Opinnäytetyö (AMK)

Tieto- ja viestintätekniikka

2024

Ville Korpiaho

C++- ja Blueprint-ohjelmointikielten suorituskyvyn vertailuanalyysi Unreal Engine 5 -ympäristössä

TURKU AMK
TURKU UNIVERSITY OF APPLIED SCIENCES

Opinnäytetyö (AMK) | Tiivistelmä

Turun ammattikorkeakoulu

Tieto- ja viestintätekniikka

2024 | 29 sivua

Ville Korpiaho

C++- ja Blueprint-ohjelmointikielten suorituskyvyn vertailuanalyysi Unreal Engine 5 -ympäristössä

Pelikehittäjien keskuudessa on ollut epäselvyyttä kumpaa ohjelmointikieltä C++:a vai Blueprintiä tulisi käyttää Unreal Engine 5 -pelimoottorilla ja kumpi on tehokkaampi suorituskyvyltään. Opinnäytetyön tavoitteena oli vertailla näiden suorituskykyä ja soveltuvuutta videopelien kehittämisessä. Työn tarkoituksena oli tuottaa konkreettisia tuloksia kehittämällä kaksi identtistä peliprojektia kummallakin ohjelmointikielillä ja vertailemalla niiden suoritustehokkuutta.

Tutkimuksessa luotiin kaksi projektia, joista toinen tehtiin C++:lla ja toinen Blueprintillä. Projektien suorituskykyä arvioitiin mittaamalla erityisesti kolmea elementtiä: objektien dynaamista luontia, matemaattisten operaatioiden suorituskykyä ja hahmojen kiertokulman ja sijainnin muuttamista. Suorituskyvyn mittaamiseen käytettiin Unreal Insights -työkalua, joka mahdollisti yksityiskohtaisten aikatietojen keräämisen ja analysoinnin.

Tutkimustulokset osoittivat, että C++ on tehokkaampi vaihtoehto suurien objektimäärien käsittelyssä, matemaattisissa laskutoimituksissa ja hahmojen liikkeen hallinnassa. Blueprintin etuina ovat kuitenkin nopeus ja helppous kehityksessä, mikä tekee siitä sopivamman vähemmän vaativiin projekteihin tai tilanteisiin, joissa nopea iterointi on tärkeää. Näiden tulosten perusteella voidaan suositella C++:n käyttöä suorituskykykriittisissä projekteissa, kun taas Blueprint sopii hyvin nopeisiin prototyyppeihin ja vähemmän monimutkaisiin peliprojekteihin.

Asiasanat: Unreal Engine 5, C++, Blueprint, pelikehitys, suorituskyky, ohjelmointi

Bachelor’s Thesis | Abstract

Turku University of Applied Sciences

Information and Communications Technology

2024 | 29

Ville Korpiaho

Performance Comparison Analysis of C++ and Blueprint Programming Languages in the Unreal Engine 5 Environment

This thesis explores the performance differences and overall suitability of C++ and Blueprint programming in Unreal Engine 5 for game development. Game developers often debate which approach offers better performance and efficiency, so this study set out to generate concrete data by building two nearly identical game projects, one using C++ and the other using Blueprint and then compare their performance results.

The comparison focused on three main performance aspects: dynamic object creation, math operations, and character movement adjustments. Unreal Insights was utilized for gathering and analysing timing data because this tool provided detailed breakdowns of how each programming method performed in these areas.

The results demonstrated that C++ generally performs better for handling several objects, complex calculations, and detailed character movements. On the other hand, Blueprint proved easier and faster to use, making it a better fit for simpler projects or situations where fast iteration is needed. Overall, C++ seems to be the best choice for projects where performance is key, while Blueprint is great for prototyping or less demanding games.

Keywords: Unreal Engine 5, C++, Blueprint, game development, performance, programming

Sisältö

Kuvat

Käytetyt lyhenteet ja sanasto

3D	3D tarkoittaa kolmiulotteista.
Algoritmi	Algoritmi on ohjeistus tai sääntöjen kokoelma, joka määrittää kuinka tietty tehtävä ratkaistaan.
Blueprint	Blueprint on Unreal Engine -pelimoottorissa käytetty visuaalinen skriptauskieli. (Epic Games 2024a)
C++	C++ on monikäyttöinen ohjelmointikieli. (iD Tech 2020)
FPS	FPS eli frames per second kertoo, kuinka monta kuvaa piirretään näytölle sekunnissa. Mitä enemmän niin sitä parempi ja sulavampi pelikokemus on.
For-silmukka	For-silmukka on ohjelmointirakenne, joka mahdollistaa koodin suorittamisen moneen kertaan samanaikaisesti.
Objekti	Object tarkoittaa lähes mitä vain asiaa pelimaailmassa. Esimerkiksi valoa, hahmoa tai esineitä. (OpenAI 2024)
Päivitys	Päivitys tarkoittaa koodin tai Blueprint skriptin suorittamista säännöllisin välein, yleensä jokaisella FPS-päivityksellä. (Epic Games 2024b)
Skeletal Mesh	Skeletal mesh on 3D-mallinnus, joka käyttää luita ja jota on mahdollista animoida. (Epic Games 2024c)
Solmu	Solmu on visuaalisessa ohjelmoinnissa käytetty elementti, joka on yleensä neliön mallinen. Perinteiseen ohjelmointiin rinnastettuna se vastaa riviä tai useampaa riviä koodia. Esimerkiksi yhtä funktiota tai matemaattista operaattoria. (Epic Games 2024a)
Visuaaliskriptauskieli	Visuaaliskriptauskieli tarkoittaa ohjelmointikieltä, joka käyttää visuaalisia solmu palikoita perinteisen kooditekstin sijaan. (Unity n.d.)

1 Johdanto

Pelikehittäjien keskuudessa on ollut epäselvyyttä kumpaa ohjelmointikieltä C++:a vai Blueprintiä tulisi käyttää Unreal Engine 5 -pelimoottorilla. Molemmat ohjelmointikielet ovat suosittuja, mutta niiden vaikutusta projektien suorituskykyyn ja kehitysprosessin tehokkuuteen ei ole kattavasti tutkittu. Opinnäytetyö vertailee Unreal Engine 5:n C++ ja Blueprint ohjelmointikielien suorituskykyä ja kehitysprosessin haasteita videopelin kehittämisessä.

Kumpikin ohjelmointitapa tarjoaa omat etunsa: C++ mahdollistaa syvällisemmän hallinnan ja optimoinnin, kun taas Blueprint tarjoaa visuaalisen ja käyttäjäystävällisen lähestymistavan. Keskustelua kuitenkin käydään siitä, kumpi näistä tavoista soveltuu paremmin erilaisiin peliprojekteihin, ja pelikehittäjät kaipaavat selkeitä suosituksia perusteltujen vertailujen pohjalta. Tämän tutkimuksen tavoitteena on paitsi arvioida kahden kielen teknistä suorituskykyä, mutta myös niiden käytännön soveltuvuutta peliprojektien toteutuksessa. Tämä tutkimus pyrkii tarjoamaan selkeitä tuloksia, jotka perustuvat molempien ohjelmointikielten suorituskykyä koskevien testitulosten analysointiin.

Tutkimus koostuu teoreettisesta viitekehyksestä, metodologiasta, pelin kehityksestä, suorituskyvyn vertailusta ja kehitysprosessin haasteiden arvioinnista. Tavoitteena on antaa kattava kuva siitä, miten C++ ja Blueprint eroavat toisistaan suoristuskyvyn ja kehitystyön näkökulmasta ja siten auttaa kehittäjiä valitsemaan oikean työkalun tuleviin projekteihinsa.

Aiheesta on hyvin vähän tehty tutkimuksia, koska monet artikkelit ja lausunnot perustuvat mielikuvaan ennemmin kuin tieteelliseen tutkimukseen.

2 C++ ja Blueprintit Unreal Engine 5 -pelimoottorilla

2.1 Pelimoottorin määritelmä

Pelimoottori on erityisesti videopelien kehitykseen suunniteltu ohjelmisto, joka tarjoaa kehittäjille valmiin ja tehokkaan pohjan pelin luomiseen. Sen avulla kehitysprosessi nopeutuu, sillä se sisältää valmiina keskeiset ominaisuudet, kuten fysiikkamoottorin, grafiikan renderöinnin ja tekoälytoiminnot. Kehittäjien ei tarvitse luoda jokaista toimintoa alusta asti, mikä säästää merkittävästi aikaa ja resursseja. Pelimoottorit on yleensä suunniteltu konsoli-, tietokone- tai mobiilipelien kehitykseen. (Phil, 2024.)

Pelimoottorin tehtävä on tarjota pelikehittäjille tarvittavat työkalut pelin luomiseen (Lee, Doran & Misra, 2016). Ennen pelimoottoreiden käyttöön tuloa pelit ohjelmoitiin yksilöllisesti, ja toiminnot, kuten renderöinti ja fysiikkasimulaatio, piti luoda aina uudelleen. Tämä ongelma ratkaistiin luomalla pelimoottoreita, joihin kerättiin suuri määrä tarpeellisia kirjastoja ja edellä mainittuja ominaisuuksia, jolloin kehittäjät pystyivät kätevästi uusiokäyttämään niitä uusissa peleissä. (Phil, 2024.)

Nykyään on olemassa pelimoottoreita, joita tavallisetkin ihmiset voivat käyttää ilmaiseksi, kuten Unreal Engine, Unity ja Godot. Nämä pelimoottorit tarjoavat laajat työkalupakit, jotka mahdollistavat erilaisten pelien kehittämisen ilman suuria taloudellisia investointeja. Esimerkiksi Unity on tunnettu käyttäjäystävällisyydestään ja laajasta dokumentaatiostaan, mikä tekee siitä suositun valinnan niin aloittelijoiden kuin kokeneiden kehittäjien keskuudessa. Unreal Engine puolestaan tarjoaa huipputason grafiikkaominaisuuksia ja on erityisesti suosittu AAA-pelien kehittämisessä, mutta se sopii myös pienemmille projekteille, koska sen käyttö on ilmaista pienituloisille kehittäjille. Godot on avoimen lähdekoodin moottori, joka on erityisen houkutteleva niille, jotka haluavat hallita kehitysprosessia täysin omilla ehdoillaan. (OpenAI, 2024.)

2.2 Unreal Engine 5 -pelimoottori

Unreal Engine on Epic Gamesin kehittämä pelimoottori, joka on yksi markkinoiden johtavimmista pelimoottoreista. Se tukee laajaa valikoimaa kehitysalustoja aina tietokoneista konsoleihin, kuten PS4, XBOX One ja Nintendo Switch. Unreal Engine 5 -pelimoottori mahdollistaa pelikehittäjien hyödyntää sekä C++ -ohjelmointikieltä, että visuaalista Blueprint -ohjelmointikieltä, jotka tarjoavat joustavuutta sekä kokeneille ohjelmistosuunnittelijoille että aloittelijoille. Unreal Engine on tunnettu tehokkaista työkaluistaan, jotka ovat käytettävissä materiaalien ja animaatioiden luomiseen. Näiden lisäksi moottorin käyttö on maksutonta, mutta Epic Games perii rojalteja pelin tuotoista. Tämä tekee Unreal Enginestä houkuttelevan vaihtoehdon pelinkehittäjille maailman laajuisesti. (Denham, 2019.)

Unreal Engine 5 edustaa viimeisintä kehitystä Epic Gamesin pitkäaikaisessa pyrkimyksessä tarjota huippuluokan pelimoottori sekä indie-kehittäjille, että AAA-pelien tuottajille. Tämän sukupolven pelimoottori tuo mukanaan merkittäviä parannuksia grafiikkaan, suorituskykyyn ja käyttäjäystävällisyyteen. (Epic Games, n.d.b)

Kuva 1. Unreal 1998 videopeli (Honkala, 1998).

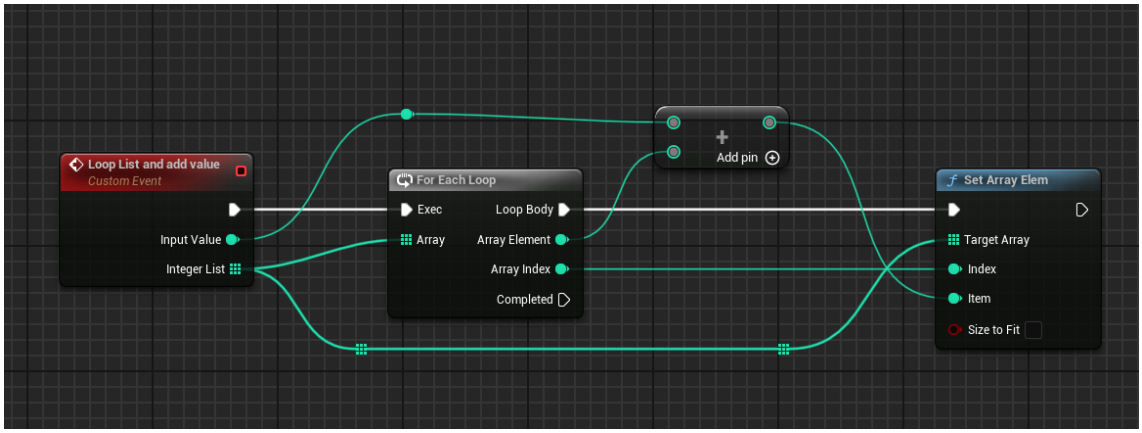
Unreal Engine kehitettiin alun perin Unreal-tietokonepeliä varten, joka julkaistiin vuonna 1998 (kuva 1). Tämän jälkeen Unreal-pelisarjan uusien osien kehityksen myötä myös Unreal Engineen lisättiin jatkuvasti uusia ominaisuuksia, kuten verkkopeliominaisuudet Unreal Tournament 2004 -videopelin yhteydessä. Vuonna 2014 julkaistu Unreal Engine 4 toi mukanaan Blueprint-järjestelmän, jonka avulla peliohjelmointia voidaan toteuttaa visuaalisesti ilman koodin kirjoittamista. Samalla Unreal Engine siirtyi käyttämään C++-ohjelmointikieltä, joka korvasi aiemmin käytössä olleen vanhentuneen Unreal Script -kielen (Lee, Doran & Misra, 2016). Nykyään Unreal Engine 5 edustaa Epic Gamesin pelimoottoreiden uusinta kehitystä. (Epic Games, n.d.b)

2.3 Blueprint-ohjelmointikieli

Blueprint on Unreal Editorille kehitetty edistynyt visuaalinen skriptausjärjestelmä, ja sen intuitiivesella käyttöliittymällä on mahdollisuus luoda skriptitapahtumia ilman perinteisen koodin kirjoittamista. Järjestelmä on suunniteltu erityisesti helpottamaan erilaisten käyttäjien pelinkehitysprosessia, mukaan lukien tasosuunnittelijat, taiteilijat sekä ohjelmointia osaamattomat henkilöt. Blueprint mahdollistaa nopean pelimekaniikan suunnittelun ja iteroinnin, ja sen avulla voidaan jopa kehittää kokonaisia pelejä. Tämä tekee siitä erityisen arvokkaan työkalun Unreal Engine -pelimoottorille, koska se poistaa teknisen osaamisen esteitä ja edistäen luovaa ilmaisua pelisuunnittelussa. Blueprintin ansioista pelikehityksen iterointiprosessi nopeutuu, mikä tukee moniammatillista yhteistyötä ja innovaatiota pelialla. (Valcasara, 2015, s.2.)

Vaikka Blueprint on suunniteltu helpottamaan pelinkehitystä visuaalisen käyttöliittymänsä avulla, se ei täysin poista tarvetta ohjelmointiosaamiselle. Blueprint-järjestelmä sisältää monia perinteisen ohjelmoinnin keskeisiä elementtejä, kuten muuttujat, funktiot, ehdolliset lausekkeet ja silmukat.

Käyttäjän on ymmärrettävä, kuinka tietoa tallennetaan ja käsitellään muuttujien avulla, miten funktioiden avulla voidaan järjestää toistuvia toimintoja ja kuinka ehdollisia lausekkeita käytetään päätöksentekoon. Lisäksi algoritmien ja loogisten riippuvuuksien suunnittelu vaatii samanlaista ajattelutapaa kuin perinteisessä ohjelmoinnissa. Visuaalinen esitystapa voi helpottaa näiden käsitteiden omaksumista, mutta se ei korvaa niitä. Näin ollen Blueprint ei poista ohjelmointiajattelun tarvetta, vaan tarjoaa visuaalisen lähestymistavan sen toteuttamiseen. (OpenAI, 2024.)



Kuva 2. Esimerkki Blueprint-koodista.

Kuva 2 esittää esimerkin Unreal Engine 5 Blueprint-koodista, jossa tapahtuman avulla voidaan lisätä käyttäjän määrittämä arvo jokaisen listan jäsenen määrään. Blueprint on visuaalinen ohjelmointikieli, jonka rakenne muistuttaa perinteisiä ohjelmointikieliä, kuten C++, mutta merkittävä ero on sen solmupohjainen käyttöliittymä tekstipohjaisen koodin sijaan. Solmu-pohjaisuus tarkoittaa, että koodi rakennetaan yhdistämällä erilaisia visuaalisia solmuja, joista jokainen edustaa tiettyä toimintoa tai ehtoa. Tämä mahdollistaa koodin luomisen ja lukemisen ilman varsinaista kirjoitettua koodia, mikä helpottaa ohjelmoinnin oppimista ja vähentää syntaksivirheiden mahdollisuutta.

2.4 C++ -ohjelmointikieli

C++ on matalan tason ohjelmointikieli, joka mahdollistaa laajan valikoiman ohjelmointitehtäviä ja soveltuu monenlaisiin käyttötarkoituksiin. Sen opiskelu mahdollistaa kompleksisten ongelmien ratkaisun ja antaa syvällisen ymmärryksen ohjelmien toimintaperiaatteista. C++:n avulla Unreal Engine -kehittäjät voivat toteuttaa vaativia toiminnallisuuksia ja optimoida suorituskykyä, mikä on erityisen tärkeää suurten ja monimutkaisten peliprojektien yhteydessä. Tämän kielen oppiminen ei ainoastaan laajenna kehittäjän teknistä osaamista vaan myös syventää käsitystä ohjelmistokehityksen perusteista. (iD Tech. 2020.)

Yhtenä etuna C++:n käyttämisessä pelimekaniikan luomisessa on suurempi joustavuus pelin toiminnallisuuksien mukauttamisessa. Toisin sanoen C++ antaa pääsyn koko pelin koodiin, toisin kuin Blueprint, joka rajoittuu vain tiettyihin osiin. Tämä mahdollistaa kehittäjille syvällisemmän hallinnan pelin ydinmekaniikoiden ja -toimintojen yli, avaten ovet monipuolisemmille ja kohdennetuille ratkaisuille pelisuunnittelussa. (iD Tech. 2020.)

```
std::vector<int> addToElements(const std::vector<int>& intArray, int value) {
    // Luodaan uusi lista, johon lisätään muokatut arvot
    std::vector<int> result;

    // Käydään läpi jokainen elementti intArray:sta
    for (int element : intArray) {
        // Lisätään elementtiin value ja tallennetaan tulokseen
        result.push_back(element + value);
    }

    return result;
}
```

Kuva 3. Esimerkki C++ koodista.

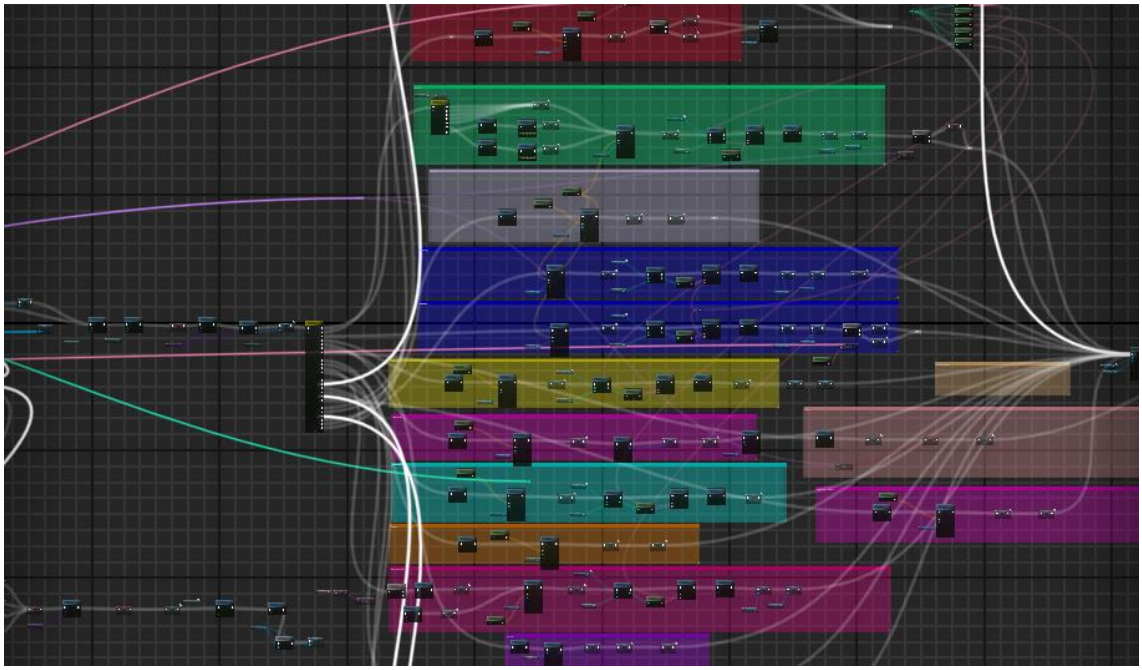
Kuva 3 osoittaa esimerkin Unreal Enginessä käytetystä C++-koodista, jossa tapahtuman avulla lisätään käyttäjän määrittämä arvo jokaisen listan jäsenen määrään. Toisin kuin visuaalinen Blueprint, C++ on tekstipohjainen ohjelmointikieli, jossa logiikka ja toiminnallisuus kirjoitetaan suorana koodina. Tämä tekstipohjainen lähestymistapa mahdollistaa koodin tarkemman hallinnan

ja täyden pääsyn Unreal Enginen ohjelmointirajapintoihin. C++-koodin käyttö antaa kehittäjälle enemmän joustavuutta ja optimointimahdollisuuksia, erityisesti resursseja vaativissa tehtävissä. C++-koodi rakennetaan kirjoittamalla yksittäisiä komentoja ja ehtolausekkeita, mikä vaatii tarkkaa syntaksia ja huolellisuutta, mutta tarjoaa syvällisemmän pääsyn ohjelman toiminnallisuuksiin. Tämä tekee C++:sta tehokkaan työkalun kokeneille ohjelmoijille, jotka haluavat maksimoida suorituskyvyn ja hallita sovelluksen sisäisiä prosesseja tarkasti. C++-koodin kirjoittaminen vaatii ohjelmointitaustaa ja voi lisätä syntaksivirheiden mahdollisuutta.

2.5 Blueprintin ja C++:n erovaisuudet

C++:n ja Blueprintin välillä valintaan liittyvät erot ja niiden vaikutus pelikehityksessä ovat merkittäviä. C++:lla toteutetut pelimekaniikat ovat yleensä tehokkaampia optimoinnin kannalta, kuin Blueprint-versiot, mikä voi olla huomattavaa erityisesti edistyneemmissä peliprojekteissa, vaikka ero ei olisikaan ilmeinen aloittelevien pelinkehittäjien projekteissa. Tämä nopeusero on tärkeä tekijä, joka kannattaa ottaa huomioon pelikehityspolulla edetessä. Toisaalta C++:n käyttöön liittyy myös haasteita, kuten suurempi virheiden tekemisen riski pelimekaniikkoja luotaessa. Vaikka useimmat virheet ovat todennäköisesti pieniä, joitakin voi olla vaikeampi jäljittää, ja kriittiset virheet saattavat jopa kaataa Unreal Engine -moottorin. (iD Tech. 2020.)

Blueprintillä on omat etunsa, kuten nopeus uusien pelimekaniikkojen luomisessa hyödyntämällä valmiita solmuja. Solmujen luominen ja yhdistäminen vie usein vähemmän aikaa, kuin koodin kirjoittaminen ja kääntäminen C++:lla. Vaikka Blueprint voi olla nopeampi useimpien toimintojen luomisessa, sen käyttö voi myös johtaa sekavampaan lopputulokseen.



Kuva 4. Esimerkki sekavasta Blueprintistä.

Monimutkaisten pelimekaniikkojen toteuttaminen Blueprintillä voi aiheuttaa vaikeasti hallittavan solmujen ja yhteyksien viidakon, mikä vaikeuttaa pelimekaniikkojen sijainnin määrittämistä tai vikojen selvittämistä, jos mekaniikka ei toimi odotetusti. (iD Tech. 2020.)

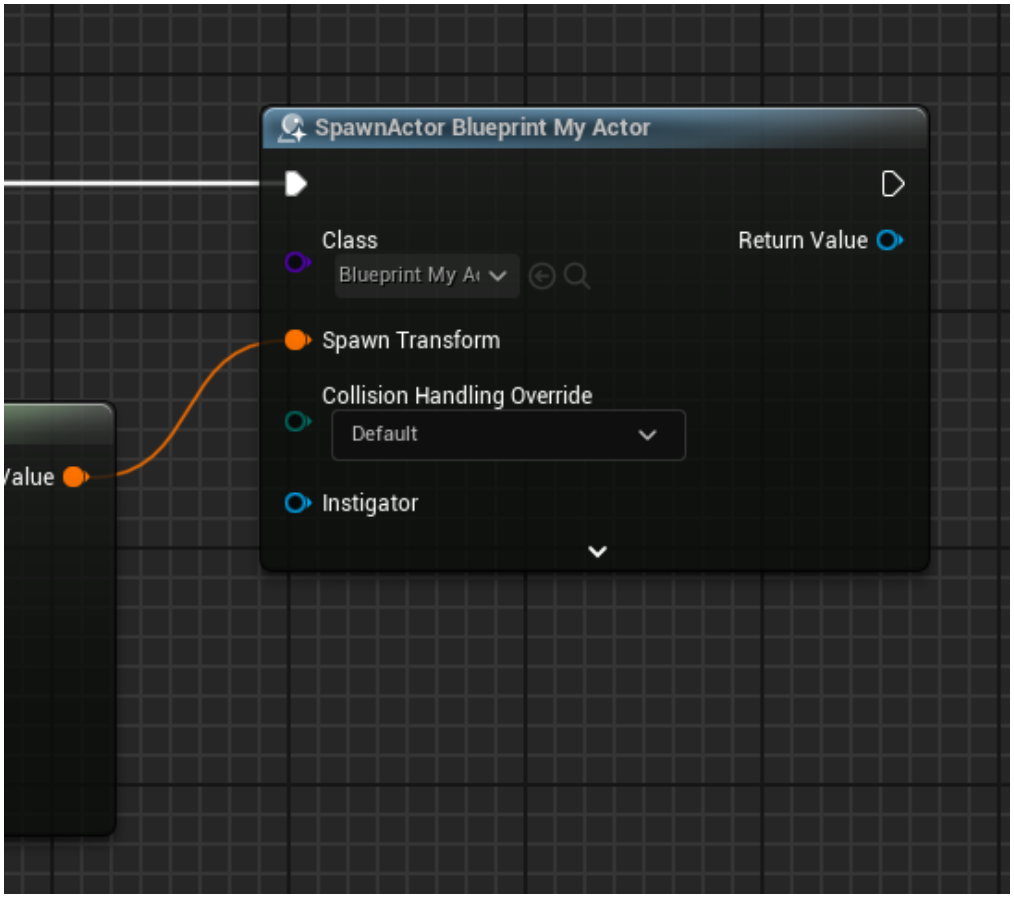
Yhteenvetona voidaan todeta, että sekä C++ että Blueprint tarjoavat omat ainutlaatuiset etunsa ja haasteensa pelinkehityksessä. Valinta näiden kahden välillä riippuu kehitystiimin osaamisesta, projektin vaatimuksista ja käytettävissä olevasta ajasta. Blueprintin helppokäyttöisyys ja nopeus voivat auttaa etenkin prototyyppien kehityksessä ja nopeissa iteraatioissa, kun taas C++ tuo mukanaan tehokkaammat optimointimahdollisuudet sekä laajemmat työkalut, joita tarvitaan erityisesti laajoissa ja suorituskykykriittisissä sovelluksissa. Molempien työkalujen yhdistäminen on monessa projektissa varteenotettava ratkaisu, sillä näin saadaan sekä joustavuutta että tarkkuutta kehitykseen. (Program Ace, 2024.)

3 Suorituskykyvertailu tutkimuksen suorittaminen

Tässä tutkimuksessa suoritetaan kaksi erillistä testiprojektia käyttäen Unreal Engine 5 -pelimoottoria vertailun alustana. Ensimmäinen projekti toteutetaan hyödyntäen Blueprint-visuaaliskriptauskieltä, kun taas toinen projekti kehitetään käyttäen perinteistä C++ ohjelmointikieltä. Tarkastelun kohteena on kolme erityistä elementtiä: ensimmäisenä objektien dynaaminen luonti peliympäristöön, toisena matemaattisen operaatioiden suorittaminen, ja kolmantena testiaiheena tutkitaan objektien kiertokulman ja sijainnin muuttaminen. Näiden elementtien suorituskykyä mitataan ja verrataan kummassakin projektitoteutuksessa suorittamalla identtiset testitapaukset. Suorituskyvyn mittauksessa käytetään Unreal Insights -työkalua, joka mahdollistaa yksityiskohtaisten tietojen keräämisen suorituskyvyn analysointia varten, keskittyen erityisesti kunkin funktion suoritusaikaan millisekunteina. Tämän tutkimuksen tavoitteena on selvittää, kuinka Blueprint-visuaaliskriptauskieltä ja perinteisen C++-ohjelmointikielen käyttö vaikuttaa suorituskykyyn Unreal Engine 5 -pelimoottorissa toteutetuissa projekteissa.

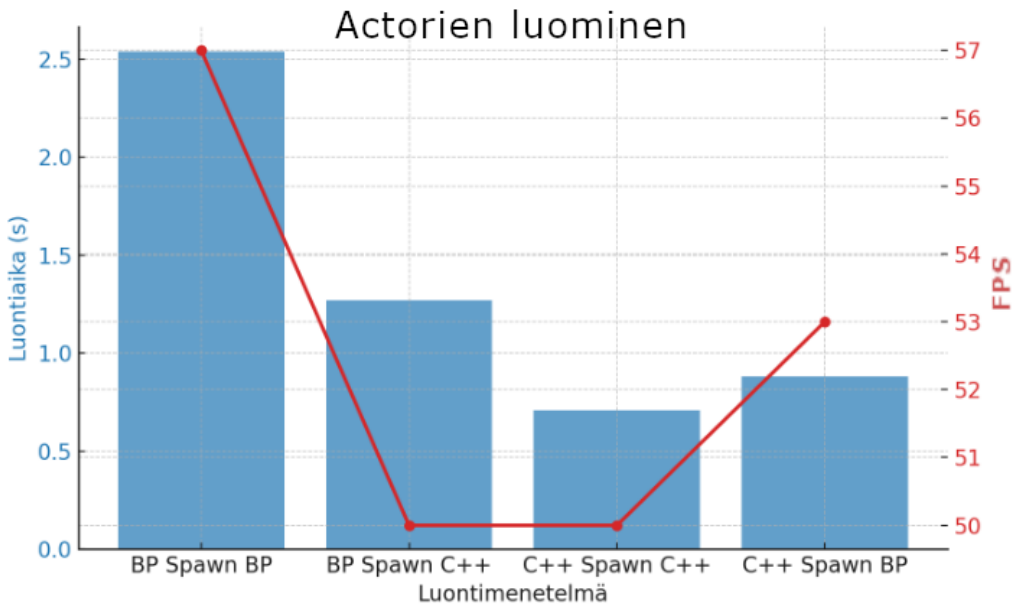
3.1 Dynaaminen luonti

Tässä testivaiheessa keskityttiin vertailemaan C++:n ja Blueprint-menetelmien suorituskykyä objektien luomisessa Unreal Engine 5 -ympäristössä. Dynaaminen luonti toteutettiin molemmilla menetelmillä käyttäen identtisiä lähtöasetuksia. Testissä pelimaailmaan luotiin 1 000, 5 000, 10 000 ja 50 000 objektia Spawn Actor -nimisellä funktiolla, joka luo pelimaailmaan uuden objektin.



Kuva 5. Blueprintillä dynaaminen luonti.

Tutkimuksessa suoritettiin neljä erilaista tilannetta: Blueprint-koodilla Blueprint-objektin tuonti maailmaan, Blueprint-koodilla C++-objektin tuonti maailmaan, C++-koodilla Blueprint-objektin tuonti maailmaan ja C++-koodilla C++-objektin tuonti maailmaan. Testauksen aikana mitattiin aika, joka kului kaikkien objektien tuomiseen pelimaailmaan, sekä se, kuinka paljon tämä vaikutti pelin FPS:ään objektien luomisen jälkeen. Ajan mittaamiseen käytettiin Unreal Insights -työkalua.



Kuva 6. Objektien luomiseen kulunut aika.

Kaaviossa (Kuva 6) esitetään aika, joka kului objektien luomiseen sekä FPS-lukema luomisen jälkeen, kun jokaisella menetelmällä luotiin 50 000 objektia. Tuloksista käy ilmi, että C++ on huomattavasti nopeampi menetelmä kuin Blueprint sekä luontiajan että FPS:n osalta. Sininen pylväs kuvaa aikaa sekunneissa, joka kului objektien luomiseen. Punainen viiva osoittaa luomisen jälkeistä FPS-lukemaa. Testissä havaittiin myös, että merkittävä tekijä on, kummalla koodikielellä toteutettu objekti luodaan.

On kuitenkin tärkeää huomioida, että suorituskyvyn erot tulivat esiin vasta, kun pelimaailmaan tuotiin suuri määrä objekteja. Tyypillisissä peliskenaarioissa, joissa objekteja luodaan kerrallaan vain vähän, kuten sotapeleissä, joissa pelimaailmaan luodaan esim. yksi luoti kerrallaan, Blueprintin ja C++:n välinen suorituskykyero on käytännössä merkityksetön. Tällaisissa tilanteissa molemmat menetelmät toimivat riittävän nopeasti ilman huomattavaa vaikutusta pelin suorituskykyyn. Jos kuitenkin tavoitteena on luoda tuhansia objekteja samanaikaisesti, kuten esimerkiksi massiivisissa taistelukohtauksissa tai monimutkaisissa simulaatioissa, on suositeltavaa käyttää C++-ohjelmointikieltä sekä luontitoimintoon että itse objekteihin. Tämä varmistaa, että pelin

suorituskyky pysyy vakaana myös vaativissa tilanteissa, joissa resurssien tehokas hallinta on kriittistä.

3.2 Matemaattisten operaatioiden testaaminen

3.2.1 Testin suorittaminen

Testausvaiheessa vertailtiin C++:n ja Blueprintin suorituskykyä matemaattisten laskutoimitusten osalta Unreal Engine 5 -ympäristössä. Suorituskykyä mitattiin suorittamalla useita matemaattisia operaatioita, kuten yhteenlaskuja, vähennyslaskuja, kertolaskuja ja jakolaskuja eri määrillä numeropareja 100, 1000, 10 000 ja 100 000. Testissä käytettiin for-silmukka operaatiota tässä. Testissä käytettiin samoja algoritmeja ja rakenteita molemmilla menetelmillä.

Matemaattisten operaatioiden testi on tärkeä, koska monet pelimoottorin toiminnot hyödyntävät matemaattisia operaatioita. Videopelien fysiikat, animaatiot ja hahmojen reitinhaku ovat esimerkkejä operaatioista, jotka tarvitsevat hyvin optimoitua ja tehokasta matemaattista laskentaa. Edellä mainittujen tehtävien nopea suorittaminen on ratkaisevaa pelin suorituskyvyn ja käyttäjäkokemuksen kannalta.

3.2.2 Testitulokset

Testissä C++ oli jopa yli satakertaa nopeampi ja tehokkaampi kuin Blueprint toteutus. Tämä suoritusero on merkittävä tekijä pelikehittäjille, sillä suurilaskentateho mahdollistaa monimutkaisempia ja käyttäjäystävällisempiä ratkaisuja.

Iteraatiot	C++	Blueprint
For loop 100	0.0051 ms	0.467 ms
For loop 1000	0.04 ms	4.5 ms
For loop 10000	0.39 ms	44.2 ms
For loop 100000	4 ms	439 ms

Kuva 7. Lista millisekuntiajoista eri iteraatiomäärillä.

Tulokset osoittavat merkittäviä eroja suorituskyvyssä C++:n ja Blueprintin välillä. C++-toteutus suoriutui huomattavasti nopeammin kaikissa testatuissa määrissä. Esimerkiksi 100 laskutoimitusta suoritettiin C++:lla vain 0,005 ms:ssa, kun taas Blueprint-toteutuksessa vastaava aika oli 0,467 millisekuntia. Suorituskykyero kasvoi testissä suurempiin datamääriin, kuten 100 000 laskutoimitukseen, jossa C++:n suoritusaika oli vain 4 ms verrattuna Blueprint-toteutuksen 439 ms:iin. Nämä tulokset vahvistavat sen, että C++ tarjoaa selvästi paremman suorituskyvyn matemaattisissa operaatioissa Unreal Enginen Blueprint-toteutuksiin verrattuna, erityisesti suurilla datamäärillä.

Tämä ero johtuu osittain siitä, että C++-koodi on kompiloitua ja optimoitua suoritettavaksi suoraan prosessorilla, kun taas Blueprint toimiii visuaalisesti tulkitulla skriptikielellä, mikä lisää suoritusaikaa erityisesti monimutkaisemmissa tai suuria laskentatehtäviä vaativissa skenaarioissa. Tämän perusteella C++ on selkeästi tehokkaampi valinta matemaattisissa laskutoimituksissa, erityisesti silloin, kun laskentatehtävien määrä kasvaa merkittävästi. (OpenAI, 2024.)

3.3 Hahmon kiertokulman ja sijainnin muuttaminen

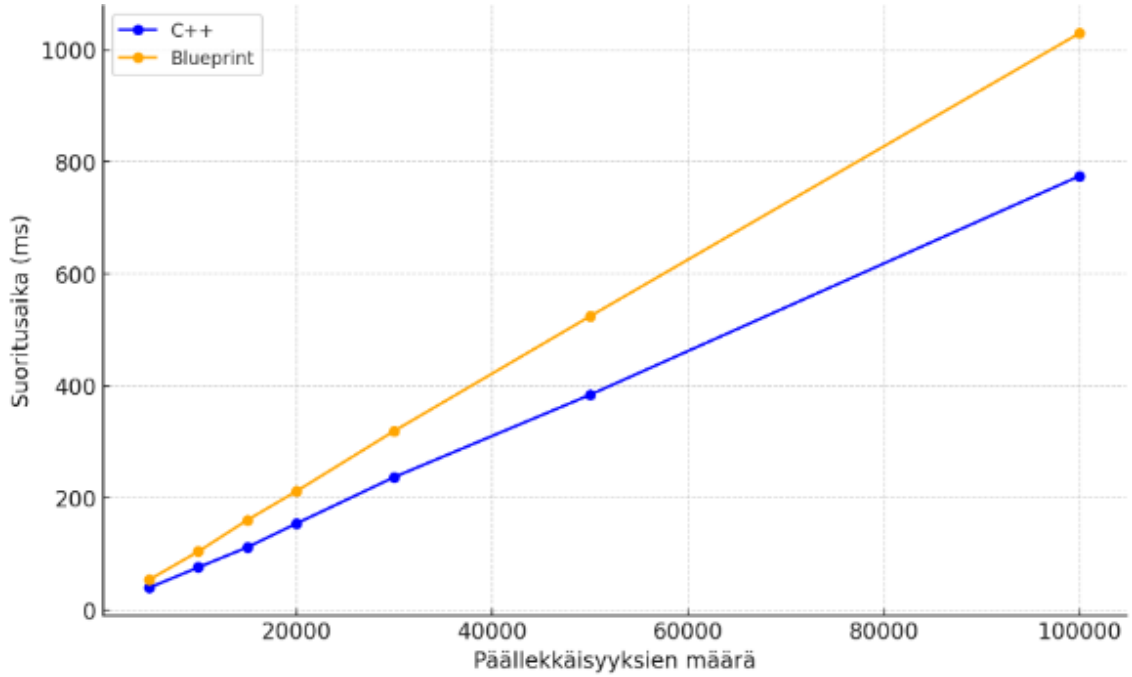
Hahmon kiertokulman ja sijainnin muokkaamista testattiin kahdessa eri tilanteessa. Ensimmäisessä testissä sekä sijainnin että kiertokulman muutos suoritettiin samanaikaisesti tuhansia kertoja käyttäen for-silmukkaa. Tämä mahdollisti tarkan analyysin suoritukseen kuluneesta ajasta, kun eri silmukkamäärien vaikutusta mitattiin millisekunneissa. Suoritusaika mitattiin

käyttäen Unreal Insights -työkalua, jotta on mahdollista saada erittäin tarkka tulos. Testi suoritetaan for-silmukan päällekkäistapaus arvoilla 5 000, 10 000, 15 000, 20 000, 30 000, 50 000 ja 100 000.

Toisessa testausvaiheessa keskityttiin realistisempaan pelitilanteeseen. Testissä pyöritettiin ja siirrettiin hahmoja jokaisella pelimoottorin päivityksellä. Testissä liikutetaan ja pyöritetään sadasta tuhanteen hahmoa samanaikaisesti ja mittaus suoritetaan Unreal Enginen omalla stat FPS komennolla. Stat FPS komento tuo ruudulle FPS lukeman, jolla on mahdollista analysoida, kumpi toteutus on tehokkaampi.

3.3.1 Kiertokulman ja sijainnin muutos for-silmukan avulla

Tässä testausvaiheessa toteutettiin hahmon kiertokulman ja sijainnin muutoksia käyttäen sekä Blueprint- että C++-luokkia Unreal Engine 5 -pelimoottorissa. Molemmat testitapaukset olivat identtisiä. Tutkimus suoritettiin käyttämällä for-silmukkaa eri arvoilla, jotta saataisiin tarkempi tulos. Pelaajahahmon sijaintiin ja kiertoon lisättiin 0.001 yksikköä eri toistomäärillä samanaikaisesti ja sitten laskettiin aika, joka kului koko operaatioon. Ajan mittaamiseen käytettiin Unreal Insights työkalua, josta aika tuli ilmi millisekunteina. Blueprint- että C++-toteutuksessa Add World Rotation -funktiota käytettiin kiertokulman muutoksen suorittamiseen. Sijainnin muutokseen tutkimuksessa oli käytössä Add World Offset -funktio.



Kuva 8. Sijainnin ja kiertokulman muutos eri for-silmukan arvoilla.

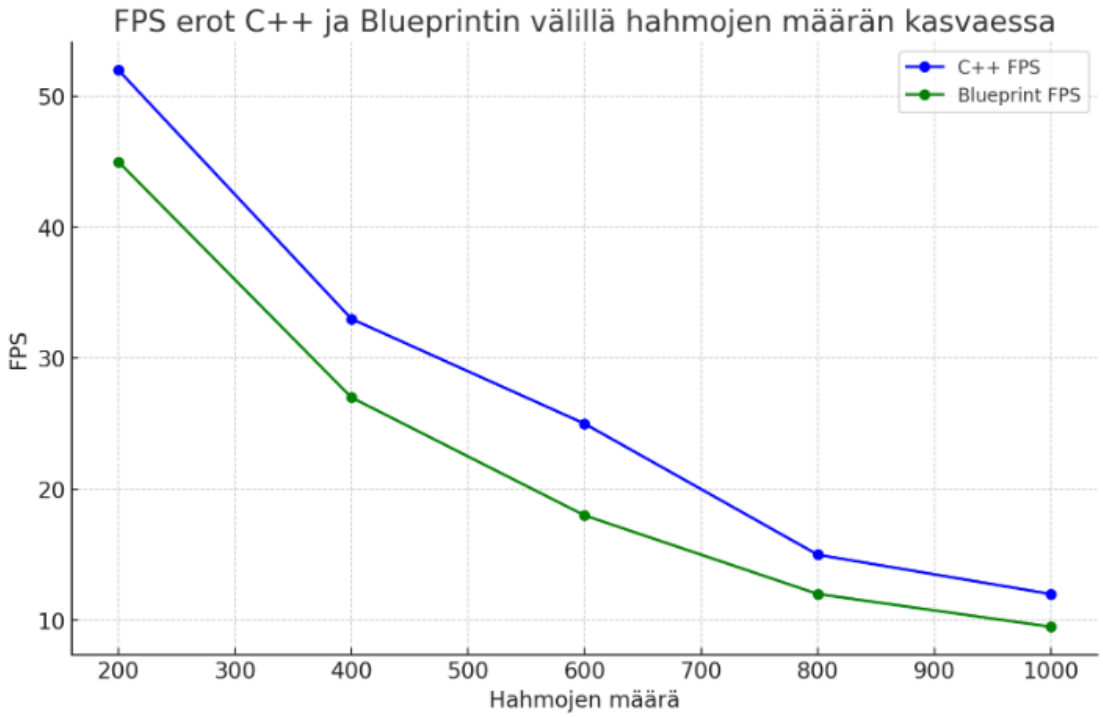
C++ osoittautui selvästi tehokkaammaksi Blueprintiin nähden, erityisesti suuremmilla toistomäärillä. Silmukkamäärien kasvaessa erot C++:n ja Blueprintin suorituskyvyssä lisääntyivät maltillisesti. Yli 100000 arvoilla testiä ei enää voinut suorittaa, koska Blueprint-versio pysähtyi ja lakkasi toimimasta toisin kuin C++ versio, jolla arvoa pystyttiin kasvattamaan jopa miljooniin asti.

Tulokset osoittivat, että C++-ohjelmointikielellä toteutettu koodi oli hieman tehokkaampi ja paremmin optimoitu kuin Blueprint-koodi. Näiden tulosten perusteella voidaan suositella C++-ohjelmointikielen käyttöä tilanteissa, joissa hahmojen kiertokulman ja sijainnin muuttaminen on keskeisessä roolissa, ja joissa suorituskyvyn optimointi on kriittistä.

3.3.2 Kiertokulman ja sijainnin muutos hahmoilla

Toisessa testausvaiheessa toteutettiin monien hahmojen kiertokulman ja sijainnin muutoksia käyttäen sekä Blueprint- että C++-luokkia. Kummatkin testitapaukset olivat jälleen identtisiä samalla tavalla kuten edellisissäkin testausvaiheessa. Tässä testauksessa pelimaailmaan sijoitettiin ensin 200

samaa hahmoa. Tämän jälkeen lisättiin hahmojen lukumäärää 200 hahmon välein tuhanteen hahmoon asti. Jokainen hahmo liikkui ja pyöri jokaisella pelimoottorin päivityksellä. Testissä laskettiin, kuinka monta päivitystä sekunnissa tapahtui eri määrillä hahmoja. Kummallakin toteutuksella sekä C++ että Blueprint päivitykset 1 s:ssa laskivat tasaisesti, mutta C++ toteutuksella päivityksiä oli aina enemmän, mikä viittaa parempaan suorituskykyyn.



Kuva 9. FPS-erot C++ ja Blueprintin välillä hahmojen määrän kasvaessa.

Kuvasta 9 voi huomata molemmissa testitapauksissa FPS laski tasaisesti, kun hahmoja lisättiin. C++ toteutus oli koko ajan hieman nopeampi ja tehokkaampi. Tulokset kertoivat, että C++ toteutus oli jälleen parempi kuin vastaava Blueprint toteutus. Työtehokkuuden kannalta molemmat testit olivat lähes yhtä nopea ohjelmoida. C++ toteutukseen käytettiin vain kaksi riviä ohjelmointi koodia, kuten kuvasta 10 voi huomata.

```
AddActorWorldOffset(FVector(0.1f, 0.0f, 0.0f));
AddActorWorldRotation(FRotator(0.1f, 0.0f, 0.0f));
```

Kuva 10. C++ toteutuksen koodi.

Kuva 11. Blueprint skriptin solmut.

Blueprint toteutukseen käytettiin vastaavasti kahta eri solmua, jotka vastaavat identtisesti C++ toteutusta. Tulokset tukevat C++ valintaa tilanteissa, joissa suorituskyky on keskeinen tekijä ja hahmojen sijainnin tai kiertokulman muuttaminen on tärkeää. C++ toteutus on tehokkaampi ja ajallisesti sen ohjelmointiin kuluu suunnilleen sama aika.

3.3.3 Yhteenveto

Tutkimuksessa vertailtiin C++- ja Blueprint-toteutuksia hahmojen kiertokulman ja sijainnin muuttamisessa Unreal Engine 5 -pelimoottorissa kahdessa eri testausympäristössä. Ensimmäisessä testissä suoritettiin useita peräkkäisiä toistoja for-silmukan avulla, ja tulokset mitattiin Unreal Insights -työkalulla tarkkojen millisekuntiarvojen saamiseksi. Toisessa testissä simuloitiin käytännön pelitilannetta mittaamalla FPS-arvoja, kun useita hahmoja liikutettiin ja käännettiin jokaisella pelimoottorin päivityksellä. Tulokset osoittivat selvästi, että C++-toteutus on suorituskykyisempi erityisesti silloin, kun hahmojen määrä tai toistomäärät ovat suuria. C++-ohjelmointikieli mahdollisti suuremmat toistomäärät ilman ohjelman kaatumista, ja FPS-lukemat pysyivät korkeampina kuin Blueprint-versiossa. Näiden havaintojen perusteella voidaan suositella C++-toteutusta erityisesti suorituskykykriittisiin tilanteisiin, joissa hahmojen sijainti ja kiertokulma ovat jatkuvassa muutoksessa. Vaikka C++-toteutus tarjosi

parempaa suorituskykyä, molemmat ratkaisut olivat melko yhtä nopeita koodata. Blueprint oli hieman parempi koodaustehokkuudessa sen graafisesta ja helpotetusta käyttöliittymästä johtuen.

4 Johtopäätökset ja tulevaisuuden tutkimusmahdollisuudet

Tämän opinnäytetyön tarkoituksena oli vertailla Unreal Engine 5 -pelimoottorin C++- ja Blueprint-skriptauskielten suorituskykyä ja kehitysprosessin haasteita pelikehityksessä. Tutkimus keskittyi erityisesti objektien dynaamiseen luontiin, matemaattisiin laskutoimituksiin sekä hahmojen kiertokulman ja sijainnin muuttamiseen. Tutkimuksen tulokset osoittavat, että C++ tarjoaa huomattavia etuja suorituskyvyssä, erityisesti suurten tietomäärien ja vaativien laskutoimitusten yhteydessä, kun taas Blueprint osoittautui hyödylliseksi nopeassa kehityksessä ja prototyyppien luomisessa.

4.1 Tutkimustulokset ja arviointi

Tutkimuksessa havaittiin, että C++-toteutukset olivat kautta linjan suorituskykyisempiä kuin Blueprint-toteutukset. C++ osoittautui erityisen tehokkaaksi objektien luomisessa ja monimutkaisissa matemaattisissa laskutoimituksissa. Blueprintit puolestaan tarjoavat nopeamman kehitysprosessin ja helppokäyttöisyyden, mikä tekee niistä sopivan valinnan projekteihin, joissa kehityksen nopeus ja helppous ovat tärkeitä tekijöitä.

Työskentelyä arvioitaessa on syytä huomata, että molemmilla kielillä on omat vahvuutensa ja heikkoutensa. C++:n etuna on sen tehokkuus ja laaja valikoima optimointimahdollisuuksia, mutta se voi olla monimutkaisempi ja alttiimpi virheille, mikä vaatii kehittäjältä syvällisempää teknistä osaamista. Blueprint-skriptaus tarjoaa helpommin lähestyttävän tavan kehittää pelejä ja niiden logiikkaa, mutta ne voivat johtaa heikompaan suorituskykyyn ja vaikeaselkoisiin ratkaisuihin monimutkaisissa projekteissa.

4.2 Tulosten hyödyntäminen

Tutkimuksen tuloksia voidaan hyödyntää monella tapaa pelikehityksen eri vaiheissa, erityisesti työkalujen valinnassa ja projektien suunnittelussa. C++-ohjelmointikielen käyttöä suositellaan projekteihin, joissa suorituskyvyn optimointi on keskeistä ja joissa tarvitaan laajaa teknistä osaamista ja kontrollia alustan toimintaan. Tämä on erityisen tärkeää, kun kehitetään laajoja, teknisesti vaativia pelejä, joissa tarvitaan tarkkaa hallintaa muistinkäytöstä, laskennasta tai suorittimen kuormituksesta. C++ mahdollistaa kehittäjille syvällisemmän pääsyn pelimoottorin ytimeen, mikä mahdollistaa tehokkaampien ja tarkempien ratkaisujen toteuttamisen vaativissa skenaarioissa.

Blueprint-visualisointikieli tarjoaa puolestaan joustavan ja helppokäyttöisen tavan nopeaan pelikehitykseen ja prototyyppien luomiseen. Sen avulla voidaan nopeasti toteuttaa ja testata uusia ominaisuuksia, mikä mahdollistaa ideointivaiheen nopean etenemisen ja helpottaa iteratiivista kehitysprosessia. Blueprintit ovat erityisen hyödyllisiä tilanteissa, joissa kehittäjätiimissä on jäseniä, joilla ei välttämättä ole syvällistä ohjelmointitaustaa tai silloin, kun projekti edellyttää tiivistä yhteistyötä pelisuunnittelijoiden ja ohjelmoijien välillä. Tällöin Blueprintien intuitiivinen käyttöliittymä mahdollistaa nopean vuorovaikutuksen ja iteroinnin eri ammattiryhmien välillä.

Keskeinen havainto tutkimuksesta on se, että C++-ohjelmointikielen ja Blueprint-työkalujen yhdistäminen saattaa usein olla paras ratkaisu monimutkaisissa projekteissa. C++-koodia voidaan käyttää silloin, kun tarvitaan tarkkaa hallintaa suorituskyvystä tai kun tietyt pelilogiikan osat vaativat suurempaa tehokkuutta ja matalamman tason optimointia. Toisaalta Blueprintit tarjoavat nopeutta ja ketteryyttä pelinkehityksen luovammissa ja vähemmän teknisesti vaativissa osioissa. Tämän lähestymistavan avulla pelikehittäjät voivat valita kuhunkin tehtävään sopivimman työkalun, mikä parantaa projektin tehokkuutta ja lopputuotteen laatua.

Tutkimus tulokset auttavat pelikehittäjiä tekemään tietoisempia ja perustellumpia valintoja työkalujen ja menetelmien osalta sekä tarjoavat pohjan

käytäntöjen kehittämiselle erilaisten projektien ja tiimien tarpeisiin. Työkalujen valinnassa on tärkeää huomioida projektin laajuus, tekniset vaatimukset, tiimin osaaminen ja kehitysaikataulu. Kun nämä kaikki tekijät on otettu huomioon, voidaan pelin kehityksessä tarvittavat työkalut suunnitella ja organisoida tehokkaasti. Yhdistämällä C++-ohjelmoinnin tehokkuus ja Blueprintien nopea kehityssykli, kehittäjät voivat optimoida sekä resurssien käytön että työskentelyprosessin joustavuuden.

4.3 Tulevaisuuden tutkimusmahdollisuudet

Jatkotutkimuksia voisi suunnata erityisesti siihen, miten ohjelmointikielen valinta vaikuttaa pelaajien kokemukseen ja pelin laatuun. Tämä edellyttäisi käyttäjätestauksia, joissa arvioidaan pelin sujuvuutta ja käyttäjäkokemusta eri kielillä toteutetuissa projekteissa. Lisäksi on tarpeen analysoida, kuinka C++ ja Blueprintit vaikuttavat monimutkaisten pelimekaniikkojen kehittämiseen ja ylläpitoon. Tähän sisältyisi esimerkiksi eri mekaniikkojen suorituskyvyn vertailu ja niiden kehitysaikataulujen arviointi. On myös huomioitavaa tutkia tekoälyn vaikutusta C++:n ja Blueprintin työtehokkuuteen. Esimerkiksi ChatGPT ei osaa luoda Blueprint-koodia, mutta se on erittäin hyvä luomaan C++-koodia. Tämä voi varsinkin tulevaisuudessa muuttaa asetelmaa, että Blueprintti on nopeampi kieli ohjelmoida.

Tutkimuksissa voisi myös tarkastella, miten eri ohjelmointikielet vaikuttavat yhteistyöhön pelinkehitystiimien sisällä. Tässä kontekstissa olisi tärkeää selvittää, miten kielivalinnat voivat edistää tai estää tehokasta viestintää ja yhteistyötä moniammatillisissa tiimeissä. Lisäksi tulisi kehittää uusia menetelmiä ja työkaluja, jotka mahdollistavat suorituskyvyn parantamisen sekä C++- että Blueprint-toteutuksissa. Tämä voisi sisältää esimerkiksi uusia optimointitekniikoita tai työkaluja, jotka helpottavat molempien kielten yhteiskäyttöä pelikehityksessä.

Lähteet

Denham, T. (2019). What is Unreal Engine? Viitattu 3.11.2024, osoitteesta https://conceptartempire.com/what-is-unreal-engine/

Epic Games (n.d.a). Unreal Engine 5.0 release notes. Unreal Engine Documentation. Viitattu 26.2.2024, osoitteesta https://docs.unrealengine.com/5.0/en-US/unreal-engine-5.0-release-notes/

Epic Games (n.d.b). Balancing Blueprint and C++. Unreal Engine 4 Documentation. Viitattu 26.2.2024, osoitteesta https://docs.unrealengine.com/4.27/en-US/Resources/SampleGames/ARPG/BalancingBlueprintAndCPP/

Epic Games (2024a). Introduction to Blueprints. Unreal Engine 5 Documentation. Viitattu 28.10.2024, osoitteesta https://dev.epicgames.com/documentation/en-us/unreal-engine/introduction-to-blueprints-visual-scripting-in-unreal-engine

Epic Games (2024b). Actor Ticking. Unreal Engine 5 Documentation. Viitattu 28.10.2024, osoitteesta https://dev.epicgames.com/documentation/en-us/unreal-engine/actor-ticking-in-unreal-engine?application_version=5.5

Epic Games (2024c). Skeletal Mesh Actors. Unreal Engine 5 Documentation. Viitattu 28.10.2024, osoitteesta https://dev.epicgames.com/documentation/en-us/unreal-engine/skeletal-mesh-actors-in-unreal-engine?application_version=5.5

iD Tech. M. (2020). C++ vs. Blueprints. Viitattu 6.11.2024, osoitteesta https://www.idtech.com/blog/c-vs-blueprints-differences

Joanna Lee, John P. Doran, Nitish Misra. (2016). Unreal Engine: Game Development from A to Z. Viitattu 11.11.2024, osoitteesta https://books.google.fi/books?id=bKbWDQAAQBAJ&lpg=PP1&ots=uFM8fJOmg9&dq=Unreal%20Engine%3A%20Game%20Development%20from%20A%20to%20Z&lr&hl=fi&pg=PP2#v=onepage&q&f=false

Phil (2024). What is a game engine: An essential overview for beginners. Viitattu 11.11.2024, osoitteesta https://drawandcode.com/learning-zone/what-is-a-game-engine/

Program Ace (2024). Unreal Engine Blueprints vs. C++: Performance and Ease of Use Compared. Viitattu 29.10.2024, osoitteesta https://program-ace.com/blog/unreal-engine-blueprints-vs-c/

Romero, M., & Sewell, B. (2022). Blueprints Visual Scripting for Unreal Engine 5 Packt Publishing. Viitattu 29.10.2024, osoitteesta https://books.google.fi/books?id=yhZuEAAAQBAJ&lpg=PP1&ots=ut-2tKLnuz&dq=unreal%20engine%205&lr&hl=fi&pg=PP1#v=onepage&q=unreal%20engine%205&f=false

OpenAI (2024). ChatGPT. Viitattu 28.10.2024, osoitteesta https://chat.openai.com/

Unity (n.d.). Unity Visual Scripting. Viitattu 28.10.2024, osoitteesta https://unity.com/features/unity-visual-scripting

Valcasara, N. (2015). Unreal Engine Game Development Blueprints. Packt publishing. Viitattu 27.2.2024, osoitteesta https://books.google.fi/books?id=4iLlCwAAQBAJ

Wikipedia (2024a). Kuvataajuus. Viitattu 28.10.2024, osoitteesta https://fi.wikipedia.org/wiki/Kuvataajuus

Wikipedia (2024b). C++. Viitattu 28.10.2024, osoitteesta https://fi.wikipedia.org/wiki/C%2B%2B

Honkala, T. (1998). Unreal – Untako vain? ”Unreal 1998”. Viitattu 12.11.2024, osoitteesta https://www.pelit.fi/artikkelit/unreal-4/