot valitulla alueella

GameThread-säiettä tutkimalla löydetään samanlainen tilanne, missä 65,508 millisekunnin kokonaisajasta tapahtuma ”*CPU Stall”* käyttää aikaa 60,772 millisekuntia. Oletuksena tapahtumat on järjestetty puumaiseen näkymään ja etsintää voidaan jatkaa esimerkiksi käymällä rakennetta läpi kunnes löydetään tapahtuma, jossa aikaa ei käytetä odottamiseen. Apuna voi myös käyttää listan järjestelyä eri tavoin. Kuvassa 27 nähdään käyttöliittymän painikkeet, joiden avulla listan järjestelytapaa voidaan muuttaa.

Kuva 27: Valinnat tapahtumalistan järjestelemiselle eri tavoin

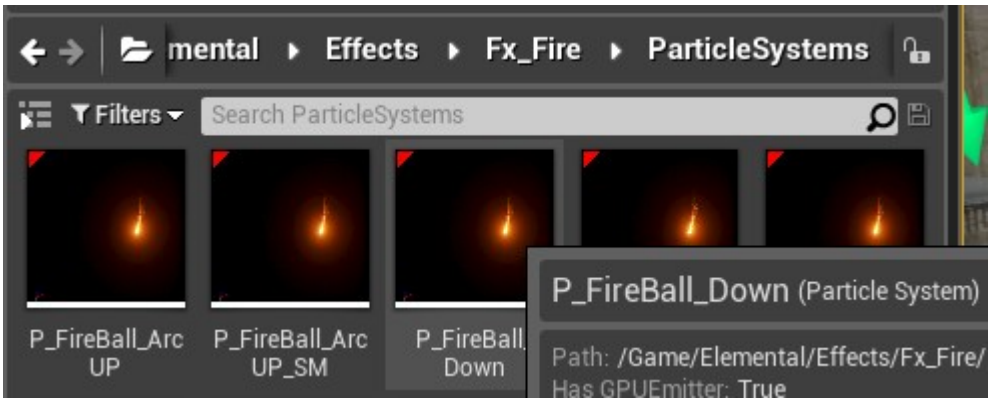
Esimerkiksi *Hierarchical*-painikkeen oikealla puolella olevan *Inclusive*-painikkeen muodostama lista voi olla helpompi käydä läpi. Painikkeiden kuvauksista selviää niiden tarkempi tarkoitus. Tässä tapauksessa kuvauksessa lukee: ”*Flat list of events, sorted by the inclusive time*”.

Etsitty pullonkaula löytyy säikeestä *TaskGraphThreadNP 0*, jonka tietoja nähdään kuvassa 28. Säie käyttää aikaa kokonaisuudessaan 67,365 millisekuntia, josta hiukkasjärjestelmää käyttävä tehoste ”*ParticleSystem/Game/Elemental/Effects/Fx_Fire/ParticleSystems/P_FireBall_Down.P_FireBall_Down”* käyttää 61,399 millisekuntia.



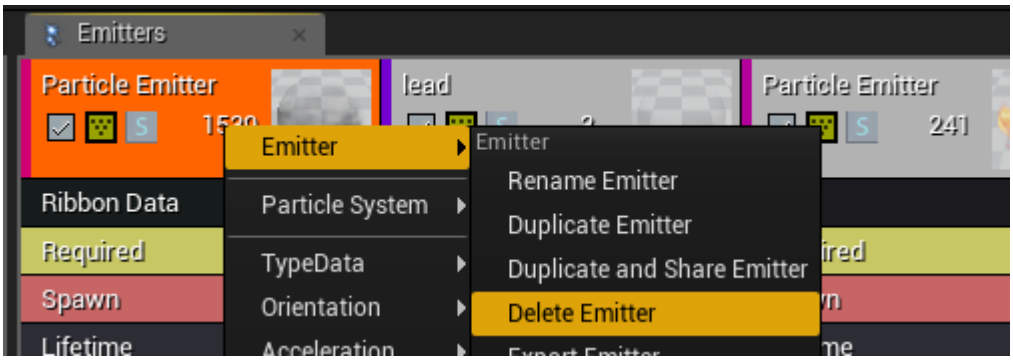
Kuva 28: Pullonkaulan muodostava säie

Muokattavan sisällön etsiminen on yksinkertaista, sillä sen sijainti näkyy suoraan sen nimessä. Tehoste avataan muokattavaksi Unreal Editorin *Content Browser* -ikkunan kautta kaksoisklikkaamalla kuvassa 29 näkyvää kohdetta.



Kuva 29: "P_FireBall_Down"-tehosteen sijainti Unreal Editorissa

Kohde avautuu UE4:n Cascade Editor -työkalussa. Kokeilemalla huomataan ensimmäisen *Particle Emitter* -komponentin olevan yksi merkittävä tekijä laskennassa. Yksinkertaisena ratkaisuna komponentti poistetaan tehosteesta kokonaan. Kuvassa 30 nähdään hiiren oikeanpuoleisella napilla aukeava valikko, jonka kautta poisto voidaan tehdä.



Kuva 30: Ensimmäisen Particle Emitter -komponentin poistaminen hiiren avulla aukeavan valikon kautta

Työkalulla nähdään tehosteesta myös esikatselu. Yksittäisiä komponentteja voi esikatsella klikkaamalla komponentissa näkyvää pientä S-kuvaketta. Esikatselun avulla voi seurata tekemiensä muutosten vaikutusta lopputulokseen. Hiukkastehosteiden yksityiskohtaisempaan profilointiin voi halutessaan käyttää myös *Stat particles* -työkalua [62].

Muutosten jälkeen suoritetaan profilointi uudestaan. Taulukko 7 esittää muokatun projektin suorituskertojen FPS-arvoja ja siitä nähdään suorituskerran 1 arvoista, että minimiarvo on edelleen liian pieni. Myös profiler-työkalun näyttämässä kuvaajassa nähdään samanlainen piikki kuin aiemminkin. Jatketaan tutkintaa etsimällä työkalulla pullonkaulan muodostavaa tapahtumaa uudestaan. Pullonkaulana ei ole enää tehoste *P_FireBall_Down* vaan *P_FireBall_StraightDown.* Tehosteen tarkka polku on ”*ParticleSystem/Game/Elemental/Effects/Fx_Fire/ParticleSystems/P_FireBall_StraightDown.P_FireBall_StraightDown”.*

Tehostetta tutkimalla päädytään yksinkertaisuuden vuoksi samankaltaiseen ratkaisuun kuin aiemmin: etsitään tehosteesta poistettavaksi sopiva komponentti, jolla saadaan vähennettyä riittävästi tarvittavaa CPU-laskentaa. Kokeilujen jälkeen päädytään jälleen poistamaan ensimmäinen *Particle Emitter* -komponentti. Muutoksen jälkeisen testiajon tulokset ovat merkittynä taulukon 7 suorituskerran 2 kohdalle.

Taulukko 7: Muokatun projektin testiajojen mittaustuloksia

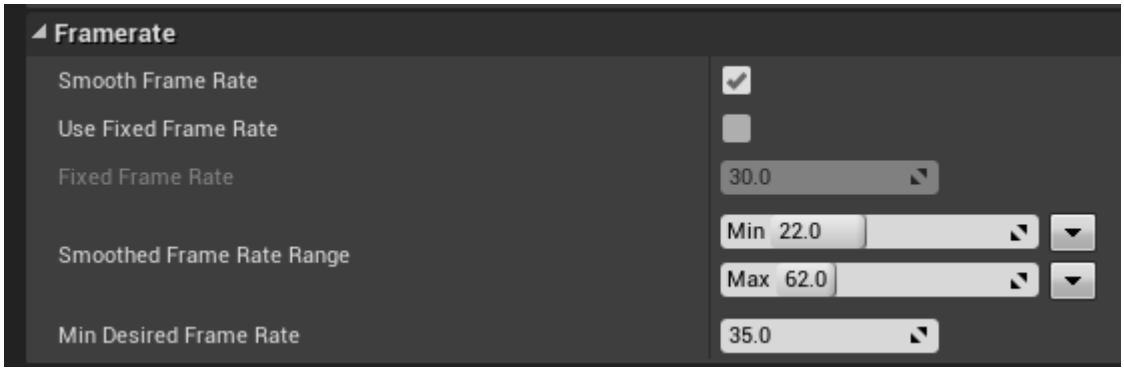
Suorituskerta	*1*	*2*	*3*	*4*	*5*
Kesto (sekuntia)	171	172	172	170	172
FPS: minimiarvo	9,76	**53,80**	**51,12**	**53,98**	**39,52**
FPS: maksimiarvo	67,51	75,08	67,83	66,32	68,46
FPS: keskiarvo	62,00	62,07	62,06	62,06	62,06

Toisen tehosteen muokkaamisen jälkeen minimiarvo on siis selvästi yli 30, joka on tavoitteen mukainen. Suorituksia tehdään vielä 3 lisää, jotta saadaan riittävä varmuus muutosten riittävyydestä. Lisäksi ennen suorituskertaa 5 tietokone käynnistetään kokeilumielessä uudestaan, jotta nähtäisiin onko sillä merkittävää vaikutusta tuloksiin. Kaikkien suorituskertojen arvot ovat myös nähtävillä taulukossa 7. Kuvassa 31 on esitetty vertailun vuoksi Profiler-työkalun piirtämä kuvaaja optimoidun projektin suorituksesta.

Kuva 31: Testilaitteistolle optimoidun version kuvaaja. Kuvan värikylläisyyttä on lisätty graafia peittävän harmaan kerroksen häivyttämiseksi.

Kuvaajassa näkyvä maksimiarvo on nyt 53,13 millisekuntia kuvassa 53 näkyvän 120,99 millisekunnin sijaan. Myös loput suorituskerrat täyttävät tavoitteen vaatimukset, joten todetaan optimointitavoitteen olevan saavutettu. Keskiarvo pysyy edelleen 60:n

tuntumassa jatkuvasti, mikä saa epäilemään jonkin ominaisuuden rajoittavan suoritusta. Projektin asetuksista löytyykin kuvassa 32 nähtävät valinnat. Tämä voi hyvinkin olla syy keskiarvon pysymiseen lähellä arvoa 62. Rajoittimella ei ole kuitenkaan merkitystä tämän työn kannalta.



Kuva 32: Framerate-asetukset projektin asetuksissa

9.5 Profilointi ja optimointi kannettavalla tietokoneella

Aloitetaan profilointi kannettavalla tietokoneella suorittamalla profilointi ensin 2 kertaa, jonka jälkeen järjestelmä käynnistetään uudelleen ja profilointidataa kerätään uudestaan kahdesta uudesta ajokerrasta. Profilointidatasta kerätty tieto on esitetty taulukossa 8.

Taulukko 8: Skaalausasetukset muokkaamattomassa projektissa

Suorituskerta	*1*	*2*	*3*	*4*
Kesto (sekuntia)	160	163	162	164
FPS: minimiarvo	1,10	1,12	0,90	1,06
FPS: maksimiarvo	30,97	32,85	38,06	45,93
FPS: keskiarvo	12,28	12,43	11,44	11,88

Maksimiarvoissa tapahtuu paikoin isojakin vaihteluita, mutta keskiarvo antaa paremman käsityksen kokonaisuudesta. Taulukon keskiarvolukemista nähdään, että suoriutuminen on heikkoa eikä liike myöskään näytä sulavalta. Tavoitteeksi otetaan siten ensiksi suorituskyvyn parantaminen niin että keskiarvo saadaan nostettua yli 30:n.

Seuraavaksi tutkitaan, missä laskenta-aikaa käytetään eniten. *Stat unit* -työkalun avulla nähdään, että yhtenä ajanhetkenä työkalun näyttämät arvot ovat:

- *Frame: 97,58 ms*
- *Game: 8,03 ms*
- *Draw: 26,28 ms*
- *GPU: 97,79 ms*

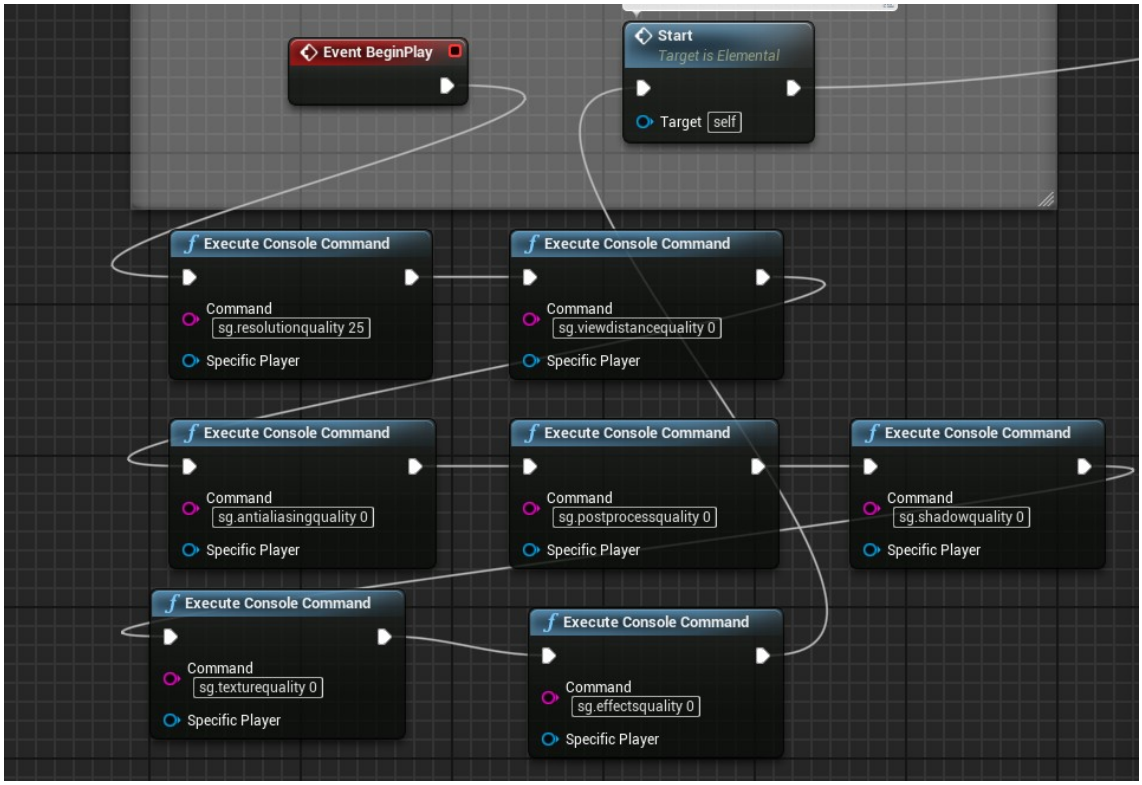
Tästä voidaan päätellä, että näytönohjain muodostaa laskennan pullonkaulan [52]. Silmämääräisesti todettuna tilanne myös pysyy samankaltaisena suurimman osan suoritusajasta. Luvuista voidaan myös havaita, että jostain syystä *GPU*-arvo on suurempi kuin *Frame*-arvo.

Profilointia jatketaan keskittymällä GPU:n profilointiin. Tähän käytetään ProfileGPU-työkalua. Jostain syystä työkalun graafinen käyttöliittymä ei kuitenkaan avaudu, kun profilointia tehdään Unreal Frontend -työkalulla ilman Unreal Editoria. Profilointitiedot tulostuvat kuitenkin myös tekstinä lokiviesteihin. Osa lokiviesteistä on esitetty seuraavilla riveillä:

```
81.60ms   Lights   Events 1   Draws 373
81.60ms   DirectLighting   Events 1   Draws 373
...
134.73ms   FRAME   Events 1   Draws 598
134.39ms   Scene   Events 1   Draws 598
1.94ms   InjectTranslucentVolume   Events 6   Draws 12
74.84ms   ShadowedLights   Events 1   Draws 308
24.87ms   PostProcessing   Events 1   Draws 62
31.02ms   StandardDeferredLighting   Events 5   Draws 29
27.46ms   ShadowDepthsFromOpaqueProjected   Events 7   Draws 257
28.05ms   WholeScene   Events 6   Draws 171
...
```

Aikaa näyttäisi kuluvan paljon muun muassa valojen ja varjojen laskentaan. Aloitetaan projektin optimointi yksinkertaisesti säätämällä skaalautuvuusasetuksia. Asetuksia muutettaessa huomattiin, että valikon kautta käytettävät skaalausasetukset **eivät** näytä vaikuttavan paketoituun peliin. Pelit voivat kuitenkin periaatteessa tarjota asetusvalikoita ja asetustiedostoja, joiden kautta käyttäjä voi asetuksia muuttaa. Tässä tilanteessa käytetty ratkaisu oli asettaa arvot muokkaamalla Level Blueprintiä. Kuvasta 33 nähdään kuinka asetuksien hallinta voidaan toteuttaa.

Käytetty funktio on siis *Execute Console Command* ja *Command*-kenttään asetetaan komennoksi muokattava muuttuja ja uusi muuttujan arvo, kuten esimerkiksi ”sg.resolutionquality 25”. Komennot on asetettu suoriutumaan heti *BeginPlay*-tapahtuman jälkeen. Muokkaamisen jälkeen painetaan Blueprint Editorin *Compile*-nappia, poistutaan Blueprint Editorista ja tallennetaan projekti. Tämän jälkeen projekti voidaan paketoida uudestaan.



Kuva 33: Skaalausasetusten muokkaus blueprintissä

Tehtyjen muutoksien vaikutukset ovat nyt selvästi nähtävissä. Sopivia skaalautuvuusasetusten arvoja haetaan asettamalla arvot ensin mahdollisimman pieniksi, jotta nähdään kuinka laitteisto hyvin laitteisto suoriutuu minimiarvoilla. Tämän jälkeen pystytään paremmin arvioimaan millaiset asetukset voisivat olla laitteistolle sopivat tavoitteen saavuttamiseksi. Eri suorituskerroilla käytetyt skaalautuvuusasetukset on esitetty taulukossa 9.

Kuvan terävyydestä tingitään asettamalla *sg.ResolutionQuality* pienemmäksi, jotta muut ominaisuudet voitaisiin säilyttää paremmalla tasolla. *sg.AntiAliasingQuality* -arvo valitaan isoksi pienemmän *sg.ResolutionQuality* -arvon vuoksi pehmentämään reunoja. Lisäksi sen käytöstä on laskennan kannalta etua kuvan skaalaukselle [58]. Muut valinnat eivät perustu suoraan mihinkään suositukseen. Varjojen laatu pidetään matalana, sillä niihin kului aiemmin suhteellisen paljon laskenta-aikaa. *Texture Quality* -arvo nostettiin lopulta maksimiinsa, sillä vaikutus arvioitiin pieneksi GPU-laskennan kannalta [58].

Taulukko 9: Skaalautuvuusasetukset muokatun projektin testiajoissa

Muuttuja	Muokatut arvot, suorituskerta 1	Muokatut arvot, suorituskerta 2	Muokatut arvot, suorituskerta 3
sg.ResolutionQuality	25	50	50
sg.ViewDistanceQuality	0	0	0
sg.AntiAliasingQuality	0	3	3
sg.PostProcessQuality	0	1	1
sg.ShadowQuality	0	1	1
sg.TextureQuality	0	1	3
sg.EffectsQuality	0	1	2

Suorituskerran 3 jälkeen löytyi tavoitteen täyttävät asetukset. Ajokertoja vastaavat FPS-arvot ovat taulukossa 10.

Taulukko 10: Muokatun projektin testiajojen mittaustuloksia

Suorituskerta	*1*	*2*	*3*
Kesto (sekuntia)	163	161	165
FPS: minimiarvo	8,81	7,36	2,83
FPS: maksimiarvo	74,06	71,73	70,97
FPS: keskiarvo	60,92	37,47	34,07

Alkuperäiseen tavoitteeseen päästään jo toisen ajokerran asetuksilla, mutta arvoja nostettiin vielä kolmatta ajokertaa varten. Nostetuilla arvoilla FPS-arvo kuitenkin putoaa välillä selvästi ajon aikana, mikä näkyy hetkellisenä ruudun pysähtymisenä. Tämä voi häiritä katselukokemusta.

Skaalautuvuusasetuksilla on todennäköisesti mahdollista ohjata laskentaa jonkin verran mieluisiin ominaisuuksiin toisten kustannuksella. Valittuja asetuksia ei ole vertailtu muunlaisiin valintoihin ja optimaaliset asetukset ovat tässä suhteessa luultavasti jossain määrin makuasia, sillä arvostus liikkuvan kuvan eri ominaisuuksia, kuten sulava liikettä tai objektien yksityiskohtia kohtaan, voi riippua katsojasta.

9.6 Tulosten arviointi

Esimerkkiprojektin tapauksessa ei suorituskertojen välillä pitäisi periaatteessa olla laskennan kannalta merkittäviä eroja, sillä käyttäjän syöte ei vaikuta suoritukseen. Lisäksi eri ajokertojen kestoa rajattiin ohjelmallisesti, jotta ajokerroista saataisiin mahdollisimman samanlaisia. Käytännössä kuitenkin profiloinnin käynnistäminen käsin sekä erot

sovelluksen käynnistymisnopeudessa eri ajokertojen välillä vaikuttavat profiloinnin aloitus- ja lopetusajankohtiin. Tällä ei kuitenkaan ole merkittävää vaikutusta FPS-keskiarvoihin eivätkä aloitus- ja lopetuskohdat ole Elemental Demo -esimerkkiprojektin sisällön kannalta merkittäviä.

Profiloimalla saaduissa arvoissa tapahtuu suorituskertojen välillä myös mittaustavoista riippumatonta vaihtelua FPS-arvoissa. Unreal Engine 4:n dokumentaation mukaan joskus yllättävätkin tekijät, kuten esimerkiksi näytönohjaimen ajurit, voivat vaikuttaa suoriutumiseen, joten saatujen tulosten toistettavuutta eri ympäristöissä on hankala arvioida. Lisäksi kaikkia käyttöjärjestelmässä suoritettavia prosesseja ei ole mielekästä yrittää sulkea, joten ne voivat vaikuttaa tuloksiin, mutta toisaalta sama vaikutus on odotettavissa myös pelin lopullista versiota suorittavalla tietokoneella.

FPS-keskiarvon vaihtelu eri suorituskertojen välillä ei ole kuitenkaan merkittävää. Keskiarvo oli kannettavalle tietokoneelle optimoitaessa ensisijainen mitattava kohde, joten yksittäisen suorituskerran keskiarvoa voitaneen pitää kohtalaisen luotettavana. Arvot myös käyttäytyivät projektia muokatessa odotetunlaisesti.

Pöytäkoneelle optimoitaessa tarkasteltiin FPS-minimiarvoa, jossa vaihtelua tapahtuu enemmän. Tämän vuoksi suorituskertoja tehtiin lopulliselle versiolle 4 ja ennen viimeistä suorituskertaa tietokone käynnistettiin uudelleen. Minimiarvo pysyy kuitenkin tavoitteen mukaisena, mutta suorituskertojen määrää kasvattamalla voitaisiin lisätä tulosten luotettavuutta.

10 YHTEENVETO

Tässä työssä käsiteltiin Unreal Engine 4 -pelimoottoria sekä siihen liittyviä työkaluja. Pelimoottoria kokeiltiin Microsoft Windows -käyttöjärjestelmän lisäksi myös Linux-käyttöjärjestelmällä. Linux-käyttöjärjestelmällä ei kuitenkaan ollut saatavilla kaikkia työkaluja ja virallinen tuki rajoittuu tällä hetkellä vain Ubuntu Linuxiin.

UE4:n työkalujen käytön perusteiden opettelun kautta saatiin pohja pelimoottorilla tehtyjen projektien perustason muokkaukseen sekä profilointityökalujen käyttöön. Työn aikana perehdyttiin erilaisiin keinoihin pelimoottorilla tehtyjen peliprojektien suorituskyvyn parantamiseksi. Keinot vaihtelivat paremmasta kenttäsuunnittelusta uudenaikaisen grafiikkakirjaston käyttöön. Lopullisena tavoitteena oli tiedon hyödyntäminen käytännössä profiloimalla ja optimoimalla olemassa olevaa projektia sopivammaksi käytetylle testilaitteistolle.

Optimoitavaksi sisällöksi helpon saatavuutensa ja sopivan korkeiden laitteistovaatimusten vuoksi valikoitui *Elemental Demo,* joka on Epic Gamesin julkaisema pelimoottorin ominaisuuksia esittelevä audiovisuaalinen demonstraatio. Projektin profilointia ja optimointia kokeiltiin pöytätietokoneella sekä kannettavalla tietokoneella.

Pöytätietokoneella suoriutuminen oli jo valmiiksi keskimäärin erinomaista (FPS-keskiarvo yli 60), mutta kauttaaltaan se ei sitä ollut, sillä paikoin arvo laski alle 20:n. Tavoitteeksi otettiin siten FPS-arvon minimiarvon nostaminen jatkuvasti yli 30:n. *Profiler*-työkalun avulla saatiin selvitettyä optimointia vaativat kohteet: optimointitarve kohdistui vain lyhyehkölle ajanjaksolle, joka sisältää pelimoottorin hiukkasjärjestelmää hyödyntäviä tehosteita. Näiden tehosteiden kuormittavuutta piti siten vähentää. Kuormittavuuden vähentämiseksi tehosteista karsittiin kuormituksen kannalta merkittäviä komponentteja. Tämän jälkeen päästiin asetettuun tavoitteeseen, jolloin suoritus kuormitti laitteistoa tasapuolisemmin ja liike näytti kauttaaltaan sulavalta.

Kannettavalla tietokoneella lähtökohdat olivat erilaiset, sillä oletusasetuksilla animaatio ei toistunut lainkaan sulavasti. Sisältöä ei kuitenkaan olisi ollut mielekästä lähteä karsimaan tai suunnittelemaan uudelleen, joten parempaa suoriutumista haettiin skaalautuvuusasetuksilla. Skaalautuvuusasetusten avulla kuvanlaatuun ja mallien laatuun voitiin vaikuttaa dynaamisemmin niitä suoraan muokkaamatta. Profilointikertoja vertaamalla etsittiin tyydyttävä kompromissi kuvanlaadun ja liikkeen sulavuuden välille, jolloin FPS-keskiarvo saatiin nostettua yli tavoitteeksi asetetun 30:n.

Testiprojektien optimoinnissa hyödynnettiin suhteellisen yksinkertaisia keinoja, jotka olivat tässä tapauksessa riittäviä. Työssä tutkittiin kuitenkin myös erikoisempien tekniikoiden tarjoamia mahdollisuuksia, mutta niitä ei kokeiltu käytännössä. Tekniikoihin si-

sältyy esimerkiksi uusi Vulkan-rajapinta, jonka etuihin kuuluu pienempi CPU:n kuormitus sekä parempi rinnakkainen laskenta. Unreal Engine 4:n tuki rajoittuu kuitenkin Vulkan-rajapinnan osalta tällä hetkellä vain mobiilialustoille, joten testaaminen ei ollut mahdollista. Yksi tutkimuskohde voisi myös olla suorituskyvyn vertailu eri käyttöjärjestelmien välillä. Tuloksilla voisi olla merkitystä projekteille, joiden kohdealustoihin kuuluvat myös eri käyttöjärjestelmät.

Työtä tehdessä pelimoottorin versio vaihtui useaan kertaan, mikä voi olla yksi syy siihen, että joissain tilanteissa dokumentaatio vaikutti olevan epätarkka. Pelimoottorin dokumentaatio on kuitenkin yleisellä tasolla hyvää ja riittää käytön opiskeluun itsenäisesti. Kaikki työkalujen tai pelimoottorin ominaisuudet eivät toimineet odotetulla tavalla. Jotkin ominaisuudet vaikuttivat toimivan jopa väärin tai eivät ollenkaan. Pelimoottorin kehittyessä myös pelimoottoriin sisältyvät työkalut todennäköisesti muuttuvat hitaasti, jolloin jotkin työssä nähtävät yksityiskohdat eivät välttämättä enää päde. Myös suoriutuminen saattaa joidenkin ominaisuuksien kohdalla parantua tai heikentyä. Unreal Engine 4:n dokumentaatiosta opitut keskeiset periaatteet ovat kuitenkin edelleen sovellettavissa ympäristöstä riippumatta ja pelimoottoriin sisältyvät työkalut ja ominaisuudet näyttävät tarjoavan hyvät edellytykset pelien optimointiin erilaisille alustoille.

LÄHTEET

[1] Free Software Foundation, Inc.Why do you call it GNU/Linux and not Linux? [WWW], Free Software Foundation, Inc., [Viitattu: 10.2.2016], Saatavissa: https://www.gnu.org/gnu/gnu-linux-faq.html#why

[2] Unreal Engine 4 Documentation: New and Updated Resources [WWW], Epic Games, Inc., [Viitattu: 3.10.2015], Saatavissa: https://docs.unrealengine.com/latest/INT/Updates/index.html

[3] Eberly, David H., 3D Game Engine Design (Second Edition), Elsevier Inc., 2007, 1040 s.

[4] Gregory, Jason, Game Engine Architecture, CRC Press, 2009, 864 s.

[5] M. Bailey, S. Cunningham, Graphics Shaders: Theory and Practice (2nd edition), CRC Press, 2016, 518 s.

[6] Millington, Ian, Game Physics Engine Development: How to Build a Robust Commercial-Grade Physics Engine For Your Game (Second Edition), Morgan Kaufmann Publishers, 2010, 524 s.

[7] Unreal Engine 4 Documentation: Performance and Profiling [WWW], Epic Games, Inc., [Viitattu: 1.10.2015], Saatavissa: https://docs.unrealengine.com/latest/INT/Engine/Performance/index.html

[8] WHAT IS UNREAL ENGINE 4? [WWW], Epic Games, Inc., [Viitattu: 29.9.2015], Saatavissa: https://www.unrealengine.com/what-is-unreal-engine-4

[9] Unreal Engine 4 Documentation: Paper 2D [WWW], Epic Games, Inc., [Viitattu: 30.9.2015], Saatavissa: https://docs.unrealengine.com/latest/INT/Engine/Paper2D/index.html

[10] UNREAL ENGINE FEATURES [WWW], Epic Games, Inc., [Viitattu: 1.10.2015], Saatavissa: https://www.unrealengine.com/unreal-engine-4

[11] Unreal Engine 4 Documentation: Profiler [WWW], Epic Games, Inc., [Viitattu: 13.12.2015], Saatavissa: https://docs.unrealengine.com/latest/INT/Engine/Performance/Profiler/index.html

[12] Unreal Engine 4 Documentation: Physics Simulation [WWW], Epic Games Inc., [Viitattu: 24.10.2015], Saatavissa:

https://docs.unrealengine.com/latest/INT/Engine/Physics/index.html

[13] Unreal Engine 4 Documentation: Tools and Editors [WWW], Epic Games, Inc., [Viitattu: 29.11.2015], Saatavissa: https://docs.unrealengine.com/latest/INT/GettingStarted/SubEditors/index.html

[14] Unreal Engine 4 Documentation: Installing Unreal Engine [WWW], Epic Games Inc., [Viitattu: 22.8.2016], Saatavissa: https://docs.unrealengine.com/latest/INT/GettingStarted/Installation/

[15] FREQUENTLY ASKED QUESTIONS (FAQ) [WWW], Epic Games, Inc., [Viitattu: 29.9.2015], Saatavissa: https://www.unrealengine.com/faq

[16] Linux Support [WWW], , [Viitattu: 30.9.2015], Saatavissa: https://wiki.unrealengine.com/Linux_Support

[17] The GNU General Public License v3.0 [WWW], Free Software Foundation, Inc., [Viitattu: 28.2.2016], Saatavissa: https://www.gnu.org/licenses/gpl-3.0.en.html

[18] What is Copyleft? [WWW], Free Software Foundation, Inc., [Viitattu: 28.2.2016], Saatavissa: https://www.gnu.org/licenses/copyleft.en.html

[19] UNREAL® ENGINE END USER LICENSE AGREEMENT [WWW], Epic Games, Inc., [Viitattu: 30.9.2015], Saatavissa: https://www.unrealengine.com/eula

[20] The MIT License (MIT) [WWW], Open Source Initiative, [Viitattu: 30.9.2015], Saatavissa: https://opensource.org/licenses/MIT

[21] Linux Launcher [REMOVED: backburnered] on UE4 Roadmap [WWW], , [Viitattu: 22.8.2016], Saatavissa: https://trello.com/c/iS3x3Wxw/553-linux-launcher-removed-backburnered

[22] Unreal Engine 4 Documentation: Building Unreal Engine from Source [WWW], Epic Games, Inc., [Viitattu: 22.12.2015], Saatavissa: https://docs.unrealengine.com/latest/INT/Programming/Development/BuildingUnrealEngine/index.html

[23] Building On Linux [WWW], Epic Games, Inc., [Viitattu: 28.10.2015], Saatavissa: https://wiki.unrealengine.com/Building_On_Linux

[24] Unreal Engine 4 Documentation: Actors [WWW], Epic Games, Inc., [Viitattu: 13.12.2015], Saatavissa: https://docs.unrealengine.com/latest/INT/Programming/UnrealArchitecture/Actors/index.html

[25] Unreal Engine 4 Documentation: Level Editor [WWW], Epic Games, Inc., [Viitattu: 13.12.2015], Saatavissa: https://docs.unrealengine.com/latest/INT/Engine/UI/LevelEditor/index.html

[26] Unreal Engine 4 Documentation: Manipulating Actors [WWW], Epic Games Inc., [Viitattu: 29.3.2016], Saatavissa: https://docs.unrealengine.com/latest/INT/GettingStarted/HowTo/ManipulatingActors/index.html

[27] Unreal Engine 4 Documentation: PIE [WWW], Epic Games, Inc., [Viitattu: 2.9.2016], Saatavissa: https://docs.unrealengine.com/latest/INT/GettingStarted/HowTo/PIE/

[28] Unreal Engine 4 Documentation: Launching to Devices [WWW], Epic Games, Inc., [Viitattu: 1.11.2015], Saatavissa: https://docs.unrealengine.com/latest/INT/Engine/Deployment/Launching/index.html

[29] Unreal Engine 4 Documentation: Content Cooking [WWW], Epic Games Inc., [Viitattu: 21.7.2016], Saatavissa: https://docs.unrealengine.com/latest/INT/Engine/Deployment/Cooking/index.html

[30] Unreal Engine 4 Documentation: Packaging Projects [WWW], Epic Games, Inc., [Viitattu: 1.11.2015], Saatavissa: https://docs.unrealengine.com/latest/INT/Engine/Basics/Projects/Packaging/index.html

[31] Unreal Engine 4 Documentation: Introduction to Blueprints [WWW], Epic Games, Inc., [Viitattu: 30.9.2015], Saatavissa: https://docs.unrealengine.com/latest/INT/Engine/Blueprints/GettingStarted/index.html

[32] Unreal Engine 4 Documentation: Blueprint Basic User Guide [WWW], Epic Games Inc., [Viitattu: 1.3.2016], Saatavissa: https://docs.unrealengine.com/latest/INT/Engine/Blueprints/BP_HowTo/BasicUsage/index.html

[33] Unreal Engine 4 Documentation: Blueprint Class [WWW], Epic Games Inc., [Viitattu: 1.3.2016], Saatavissa: https://docs.unrealengine.com/latest/INT/Engine/Blueprints/UserGuide/Types/ClassBlueprint/index.html#parentclasses

[34] Unreal Engine 4 Documentation: Types of Blueprints [WWW], Epic Games, Inc., [Viitattu: 29.11.2015], Saatavissa: https://docs.unrealengine.com/latest/INT/Engine/Blueprints/UserGuide/Types/index.html

[35] Unreal Engine 4 Documentation: Graphs [WWW], Epic Games, Inc., [Viitattu: 17.9.2016], Saatavissa: https://docs.unrealengine.com/latest/INT/Engine/Blueprints/UserGuide/Graphs/index.html

[36] Unreal Engine 4 Documentation: Construction Script [WWW], Epic Games, Inc.,

[Viitattu: 5.12.2015], Saatavissa: https://docs.unrealengine.com/latest/INT/Engine/Blueprints/UserGuide/UserConstructionScript/index.html

[37] Unreal Engine 4 Documentation: Level Blueprint [WWW], Epic Games, Inc., [Viitattu: 13.12.2015], Saatavissa: https://docs.unrealengine.com/latest/INT/Engine/Blueprints/UserGuide/Types/LevelBlueprint/index.html

[38] Unreal Engine 4 Documentation: EventGraph [WWW], Epic Games, Inc., [Viitattu: 5.12.2015], Saatavissa: https://docs.unrealengine.com/latest/INT/Engine/Blueprints/UserGuide/EventGraph/index.html

[39] Unreal Engine 4 Documentation: Events [WWW], Epic Games, Inc., [Viitattu: 6.12.2015], Saatavissa: https://docs.unrealengine.com/latest/INT/Engine/Blueprints/UserGuide/Events/index.html

[40] Unreal Engine 4 Documentation: Introduction to C++ Programming in UE4 [WWW], Epic Games, Inc., [Viitattu: 9.10.2015], Saatavissa: https://docs.unrealengine.com/latest/INT/Programming/Introduction/index.html

[41] Unreal Engine 4 Documentation: Programming Guide [WWW], Epic Games, Inc., [Viitattu: 29.9.2015], Saatavissa: https://www.unrealengine.com/what-is-unreal-engine-4

[42] William Sherif, Learning C++ by Creating Games with UE4, Packt Publishing, 2015, 344 s.

[43] Unreal Engine 4 Documentation: Setting Up Visual Studio for UE4 [WWW], Epic Games, Inc., [Viitattu: 22.12.2015], Saatavissa: https://docs.unrealengine.com/latest/INT/Programming/Development/VisualStudioSetup/index.html

[44] Unreal Engine 4 Documentation: 3 - Write and Compile C++ Code [WWW], Epic Games Inc., [Viitattu: 18.1.2016], Saatavissa: https://docs.unrealengine.com/latest/INT/Programming/QuickStart/3/index.html

[45] Unreal Engine 4 Documentation: Coding Standard [WWW], Epic Games Inc., [Viitattu: 18.1.2016], Saatavissa: https://docs.unrealengine.com/latest/INT/Programming/Development/CodingStandard/index.html

[46] qmake Project Files [WWW], Qt Company Ltd, [Viitattu: 10.2.2016], Saatavissa: https://doc.qt.io/qt-4.8/qmake-project-files.html

[47] Development/Tutorials/KDevelop/Creating a project template [WWW], KDE TechBase, [Viitattu: 16.2.2016], Saatavissa: https://techbase.kde.org/Development/Tutorials/KDevelop/Creating_a_project_template#Template_content_files

[48] CMake Tutorial [WWW], Kitware, [Viitattu: 16.2.2016], Saatavissa: https://cmake.org/cmake-tutorial/

[49] Unreal Engine 4 Documentation: Compiling Game Projects [WWW], Epic Games, Inc., [Viitattu: 1.11.2015], Saatavissa: https://docs.unrealengine.com/latest/INT/Programming/Development/CompilingProjects/index.html

[50] Unreal Engine 4 Documentation: C++ Class Wizard [WWW], Epic Games Inc., [Viitattu: 18.1.2016], Saatavissa: https://docs.unrealengine.com/latest/INT/Programming/Development/ManagingGameCode/CppClassWizard/index.html

[51] Unreal Engine 4 Documentation: Gameplay Classes [WWW], Epic Games Inc., [Viitattu: 16.2.2016], Saatavissa: https://docs.unrealengine.com/latest/INT/Programming/UnrealArchitecture/Reference/Classes/index.html

[52] Unreal Engine 4 Documentation: Performance and Profiling Overview [WWW], Epic Games, Inc., [Viitattu: 24.8.2016], Saatavissa: https://docs.unrealengine.com/latest/INT/Engine/Performance/Overview/index.html

[53] Unreal Engine 4 Documentation: Stat Commands [WWW], Epic Games Inc., [Viitattu: 2.4.2016], Saatavissa: https://docs.unrealengine.com/latest/INT/Engine/UI/LevelEditor/Modes/index.html

[54] Unreal Engine 4 Documentation: CPU Profiling [WWW], Epic Games Inc., [Viitattu: 6.4.2016], Saatavissa: https://docs.unrealengine.com/latest/INT/Engine/Performance/CPU/index.html

[55] Unreal Engine 4 Documentation: GPU Profiling [WWW], Epic Games Inc., [Viitattu: 13.4.2016], Saatavissa: https://docs.unrealengine.com/latest/INT/Engine/Performance/GPU/index.html

[56] Unreal Engine 4 Documentation: Unreal Frontend [WWW], Epic Games Inc., [Viitattu: 20.9.2016], Saatavissa: https://docs.unrealengine.com/latest/INT/Engine/Deployment/UnrealFrontend

[57] Epic Wiki: Running On Linux [WWW], , [Viitattu: 15.8.2016], Saatavissa: https://wiki.unrealengine.com/Running_On_Linux

[58] Unreal Engine 4 Documentation: Scalability Reference [WWW], Epic Games, Inc., [Viitattu: 23.9.2016], Saatavissa: https://docs.unrealengine.com/latest/INT/Engine/Performance/Scalability/ScalabilityReference/index.html

[59] Unreal Engine 4 Documentation: Performance Guidelines for Artists and Designers [WWW], Epic Games, Inc., [Viitattu: 25.8.2016], Saatavissa: https://docs.unrealengine.com/latest/INT/Engine/Performance/Guidelines/index.html

[60] Unreal Engine 4 Documentation: Tessellation Multiplier [WWW], Epic Games, Inc., [Viitattu: 26.9.2016], Saatavissa: https://docs.unrealengine.com/latest/INT/Resources/ContentExamples/MaterialNodes/1_12/

[61] Unreal Engine 4 Documentation: Point Lights [WWW], Epic Games Inc., [Viitattu: 26.9.2016], Saatavissa: https://docs.unrealengine.com/latest/INT/Engine/Rendering/LightingAndShadows/LightTypes/Point/index.html

[62] Unreal Engine 4 Documentation: Core Optimization Concepts for Particle Systems [WWW], Epic Games, Inc., [Viitattu: 26.9.2016], Saatavissa: https://docs.unrealengine.com/latest/INT/Engine/Rendering/ParticleSystems/Optimization/Concepts/index.html

[63] Unreal Engine 4 Documentation: Cascade Particle Systems [WWW], Epic Games Inc., [Viitattu: 1.10.2016], Saatavissa: https://docs.unrealengine.com/latest/INT/Engine/Rendering/ParticleSystems/

[64] Parallel Programming and Computing Platform [WWW], NVIDIA, [Viitattu: 24.10.2015], Saatavissa: https://www.nvidia.com/object/cuda_home_new.html

[65] PhysX FAQ: Why is a GPU good for physics processing? [WWW], NVIDIA, [Viitattu: 28.10.2015], Saatavissa: https://www.nvidia.com/object/physx_faq.html#q5

[66] UE4 AnswerHub: Does Unreal Engine 4 support PhysX GPU acceleration? [WWW], Epic Games, Inc., [Viitattu: 1.10.2016], Saatavissa: https://answers.unrealengine.com/questions/36308/gpu-physx.html

[67] Physics-Based Simulations and Effects - YouTube [WWW], Unreal Engine, [Viitattu: 1.10.2016], Saatavissa: https://www.youtube.com/watch?v=-lFKZRX1K_0

[68] NVIDIA FleX [WWW], NVIDIA Corporation, [Viitattu: 17.11.2015], Saatavissa: https://developer.nvidia.com/flex

[69] Github: NvPhysX/UnrealEngine [WWW], , [Viitattu: 1.10.2016], Saatavissa: https://github.com/NvPhysX/UnrealEngine/tree/FleX

[70] Preliminary PhysX FleX integration into Unreal Engine 4 is available [WWW], PhysXInfo.com, [Viitattu: 1.10.2016], Saatavissa: http://physxinfo.com/news/12540/preliminary-physx-flex-integration-into-unreal-engine-4-is-available/

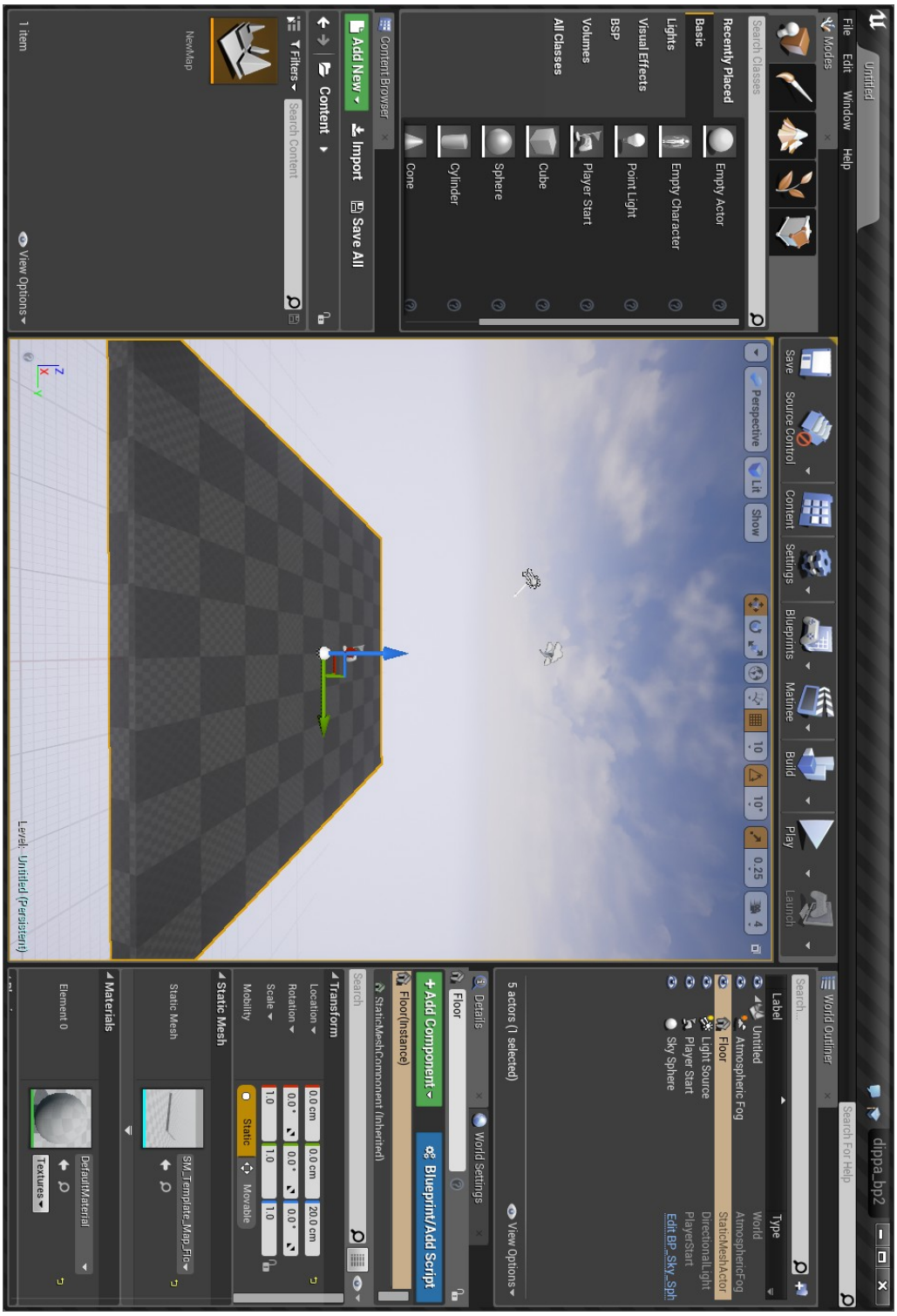
[71] Khronos Releases Vulkan 1.0 Specification [WWW], Khronos Group, [Viitattu: 1.10.2016], Saatavissa: https://www.khronos.org/news/press/khronos-releases-vulkan-1-0-specification

[72] Dana CowleyEpic Games Unveils ProtoStar at Samsung Galaxy Unpacked [WWW], Epic Games Inc., [Viitattu: 1.10.2016], Saatavissa: https://www.unrealengine.com/blog/epic-games-unveils-protostar-at-samsung-galaxy-unpacked

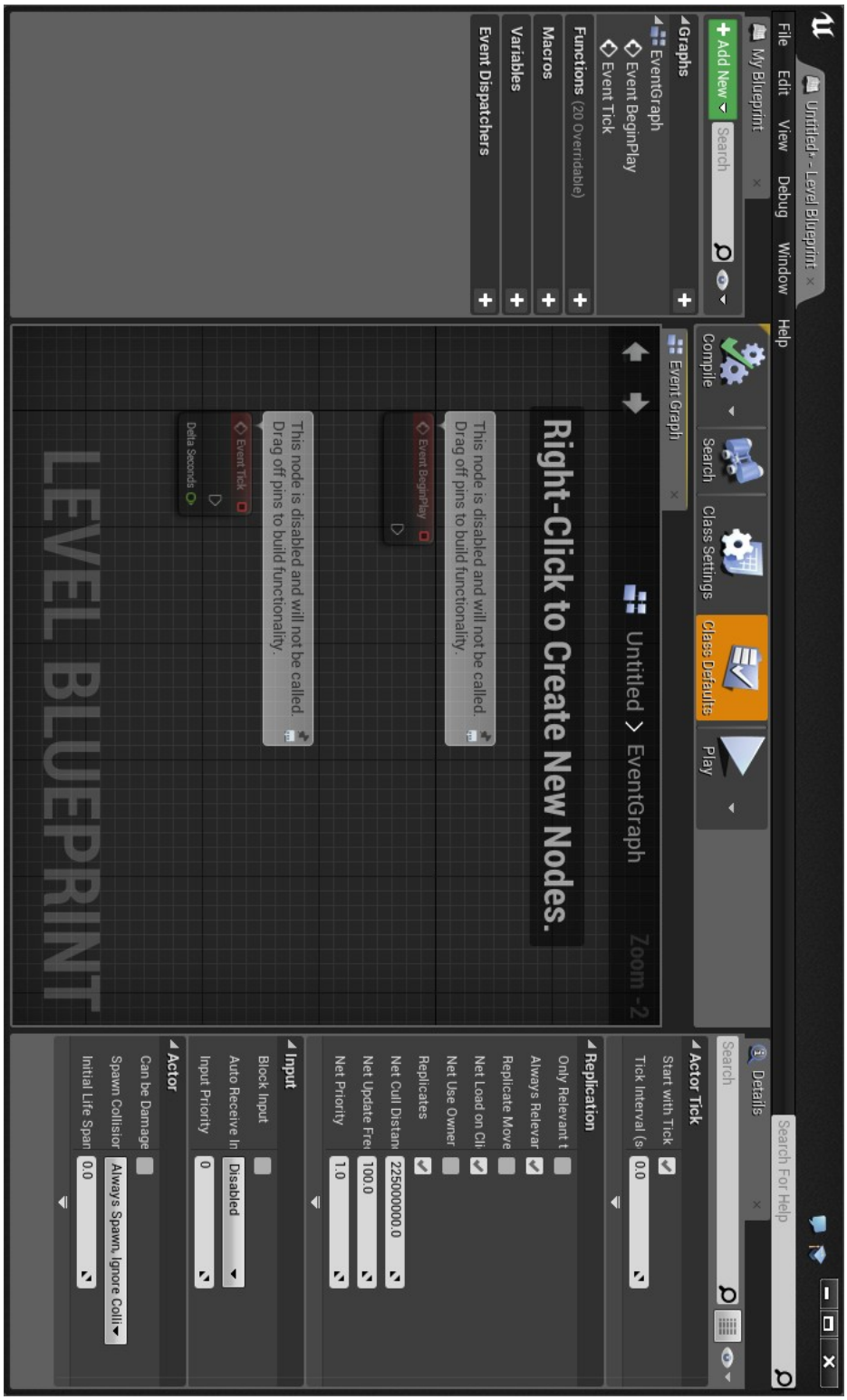
[73] About - Linux Mint [WWW], Linux Mark Institute, [Viitattu: 10.2.2016], Saatavissa: http://www.linuxmint.com/about.php

[74] Unreal Engine 4 Documentation: Profiler Tool Reference [WWW], Epic Games, Inc., [Viitattu: 13.12.2015], Saatavissa: https://docs.unrealengine.com/latest/INT/Engine/Performance/Profiler/index.html

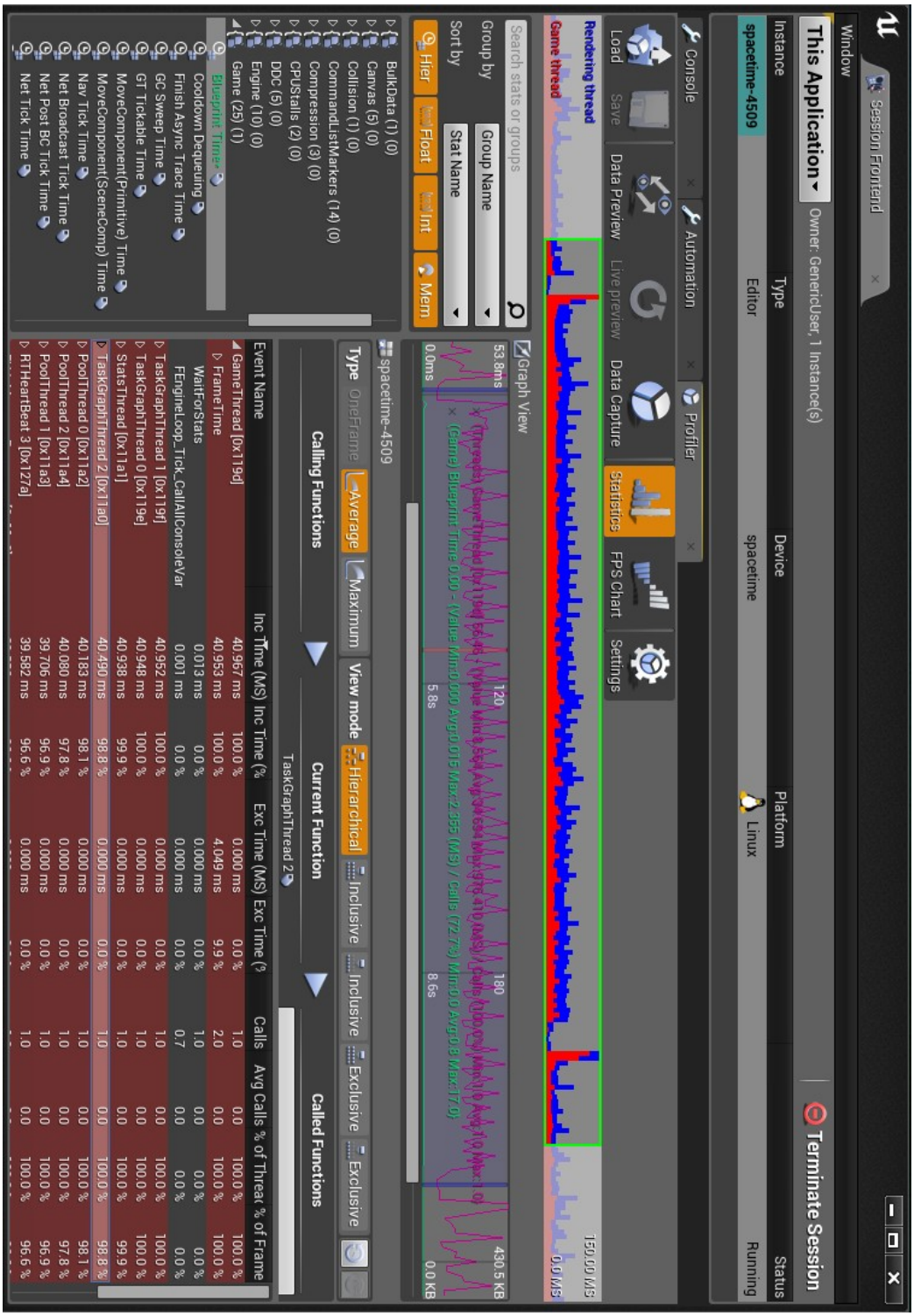
LIITE 1


Untitled
File Edit Window Help
Modes
Search Classes
Recently Placed
Basic
Lights
Visual Effects
BSP
Volumes
All Classes
Empty Actor
Empty Character
Point Light
Player Start
Cube
Sphere
Cylinder
Cone
Content Browser
Add New
Import
Save All
Content
Filters
Search Content
NewMap
1 item
View Options
Save
Source Control
Content
Settings
Blueprints
Matinee
Build
Play
Launch
Perspective
Lit
Show
Level: Untitled (Persistent)
dippa_bp2
Search For Help
World Outliner
Search...
Label
Type
Untitled
World
Atmospheric Fog
AtmosphericFog
Floor
StaticMeshActor
Light Source
DirectionalLight
Player Start
PlayerStart
Sky Sphere
Edit BP_Sky_Sph
5 actors (1 selected)
View Options
Details
World Settings
Floor
+ Add Component
Blueprint/Add Script
Floor(Instance)
StaticMeshComponent (Inherited)
Search
Transform
Location
Rotation
Scale
Mobility
Static
Movable
Static Mesh
Static Mesh
SM_Template_Map_Flo
Materials
Element 0
DefaultMaterial
Textures

LIITE 2


Untitled* - Level Blueprint
File Edit View Debug Window Help
My Blueprint
+ Add New
Search
Graphs
EventGraph
Event BeginPlay
Event Tick
Functions (20 Overridable)
Macros
Variables
Event Dispatchers
Compile
Search
Class Settings
Class Defaults
Play
Event Graph
Untitled > EventGraph
Zoom -2
Right-Click to Create New Nodes.
This node is disabled and will not be called. Drag off pins to build functionality.
Event BeginPlay
This node is disabled and will not be called. Drag off pins to build functionality.
Event Tick
Delta Seconds
LEVEL BLUEPRINT
Search For Help
Details
Search
Actor Tick
Start with Tick
Tick Interval (s
0.0
Replication
Only Relevant t
Always Relevar
Replicate Move
Net Load on Cli
Net Use Owner
Replicates
Net Cull Distan
225000000.0
Net Update Fre
100.0
Net Priority
1.0
Input
Block Input
Auto Receive In
Disabled
Input Priority
0
Actor
Can be Damage
Spawn Collisior
Always Spawn, Ignore Colli
Initial Life Span
0.0

LIITE 3


Session Frontend
Window
This Application
Owner: GenericUser, 1 Instance(s)
Terminate Session
Instance
Type
Device
Platform
Status
spacetime-4509
Editor
spacetime
Linux
Running
Console
Automation
Profiler
Load
Save
Data Preview
Live preview
Data Capture
Statistics
FPS Chart
Settings
Rendering thread
Game thread
150.00 MS
0.0 MS
Search stats or groups
Group by
Group Name
Sort by
Stat Name
Hier
Float
Int
Mem
Graph View
53.8ms
120
180
430.5 KB
0.0ms
5.8s
8.6s
0.0 KB
BulkData (1) (0)
Canvas (5) (0)
Collision (1) (0)
CommandListMarkers (14) (0)
Compression (3) (0)
CPUStalls (2) (0)
DDC (5) (0)
Engine (10) (0)
Game (25) (1)
Blueprint Time
Cooldown Dequeuing
Finish Async Trace Time
GC Sweep Time
GT Tickable Time
MoveComponent(Primitive) Time
MoveComponent(SceneComp) Time
Nav Tick Time
Net Broadcast Tick Time
Net Post BC Tick Time
Net Tick Time
spacetime-4509
Type
OneFrame
Average
Maximum
View mode
Hierarchical
Inclusive
Exclusive
Calling Functions
Current Function
Called Functions
TaskGraphThread 2
Event Name
Inc Time (MS)
Inc Time (%
Exc Time (MS)
Exc Time (%
Calls
Avg Calls
% of Threat
% of Frame
GameThread [0x119d]
40.967 ms
100.0 %
0.000 ms
0.0 %
1.0
0.0
100.0 %
100.0 %
FrameTime
40.953 ms
100.0 %
4.049 ms
9.9 %
2.0
0.0
100.0 %
100.0 %
WaitForStats
0.013 ms
0.0 %
0.000 ms
0.0 %
1.0
0.0
0.0 %
0.0 %
FEngineLoop_Tick_CallAllConsoleVar
0.001 ms
0.0 %
0.000 ms
0.0 %
0.7
0.0
0.0 %
0.0 %
TaskGraphThread 1 [0x119f]
40.952 ms
100.0 %
0.000 ms
0.0 %
1.0
0.0
100.0 %
100.0 %
TaskGraphThread 0 [0x119e]
40.948 ms
100.0 %
0.000 ms
0.0 %
1.0
0.0
100.0 %
100.0 %
StatsThread [0x11a1]
40.938 ms
99.9 %
0.000 ms
0.0 %
1.0
0.0
100.0 %
99.9 %
TaskGraphThread 2 [0x11a0]
40.490 ms
98.8 %
0.000 ms
0.0 %
1.0
0.0
100.0 %
98.8 %
PoolThread 0 [0x11a2]
40.183 ms
98.1 %
0.000 ms
0.0 %
1.0
0.0
100.0 %
98.1 %
PoolThread 2 [0x11a4]
40.080 ms
97.8 %
0.000 ms
0.0 %
1.0
0.0
100.0 %
97.8 %
PoolThread 1 [0x11a3]
39.706 ms
96.9 %
0.000 ms
0.0 %
1.0
0.0
100.0 %
96.9 %
RTHeartBeat 3 [0x127a]
39.582 ms
96.6 %
0.000 ms
0.0 %
1.0
0.0
100.0 %
96.6 %