LAB University of Applied Sciences

Pelaajahahmon mainejärjestelmä videopeliin

Case: Pixtell oy

LAB-ammattikorkeakoulu
Insinööri (AMK)
2023
Jari Mäenpää

Tiivistelmä

Tekijä(t)	Julkaisun laji	Valmistumisaika
Jari Mäenpää	Opinnäytetyö, AMK	2023
	Sivumäärä 24	

Työn nimi

Pelaajahahmon mainejärjestelmä videopeliin

Case: Pixtell oy

Tutkinto ja koulutusala

Insinööri (AMK), tieto- ja viestintätekniikan koulutus

Toimeksiantajaorganisaatio (jos opinnäytetyöllä on toimeksiantaja)

Pixtell oy

Tiivistelmä

Opinnäytetyön tavoitteena oli toteuttaa maine-, ryhmittymä- sekä tervehtimisjärjestelmät Unreal Enginellä luotavaan videopeliin The Last Drop: Legend of Seppo. Toteutustapana käytettiin Unreal Enginen omaa Blueprint-järjestelmää sekä C++ -ohjelmointikieltä. Työn toimeksiantaja oli pelin kehittäjä Pixtell oy.

Työssä tutkittiin Unreal Enginen historiaa sekä sen tärkeimpiä virstanpylväitä aina ohjelmointikielen vaihdosta moottorin lisensointimallin muuttamiseen. Tutkittiin myös Blueprintin ja C++-ohjelmointikielen perustoja ja näiden suhdetta ja yhtenäistä toimivuutta Unreal Enginen sisällä.

Tervehtimis-, ryhmittymä- ja mainejärjestelmät rakennettiin alusta asti yhtenäiseksi kokonaisuudeksi. Mainejärjestelmät ja sen osana olevat ryhmittymät kirjoitettiin C++:lla, josta tarvittavat tiedot pystyttiin lähettämään pelimoottoriin ja käyttämään sen sisällä. Tervehtimisosuus tuotettiin Blueprinteillä ja se kiinnitettiin osaksi muita järjestelmiä.

Järjestelmien lopullinen toimeenpano pelin sisälle jäi vielä tekemättä, sillä yhtä järjestelmien kannalta oleellista C++-koodia ei vielä projektin tekovaiheessa oltu implementoitu toimimaan pelimoottorin sisälle. Työssä ei keskitytty järjestelmien tukiominaisuuksien luomiseen. Tarkoitus oli tehdä pohjarakenne, johon pystyttytäisiin myöhemmin lisäämään haluttuja ominaisuuksia.

Asiasanat

Unreal Engine, Blueprint, C++, videopeli

Abstract

Author(s)	Type of Publication	Published
Jari Mäenpää	Thesis, UAS	2023
	Number of Pages 24	

Title of Publication

Player characters reputation system for a videogame

Case: Pixtell oy

Degree, Field of Study

Engineer (UAS), Information and Communications Technology

Organisation of the client (if the thesis work is commissioned by another party)

Pixtell oy

Abstract

The objective of this thesis was to create three in-game systems: greeting, faction, and player reputation for a videogame called Last Drop: Legend of Seppo. Since the game uses Unreal Engine as its game engine, both Blueprint and C++ were used in this project. The client for this thesis was the games developer Pixtell oy.

The purpose was to examine the history and the most notable milestones of Unreal Engine and the foundation for both the Blueprint system and C++. The thesis also investigates how Blueprint and C++ can work and operate together inside Unreal Engine.

Greeting, faction, and reputation systems were meant to function as a cohesive unit. Basis for the reputation and faction systems was built on C++ from where the necessary info was sent to the game engine and Blueprints. Greeting system was created using Blueprint and it was bonded to the other systems.

Because of one of the C++-codes used in the systems had not yet been implemented into the game itself, the final merging of these systems into the larger whole was not done during the writing phase of the thesis. The focus for the thesis was not on creating the support features such as animations and audios for these systems. The main purpose was to create a solid foundation to which all the necessary elements could be added later.

Keywords

Unreal Engine, Blueprint, C++, video game

Sisällys

1 Johdanto ... 1
2 Tausta Last Drop: Legend of Seppo ... 3
3 Ohjelmistot ... 4
3.1 Unreal Engine ... 4
3.2 Blueprint ... 6
3.3 C++ ... 10
3.4 Blueprintin ja C++ yhteistoiminta Unreal Enginessä ... 11
4 Case: Pelaajahahmon mainejärjestelmä ... 13
4.1 Työn taustatiedot ... 13
4.2 Tervehtiminen ... 14
4.2.1 Hahmon tervehtiminen ... 14
4.2.2 Tervehtimisanimaatio ... 16
4.3 Ryhmittymät ... 17
4.4 Mainejärjestelmä ... 19
4.5 Vastatervehdys ... 21
5 Yhteenveto ja pohdinta ... 24
Lähteet ... 25

1 Johdanto

Videopelit koostuvat ulkoisten aspektien lisäksi monista erinäköisistä ja eri käyttötarkoituksiin ohjelmoiduista järjestelmistä. Nämä järjestelmät ohjaavat peliä saaden sen toimimaan tarkoitetulla tavalla. Jotkin järjestelmät voivat toimia täysin itsenäisesti suorittaen ja hallinnoiden omaa osuuttaan ilman muiden järjestelmien apua. Toiset järjestelmät ovat sidottuja toisiinsa ja laajempaan toimintaympäristöönsä, jolloin ne ovat riippuvaisia toisistaan oman toimintansa ylläpitämiseksi. Ne lähettävät ja vastaanottavat tietoa toisiltaan ollen osa isompaa kokonaisuutta.

Tämä opinnäytetyö keskittyy näistä jälkimmäiseen. Tavoitteena on luoda kolme järjestelmää, jotka ovat kiinni toisissaan halutun lopputuloksen saavuttamiseksi. Nämä kolme järjestelmää hallinnoivat pelaajan sekä npc-hahmojen (eng. non-playable character) välistä tervehtimistä, pelialueen ryhmittymiä sekä pelaajan mainetta kyseisissä ryhmittymissä. Mainejärjestelmän luominen on opinnäytetyön pääasiallisin tavoite, tervehtiminen ja ryhmittymät ovat vain osa maineen hallintaa ja operointia. Näiden järjestelmien avulla peli antaa pelaajalle mahdollisuuden tervehtiä tietokoneen ohjaamia hahmoja, samalla pitäen sisäisesti kirjaa pelaajan maineesta jokaisessa ryhmittymässä. Tämä mahdollistaa sopivan vastauksen tai toimenpiteen pelaajaan hänen tervehtiessään npc-hahmoja. Mainitut järjestelmät toteutetaan Unreal Enginen omalla visuaaliseen ohjelmointiin perustuvalla Blueprint-ohjelmointialustalla, C++-ohjelmointikielellä sekä näitä yhdistelemällä.

Pixtell oy on jyväskyläläinen videotuotantoon erikoistunut yritys, joka perustettiin tammikuussa 2020 Henri Koskisen ja Harri Paavolan toimesta. Yrityksen missiona tehdä ja tuottaa hinnaltaan asiakasystävällisiä tarinallisia videoita. Idea videopelistä oli ollut Koskisen mielessä pitkään, alun perin pelin ensimmäisiä versioita kehitettiin jo vuonna 2010, mutta silloin pelin kehitys tyrehtyi nopeasti. Nykyisessä muodossaan olevan pelinkehitysprojektin Koskinen käynnisti tammikuussa 2020 kehittäen ensimmäisen testiversionsa itse ja tämän jälkeen rekrytoiden mukaansa muita pelin kehittäjiä edistämään pelin kehitystä. Alun jälkeen pelinkehittäjien määrä on kasvanut huomattavasti, sillä alussa Koskisen lisäksi projektitiimissä peliä kehitti neljä henkilöä, nykyisin luku on lähes nelinkertaistunut alkuperäisestä. Projektitiimin lisäksi Pixtell on tarjonnut opiskelijoille työharjoittelu- ja opinnäytetyömahdollisuuksia, näin kasvattaen kehitystiimin määrää entisestään.

Videopelissä Last Drop: Legend of Seppo pelaaja navigoi pelimaailmassa, joka sisältää monia tietokoneen ohjaamia hahmoja. Nämä hahmot, tuttavallisemmin npc-hahmot kuuluvat pelin sisällä moniin eri ryhmittymiin, joiden kanssa pelaajan pitää pystyä toimimaan. Jotkin ryhmittymät suhtautuvat positiivisemmin pelaajan pelihahmoon kuin toiset, mutta

vuorovaikuttamalla tervehtimisen muodossa ryhmittymän hahmoihin pystyy pelaaja parantamaan tai pahentamaan asemaansa ja mainettaan kyseisessä ryhmässä.

Koska peli on vielä kehitysvaiheessa ja muutokset lopulliseen tuotokseen ovat mahdollisia, kaikki järjestelmät luodaan alusta asti niin, että niitä on helppo muokata ja säätää toimimaan toivotulla tavalla uusien haluttujen ominaisuuksien lisäämisen muodossa. Mainejärjestelmä tullaan myös sitomaan muihin olemassa oleviin tai tuleviin pelin toimintoihin, esimerkiksi kaupankäyntiin ja alennusten saamiseen sekä tehtävien aukeamiseen. Tämä ei kuitenkaan ole osa projektin toteutusta. Myöskään itse tervehtimisanimaatioiden ja animaatioissa käytettävien audioiden luonnit eivät kuulu projektin toteutukseen. Tarkoitus on tehdä yksinkertaisesti käytettävä pohja, mihin kyseisiä ominaisuuksia pystytään myöhemmin liittämään.

2 Tausta Last Drop: Legend of Seppo

Last Drop: Legend of Seppo sijoittuu vuoden 1994 Jyväskylän lähiöihin. Tarina keskittyy kertomaan alkoholismista, jota myös päähenkilönä toimiva Seppo (kuva 1) sairastaa. Pelin tavoitteena on kuvastaa aitoa suomalaista tunnelmaa aihealueen ympärillä. Pelialue on pelaajalle avoin, mahdollistaen alueilla kulkemisen ja seikkailun ja uusien alueiden tutkimisen.

Kuva 1. Pelin päähahmo Seppo (Jari Mäenpää)

Pelin pelattavuutta kehitetään sillä ajatuksella, että peliä pitäisi olla helppo pelata, mutta samalla sen pitää pystyä tarjoamaan haasteita pelaajalle. Kehityksessä on tarkoituksellisesti haluttu välttää sitä, että pelissä olisi erinäisiä minipelejä, loputtomia tutoriaaleja tai tarpeettomia indikaattoreita. Pelin juoni, tunnelma ja ominaisuudet ovat ottaneet vaikutteita pääkehittäjien omien havaintojen lisäksi monista muista videopeleistä, näistä mainittakoon monet erilaiset klassikkopelit kuten Legend of Zeldat ja Final Fantasyt sekä modernimmat julkaisut kuten esimerkiksi Grand Theft Auto 5 ja Red Dead Redemption 2.

Pelin kehityksen alkuvaiheessa vuonna 2020 pelimoottoriksi valikoitui Unreal Engine, kun todettiin, että Unreal Engine pystyi täyttämään laatukriteerit paremmin kuin muut tarjolla olevat pelimoottorit. Tästä eteenpäin visio pelistä on koko ajan kehittynyt sekä pelin laatutaso on noussut, kun tekijätiimi on kasvanut ja kehittynyt.

3 Ohjelmistot

3.1 Unreal Engine

Unreal Enginen historia alkoi Epic Gamesin kehittämästä ja vuonna 1998 julkaisemasta pelistä Unreal. Peli vakuutti pelaajat ja muut pelinkehittäjät sekä näyttävillä grafiikoillaan että myös kyvyllään käsitellä isojakin pelialueita vaivattomasti. Kun kiinnostusta pelimoottoriin ja sen vuokraamiseen esiintyi, Epic Games päätti lisensoida pelimoottorinsa ulkopuolisten toimijoiden käyttöön. (Thomsen 2012.)

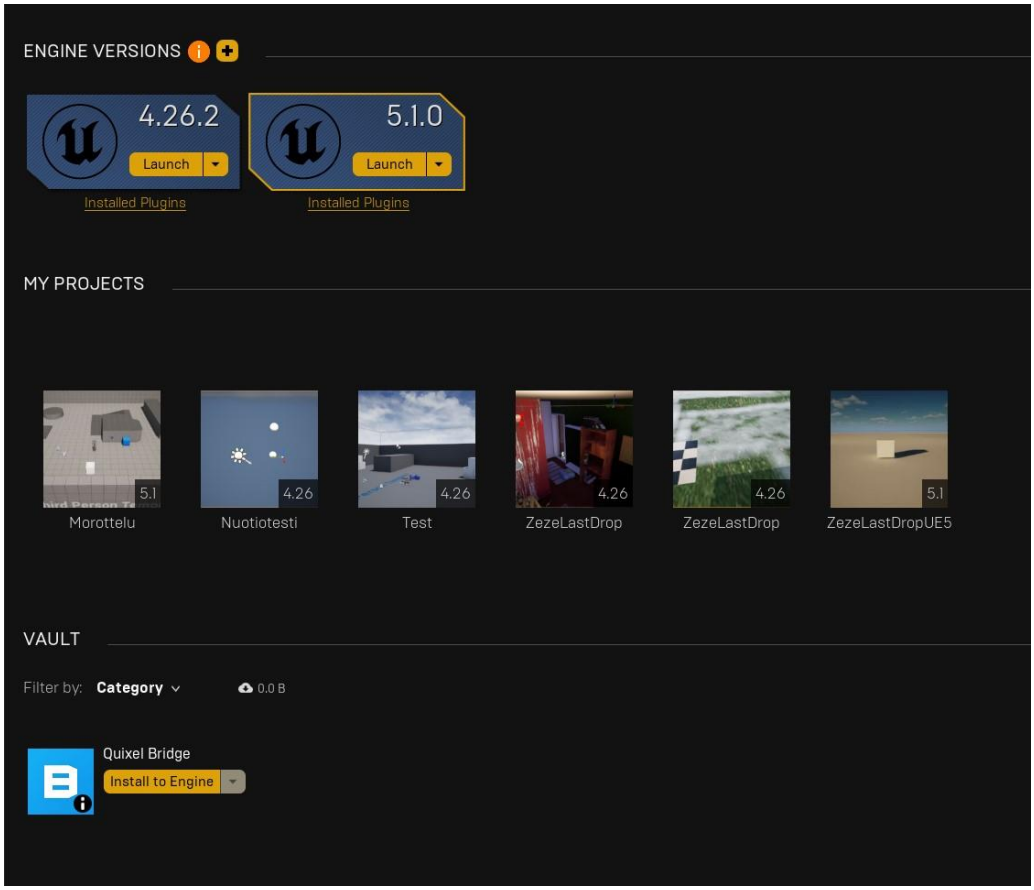
Viimeistään Unreal Engine 2 aikana pelimoottorilla kehitettiin paljon hyvin toisistaan eroavia pelejä. Tämä ilmiö sai aikaan sen, että tietyn pelimoottorin käyttö ei enään sanellut sitä, minkälainen peli tulisi olemaan. Pelien kehittämisen monipuolisuus nostatti kysyntää Unreal Enginelle entisestään. Valtavan kysynnän kasvun myötä pelimoottorin lisensoinnista kehittyi yksi Epic Gamesin suurimmista, luotettavimmista ja tuottoisimmista tuotteista. Epic Games päätti ajoittaa seuraavan sukupolven pelimoottorinsa Unreal Engine 3:n julkaisun juuri ennen uusien pelikonsoleiden saapumista. Tämä takasi pelinkehittäjille mahdollisuuden kehittää pelejä uusille tuleville konsoleille, jotta pelit olisivat valmiita konsolien julkaisun aikaan. (Thomsen 2012.)

Unreal Engine 3:een asti pelimoottori käytti juuri Unreal Engineen kirjoitettua ohjelmointikieltä nimeltään UnrealScript. Ohjelmointikielen tarkoitus oli olla helppo, mutta korkeatasoinen Javaa muistuttava kieli. Vuonna 2015 julkaistu Unreal Engine 4 kuitenkin poisti UnrealScriptin ja toi sen tilalle omiin tarkoitusperiinsä muokatun C++-ohjelmointikielen. Syynä vaihdokseen oli C++-kielen laaja tunnettavuus ja huoli UnrealScriptin suorituskyvystä. (Unreal Engine a.; Schultz.)

Unreal Engine 1:stä alkaen kehittäjien piti maksaa Epic Gamesille jo peliensä kehitysvaiheessa, jotta he saivat pelimoottorin käytettäväkseen. Maaliskuussa 2015 Epic Games kuitenkin vapautui Unreal Enginen ilmaiseksi käyttöön kaikille pelinkehittäjille Unreal Engine 4:n julkaisun myötä. Kehitysaikaisen ilmaisuuden sijaan Epic Games alkoi periä 5 prosenttia pelien tuotoista ensimmäisen 3000 Yhdysvaltojen dollarin (USD) jälkeen joka vuosineljännes. Nykyisin hinnoittelumalli on muuttunut hieman aikaisemmasta ja laskenta tuloista perittävään 5 prosentin maksuun alkaa vasta ensimmäisen miljoonan Yhdysvaltain dollarin jälkeen. (Sirani 2017; Unreal Engine b.)

Epic Games on aina lähtenyt Unreal Enginen kehittämisessä siitä, että kehitystyössä on paljon kommunikaatiota pelinkehittäjien ja pelimoottorin kehittäjien välillä. Tämä antaa pelimoottorin kehittäjille arvokasta palautetta moottorin toimivuudesta ja kehityskohteista. Samalla Unreal Enginen kehittämisessä on panostettu näyttäviin grafiikoihin ja pelimoottorin

helppokäyttöisyyteen, mistä Blueprint-ohjelmointijärjestelmä on esimerkkinä, sekä monipuolisuuteen julkaisualustojen suhteen. Kuvassa 2 nähdään Unreal Enginen projektivalikkoa, josta kehittäjän on helppo valita jo olemassa olevista projekteistaan tai aloittaa täysin uuden projektin luonti. (Thomsen 2012; iD Tech 2016.)



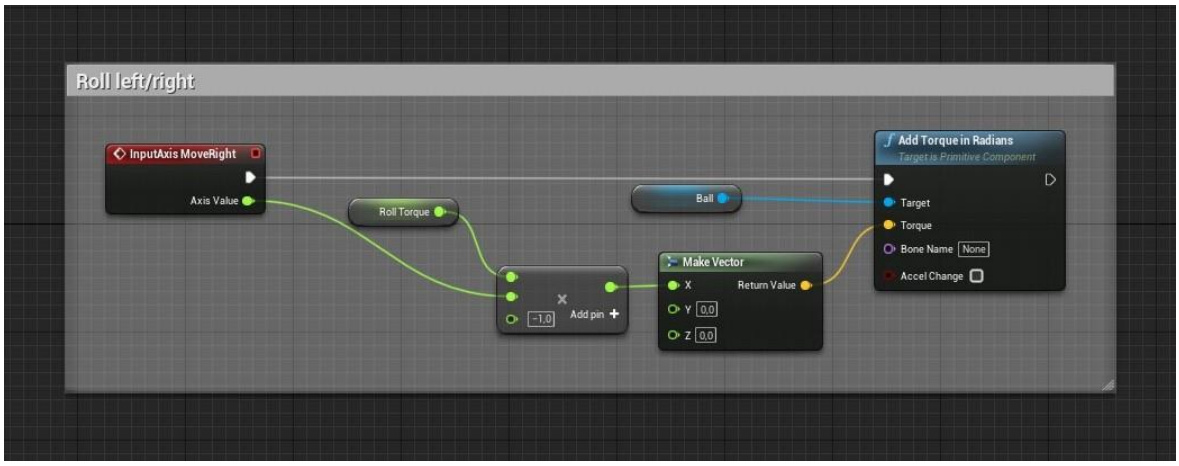
Kuva 2. Unreal Enginen projektivalikko (Jari Mäenpää)

Pelimoottorimarkkinoilla on monia eri valmistajien pelimoottoreita. Kun verrataan julkaistujen pelien määriä videopelialusta Steamissä per vuosi, vuonna 2021 Unreal Engine sijoittui vertailussa toiseksi edellään vain pelimoottori Unity. Unity on tunnettu varsinkin pienempien pelistudioiden käyttämänä pelimoottorina, joten sen avulla kehitettyjä ja julkaistuja pelejä oli vuonna 2021 selvästi yli puolet enemmän kuin Unreal Enginellä. Syitä Unityn suosioon ovat ennen kaikkea sen käyttämä helppokäyttöinen C# ohjelmointikieli sekä hyvin toimiva pelien järjestelmäriippumaton tuki eri alustoille ja konsoleille. Unity eroaa Unreal Enginestä sen käyttämän hinnoitteluperiaatteen vuoksi, sillä Unity perii lisenssiä vain pelien kehitysvaiheessa. Pelin julkaisun jälkeen lisenssimaksuja ei kehittäjiltä enää vaadita. (Doucet & Pecorella 2021; Unity)

3.2 Blueprint

Blueprint on Unreal Enginen sisäinen visuaaliseen ohjelmointiin perustuva järjestelmä, jolla kehittäjä pystyy joustavasti ja nopeasti luomaan toiminnallisuutta ilman tarvetta kirjoittaa koodeja Unreal Enginen tukemalla C++-ohjelmointikielellä. Blueprintin tarkoituksena on toimia kokonaisvaltaisena ohjelmointijärjestelmänä, jolla on mahdollista luoda kaikki tarvittavat elementit pelieditorin sisällä. Blueprint perustuu olio-ohjelmoitavan solmupohjaisen käyttöliittymän käyttöön, jossa määritellään olioita tai olio-ohjattuja luokkia pelimoottorin sisällä. Jokainen editorin sisälle tuotu tai siellä luotu komponentti muodostaa yhden solmun ja näistä solmuista muodostuu solmusarjoja. (Unreal Engine c.)

Solmut etenevät vasemmalta oikealle seuraten solmujen välissä meneviä viivoja. Valkoinen viiva kertoo tapahtumien pääasiallisen etenemisen ja muun väriset viivat kertovat pääsolmuista lähtevistä tai niihin tulevista lisäyksistä ja tarkennuksista. Kuvan 3 solmusarja alkaa näppäinsyötteestä, josta lähtee suora linkki sarjan päätteeseen, jonka tarkoitus on pyörittää objektia. Näiden kahden pisteen välissä vihreissä solmuissa määritellään pyörimisen voimakkuuksia, sekä päätepisteelle kerrotaan sen kohde, mitä pyörittäminen koskee. (Unreal Engine d.)



Kuva 3. Blueprintin solmusarja (Jari Mäenpää)

Blueprinttejä hallinnoidaan Unreal Enginen sisällä yleisimmin Blueprint editorin avulla. Se on pohjimmiltaan solmupohjainen grafiikkaeditori, joka toimii Blueprintin käyttö- ja ohjelmointityökaluna. Kun tiedosto avataan Blueprintissä, aukeaa se Blueprint editoriin. Editor näkymä näyttää kaikki kyseisin Blueprintin sisältämät ominaisuudet, joita käyttäjä pääsee käsittelemään helposti yhden työkalun avulla. (Unreal Engine e.)

Blueprintin ominaisuudet

Unreal Engine sisältää monia eri käyttötarkoitukseen luotuja Blueprinttejä. Ne antavat kehittäjille monipuolisen valikoiman luoda juuri heille sopivia kokonaisuuksia. Blueprint Class (luokka), jota yleisemmin kutsutaan vain pelkästään Blueprintiksi on näistä yleisin, sillä lähes kaikki pelin sisäiset toimijat ovat Blueprint-luokkia. Ne antavat kehittäjälle mahdollisuuden lisätä toiminnallisuutta olemassa oleviin peliluokkiin. Data-only (vain data) Blueprintit sisältävät vanhemmilta (parent) perittyjä solmuja, jota on mahdollista muokata, mutta uusien solmujen lisäys ei onnistu. Level (taso) Blueprintit hallinnoivat kokonaisia tasoja ja tapahtumia niiden sisällä, uusi Level Blueprint syntyy automaallisesti uuden tason luonnin myötä. Blueprint Interface (käyttöliittymä) toimii funktioiden kokoelmana, josta toiset Blueprintit voivat jakaa ja lähettää tietoa toisilleen. Blueprint Macro Library (makrokirjasto) toimii säiliönä makroille ja kaavioille, joita voidaan käyttää solmuina toisissa Blueprinteissä. Kaikista näistä Blueprint Class ja Level Blueprint ovat yleisimmin käytössä, sillä ne toimivat pohjana pelinkehityksessä ja ilman niitä on muiden Blueprinttien käyttö on erittäin hankalaa. (Unreal Engine c.)

Elementit, joilla Blueprintin toiminnallisuutta muokataan, voivat määritellä komponentteja, suorittaa alustus- tai asetusoperaatioita, reagoida tapahtumiin, organisoida ja mukauttaa toimintoja ja määritellä ominaisuuksia. Blueprintin sisäisiä ominaisuuksia ovat muun muassa

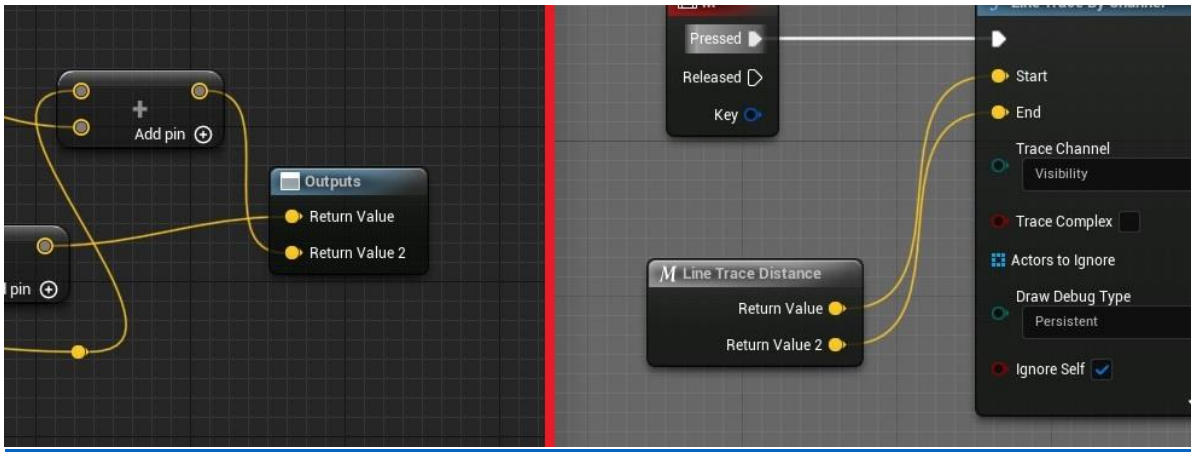
- Komponentti-ikkuna, joka mahdollistaa komponenttien lisäämisen Blueprinttiin. Komponentteihin sisältyvät muun muassa törmäysgeometria, renderöinti geometria (StaticMesh) ja liikuntakomponentit.
- Rakennuskomentosarja seuraa komponenttilistaa, kun Blueprint Class luodaan. Sen sisällä on solmukaavio, jolla Blueprint Class voi suorittaa toimintoja.
- Event Graph (tapahtumakaavio) on solmukaavio, joka suorittaa tapahtumia ja funktiokutsuja vastauksena pelitapahtumiin.
- Funktiot ovat solmukaavioita, jotka kuuluvat tiettyyn Blueprinttiin, joita voidaan toteuttaa tai kutsua toisesta saman Blueprintin kaaviosta.
- Muuttujat pitävät sisällään arvoja tai viittauksia objekteihin tai toimijoihin (actor). Nämä voivat olla vain tietyn Blueprintin käytettävissä tai jaettavissa myös muille Blueprinteille.

- Blueprintin tilassa määritellään, mikä asia näkyy editorin ikkunassa. Näihin lukeutuvat muun muassa tasopiirrokset, makrot, funktiot sekä tapahtumakaavio. Blueprint Class sisältää myös oletustilan, missä määritellään oletusasetuksia, ja komponenttitilan, jossa lisätään, muokataan ja poistetaan komponentteja.

(Unreal Engine c.)

Makrot

Blueprintissä makrot ovat kutistettuja tai omaan pakettiinsa koottuja solmusarjoja. Niillä on omat input ja output -pisteensä ja ne voivat sisältää kehittäjien määrittelemää toiminnallisuutta. Makrot toimivat vain siinä Blueprintissä, mihin ne ovat luotu, niitä ei siis voi kutsua kyseisen Blueprintin ulkopuolella. Kuvan 4 vasemmalla puolella näkyy, kuinka makron sisäinen toiminta on jakautunut kahteen erilliseen arvoon ja molemmat arvot päättyvät output pisteeseen. Oikealla puolella näkyy, kuinka molempia tehdyn makron arvoja kutsutaan samassa Blueprintissä, tässä tapauksessa nimellä Line Trace Distance. Makrojen tarkoituksena on tiivistää ja siistiä solmuilla suoritettavaa toimintaa, mahdollistaen samalla yksittäisen makron kutsumisen useaan otteeseen samassa kokonaisuudessa ilman, että sen sisältämä funktio kirjotettaisiin useasti. (Unreal Engine f.)



Kuva 4. Line Trace -makron Output-piste ja sen kutsu Event Graphissa (Jari Mäenpää)

Komponentit

Komponentit ovat Blueprintin toiminnallisuutta ja käyttäytymistä lisääviä objekteja, joita voidaan liittää toimijoihin (Actor). Vaikka komponentit luodaan omiksi kokonaisuuksiksi, ne eivät pysty operoimaan ilman toimijaansa. Esimerkkeinä erilaisista komponenteista ovat

- Liikekomponentit, joilla ohjaillaan hahmojen liikettä.
- Hahmojen attribuutteja hallinnoivat komponentit.
- Luurankokomponentit, joita käytetään hahmoanimaatioissa.
- Pelikameraa operoivat komponentit.
- Renderöintikomponentit, joilla määritellään hahmojen ja pelimaailman tekstuureja ja visuaalisuutta.
- Fysiikkakomponentit, jotka hallitsevat muun muassa pelimaailman painovoimaa.
- Ääntä toistavat audiokomponentit.
- Npc-hahmojen käyttäytymistä hallinnoivat AI komponentit.

(Unreal Engine g.)

Komponentteja pystytään lisäämään Blueprintissä toimijan omassa editorissa. Editori sisältää komponenttien lisäystoiminnon, jossa määritellään, minkälainen komponentti luodaan. Tämän jälkeen komponentin ominaisuuksia päästään muokkaamaan sen omalla välilehdellä. Myös C++-koodeilla luotuja luokkia pystytään hyödyntämään mukautettuina komponentteina. (Unreal Engine g.)

Yksi komponenttien suurimmista hyödyistä on yhteisten käyttäytymispiirteiden jakamisessa, jolloin toimijoille ei tarvitse yksi kerrallaan ohjelmoida haluttuja ominaisuuksia, vaan riittää, että niille linkitetään tarvittavat ominaisuudet omaava komponentti. Lisäksi komponentit pystyvät keskustelemaan toisten komponenttien kanssa mahdollistaen pelin sujuvan ja monipuolisen toiminnan. (Unreal Engine g.)

Input ja Output

Input objektit muuttavat annetun syötteen ymmärrettäväksi tiedoksi, jota toimijat voivat käyttää. Yleisimpiin inputeihin kuuluu laitteistosyöte, jossa muutetaan pelaajan laitteella, esimerkiksi näppäimistölle, hiirelle tai ohjaimelle antama kehote erilaisiksi tapahtumiksi ja toiminnoiksi, kuten liikkeeksi. Unreal Enginen sisällä pystytään lisäämään ja muokkaamaan haluttuja laitteistosyötteitä, jolloin määritelty syöte toteuttaa suunnitellun toiminnon. Tämä toteutetaan määrittelemällä syöteasetuksia niin, että haluttu näppäin on toiminnassa ja tämän jälkeen luomalla haluttu toiminto Blueprintin kautta. Outputilla tarkoitetaan sitä, mitä

pelaaja näkee ja kokee. Pelaajahahmon liike ja toiminta sekä pelin grafiikat ja äänet ovat outputin näkyvimpiä muotoja. (Unreal Engine h.; Hu 2017.)

Hyvän pelinkehityksen kannalta pelinkehittäjien on tärkeää löytää näiden kahden välinen balanssi. On tärkeää, että pelaaja tuntee hänen syötteensä sujuvan toimivuuden. Input ja output -toimintojen sekä niiden välisen toimivan prosessoinnin on heijastettava sitä, millainen peli on ja miten se haluaa heijastaa omaa maailmaansa. Vuoropohjaisilla videopeleillä, jotka painottavat valinnantekoprosessia on hyvin erilaiset tavoitteet kuin ensimmäisestä persoonasta nähtävillä toimintapeleillä, joissa pelin paino on hiiren ja tähtäyksen virheettömällä yhteistyöllä sekä pelimaailman graafisella näyttävyydellä. (Hu 2017.)

3.3 C++

C++-ohjelmointikielen kehitys alkoi vuonna 1979 Bjarne Stroustrup:n toimesta. Hän halusi kehittää ohjelmointiin soveltuvan kielen, joka sisältäisi olioita ja luokkia. Pohjaksi hän valitsi C-kielen. Siihen Stroustrup:n oli tarkoitus lisätä C-kielen jo olemassa olevien toimintojen lisäksi luokkia, periytymistä, avoimuutta, funktioargumentointia sekä vahvaa koodin tyyppitarkastusta. (Cplusplus)

C++ otettiin ensimmäistä kertaa käyttöön vuonna 1985 vaikkakaan se ei ollut vielä saanut standardiluokitusta. Tämän jälkeen kieltä päivitettiin ja muokattiin useaan otteeseen soveltumaan eri toimintatapojen tarpeisiin. C++ sai vihdoin ensimmäisen virallisen standardiluokituksensa vuonna 1998. Tulevina vuosina kielelle julkaistiin uusia paranneltuja standardeja, jotka ottivat kantaa esiintyneisiin ongelmakohtiin. (Cplusplus)

Samankaltaisuutensa lisäksi muihin C-kieliin C++:n vahvuuksiin ohjelmointikielenä lukeutuvat muun muassa kielen siirrettävyys alustalta toiselle laitteistoista riippumatta, skaalautuvuus tasolta toiselle, olio-ohjelmoinnin olemassaolo, ohjelmoijien annettu täydellinen muistinhallinta sekä vapaus suunnitella ohjelman toiminta halutulla tavalla. (Cheema 2022.)

C++ ei ole kuitenkaan vailla heikkouksia. Se on monimutkainen kieli, joka voi aiheuttaa monenlaisia ongelmia kokemattomalle ohjelmoijalle. Osoittimet voivat aiheuttaa ohjelmoijille ongelmia, jos niitä ei alusteta huolella. Näiden osoitinvirheiden korjaaminen voi olla vaikeaa osaavallekin tekijälle. On myös mahdollista, että koko kieli saattaa korruptoitua, jos muistia käytetään ja hallinnoidaan väärin. Tämä saattaa aiheuttaa mahdollisia tietoturvaongelmia. C++ ei myöskään sisällä roskankerääjiä, jos tähän ei puututa, voivat tarpeettomat tiedot lisätä muistinkulutuksen määrää. (Cheema 2022.)

C++-ohjelmointikieli rakentuu .h ja .cpp -tiedostojen ympärille. Ensimmäiseksi mainitut .h-tiedostot sisältävät jaettavia määrittelyitä, jotka voivat olla sisällytetty useaan eri .cpp-tiedostoon. Näin ollen yksi .h-tiedosto voidaan sisällyttää useaan otteeseen yksittäisen ohjelman käännösprosessissa. Toinen osa C++-kieltä ovat .cpp-tiedostot ja ne sisältävät luokkien toteutuskoodit. (Rush 2022.)

Unreal Enginen mahdollistaa uuden C++-luokan luonnin editorin sisällä, jolloin kyseinen luokka perii suoraan monia C++-koodin ja pelimoottorin yhteistoiminnan kannalta oleellisia toimintoja. Luotuun luokkaan kirjataan halutut toiminnot ja kun se on rakennettu ohjelmointieditorissa, voidaan sitä käyttää Unreal Enginen editoreissa. (Unreal Engine i.)

3.4 Blueprintin ja C++ yhteistoiminta Unreal Enginessä

Unreal Enginellä tehtyjen pelien sisällä olevat asiat voidaan jakaa logiikkaan ja dataan. Logiikkaan kuuluvat ohjeet ja rakenteet, kun taas logiikka käyttää dataa kuvailemaan sitä, mitä ja miten peli tekee asioita. Sekä C++ että Blueprint pystyvät molemmat toimimaan logiikan ja datan parissa itsenäisesti, mutta yhdistelemällä näitä kahta pystytään kehittämään monipuolisempia ja ketterämpiä kokonaisuuksia. (Unreal Engine j.)

Logiikassa on hyvä antaa C++:n luoda muuttujat ja funktiot ja antaa Blueprintin käyttää näitä pelin sisällä editoreissaan. Tämä siksi, että C++ on Blueprinttiä parempi alusta datan arvojen asettamiseen. Blueprint-luokat osaavat myös tukea datan arvojen asettamista ja niillä on hyvä käsitellä dataa paikallisesti ja tallentaa se funktioiden paikallisiin muuttujiin, josta tieto viedään tarvittaessa takaisin C++-olioille. (Unreal Engine j.)

Vaikka Blueprintillä on mahdollista toteuttaa suurin osa pelin toiminnoista, verrattuna C++:aan Blueprint on hitaampi ja se käyttää enemmän muistia (Unreal Engine j.). Tämä ei aina ole huomattavaa, mutta jos Blueprintiä ei ole suunniteltu optimaalisesti, heikentää monimutkaisesti ohjelmoitu järjestelmä pelin suorituskykyä (Epic Games a.).

Blueprint ei myöskään sisällä kaikkia C++:n ominaisuuksia, joita voidaan hyödyntää Unreal Enginen sisällä, eikä kaikkia C++:n ominaisuuksia voida hyödyntää Blueprinteissä. Esimerkkinä tästä on erillään luotu käyttöliittymäkehys, joka ei käytä olioita, joten sitä ei voi hyödyntää Blueprinteissä, vaan on hyödynnettävä C++:aa. Ongelmaksi tämä muodostuu, jos pelin kehitystiimissä ei ole C++-ohjelmointia osaavia kehittäjiä, sillä silloin kehitystiimi ei tiedä mitä moottorin sisällä tapahtuu. Kehittäjät eivätkä pysty muokkaamaan moottorin pohjakoodia soveltumaan heidän tarpeisiinsa tai ratkaisemaan eteen tulevia ongelmatilanteita.

Loppujen lopuksi painotus pelinkehityksessä Blueprintin sekä C++ välillä on täysin riippuvainen kehitystiimin omista tarpeista sekä taitotasosta. (Epic Games a.)

Blueprint ja C++ ovat suunniteltu toimimaan Unreal Enginen sisällä samanaikaisesti. Parhaimmillaan ne tukevat toisiaan ja paikkaavat toistensa heikkouksia, näin auttaen pelinkehittäjiä kehitystyössään. C++ toimii Blueprinttiä paremmin suurien konseptien ja työkalujen luomisessa. Tämä vapauttaa Blueprintin erilaisten assettien, skriptien sekä C++:lla tehtyjen työkalujen hyödyntämisen yksinkertaisten konseptien luomiseen. (Epic Games a.)

Rajapinnat

Unreal Enginessä sekä C++ että Blueprint voivat luoda omia rajapintaluokkiaan, joille pystytään määrittelemään yhteisiä funktioita. Näitä funktioita pystytään tämän jälkeen käyttämään eri luokkien sisällä. (Unreal Engine k.)

Käyttääkseen rajapintoja, luokan on perittävä koko rajapinta ja toteutettava kaikki sen funktiot. C++-koodissa luokalle kirjataan UCLASS-määrittely ja sille tarkennuksena UFUNCTION-makro, jotta rajapinnan toiminnot voidaan toteuttaa. Blueprintissä pystytään luomaan omia rajapintoja pelimoottorin oman Blueprint Interface toiminnon avulla ja sen funktioita voidaan kutsua Blueprintin editorissa. (Epic Games b.)

Kun rajapinta on toteutettu, voidaan sitä käyttää eri luokkien sisällä. Näin luokat, jotka hyödyntävät kyseistä rajapintaa pystyvät myös välittämään tietoa toisilleen. Tämä mahdollistaa monien eri järjestelmien eheän toimivuuden Unreal Enginen sisällä. (Unreal Engine k.)

4 Case: Pelaajahahmon mainejärjestelmä

4.1 Työn taustatiedot

Projekti sisältää kolme erillistä osuutta, jotka nivoutuvat yhdeksi kokonaisuudeksi. Ensimmäinen näistä on pelaajahahmon kyky tervehtiä npc-hahmoja. Toisena osuutena on npc-hahmoille tehty ryhmittymäjärjestelmä, jossa hahmoille pystytään määrittelemään tietty ryhmittymä, johon he kuuluvat. Kolmanneksi käsitellään pelaajahahmon ryhmittymäkohtaista mainetta hallinnoivaa järjestelmää, joka tietää ja vaikuttaa siihen, miten npc-hahmot suhtautuvat pelaajaan. Tervehtimällä ryhmittymien hahmoja, pelaaja muuttaa mainettaan ryhmittymän sisällä.

Last Drop: Legend of Seppo -pelin järjestelmät rakennetaan niin, että raskaimmat ja monimutkaisimmat toiminnot toteutetaan C++-koodien sisällä ja Blueprint-järjestelmät suorittavat kevyemmät toiminnot. Näin varmistetaan, että pelin sisäinen toiminta on tehokasta.

Projektissa käytetään seuraavia pohjia joihin järjestelmät rakentuvat:

- ThirdPersonCharacter on pelaajan pelihahmon toiminnot sisällään pitävä Blueprint pohja. Näihin lukeutuvat muun muassa pelihahmon luuranko, ohjailtava kamera, liikehdinnät ja muita pelihahmolle ohjelmoidut ominaisuudet.
- Animaatio Blueprint, lyhennettynä ABP, hallinnoi pelihahmoa koskevia animaatioita.
- Montaasit ovat animaatioiden hallintaan tarkoitettuja työkaluja, joissa pystytään muokkaamaan ja yhdistelemään yksittäisiä animaatiosarjoja yhdeksi kokonaisuudeksi.
- StatusCharacter on C++-olio, joka pitää sisällään pelaajahahmon tilaa ja statusta koskevia asetuksia ja tietoja, kuten esimerkiksi pelaajan elämäpisteet, voimat sekä juopumustilan.
- CharacterAI on C++-olio, joka käsittelee määritellyt ryhmittymät ja luo pohjan npc-hahmojen reagointiin puhetekstin ja animaation muodossa.
- AIControl on CharacterAI:n ohella toinen ryhmittymien hallintaan vaikuttava C++-olio. Se ottaa npc-hahmon hallintaansa, kun hahmoa tarvitaan vastatervehdystoiminnan aikana.

- BehaviorTree ja Blackboard hallinnoivat npc-hahmojen käyttäytymistä pelimaailmassa. BehaviorTree-kaavioon luodaan solmujen avulla käskyjä, mitä hahmot järjestelmällisesti noudattavat. Blackboard toimii BehaviorTreen tietopankkina, mihin kirjataan kaikki tarvittavat käyttöavaimet kuten esimerkiksi muut hahmot, jotka npc-hahmon on tunnistettava.

Projektin C++-oliot käyttävät monia makroja varmistaakseen, että ne toimivat halutulla tavalla Unreal Enginessä. UCLASS() osoittaa, että luokka on käytettävissä Unreal Enginessä. PROPERTY() tuo määritellyt muuttujat editoriin. Näitä määrittelyjä ovat muun muassa EditAnywhere, joka antaa Blueprintille luvan muokata ominaisuutta ominaisuusikkunoissa, BlueprintReadOnly ja BlueprintReadWrite, jotka määrittelevät, saako ominaisuutta vain lukea Blueprintin kautta vai saako siihen tehdä myös muutoksia, sekä EditDefaultsOnly, joka rajoittaa ominaisuuden muokkaamista editorin ominaisuusikkunoissa ennakkoon määriteltyihin vaihtoehtoihin. UFUNCTION() mahdollistaa kirjattujen funktioiden käytön Blueprintissä. USTRUCT() on tietorakenne, jolla järjestellään ominaisuuksia. Sillä ryhmitellään erityyppisiä muuttujia yhdeksi tyypiksi. Tämä määrittely mahdollistaa kyseisten rakenteiden tuonnin ja käytön Blueprintissä BlueprintType-tarkenteen avulla. UENUM eli enumeration muuttaa kokonaislukuarvoja nimiksi, mahdollistaen helposti ymmärrettävän listamäärittelyn nimillä.

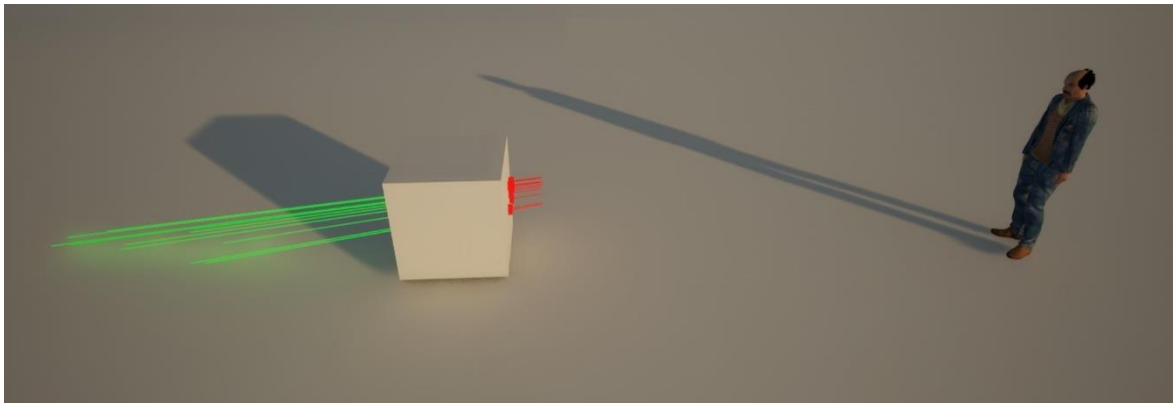
Animaatioiden ja audioiden luonnit eivät kuulu työn piiriin, vaan työssä käytetään projektiin tehtyjä komponentteja. Työn tarkoitus on luoda pohjarakenteet, joihin pystytään myöhemmin lisäämään haluttuja materiaaleja.

4.2 Tervehtiminen

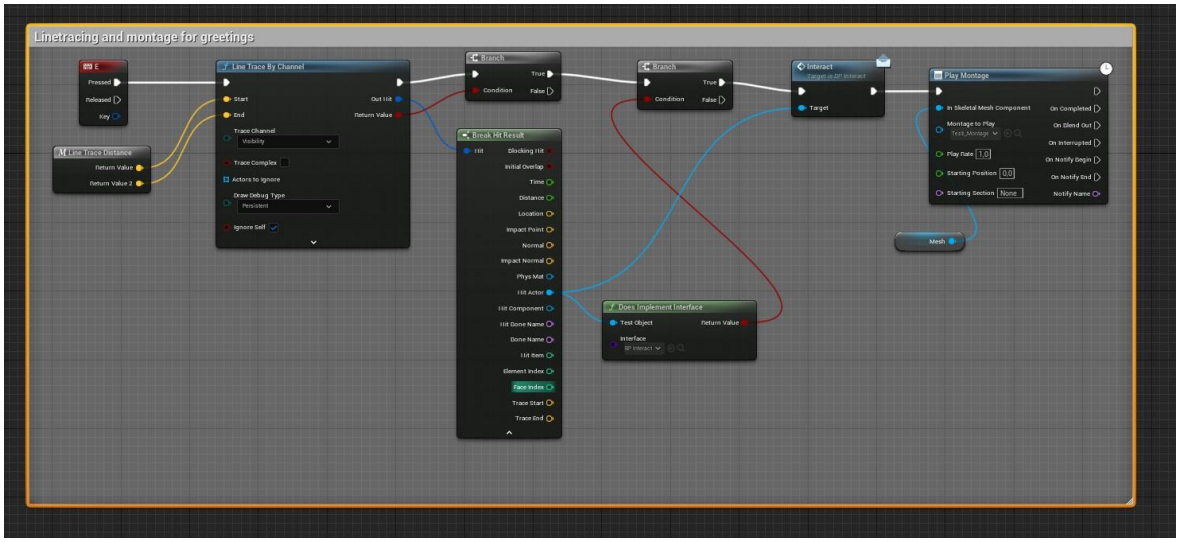
4.2.1 Hahmon tervehtiminen

Tervehtimisjärjestelmän mekaniikka aloitetaan tekemällä pelaajahahmon ThirdPersonCharacterin Blueprinttiin linjajäljittimen etäisyys -makro, joka ohjelmoidaan toimimaan, kun pelihahmo on pelimaailmassa 5–10 metrin etäisyydellä kohteestaan. Linjajäljitin on ominaisuus, jossa pelimoottori luo viivasegmentin ja tarkastaa, osuuko viiva haluttuun objektiin. Kuva 5 näyttää testimaailmassa olevan linjajäljittimen, jonka piirtämät viivat ovat laitettu näkyviksi. Niin kauan kuin linjajäljitin ei ole osunut sopivaan objektiin, näkyy viiva punaisena. Osuttuaan objektiin, vaihtaa viiva väriään vihreäksi kertoakseen onnistuneesta osumisesta. Tälle jäljitinmakrolle määritellään tarvittavat etäisyyslaskennat ja -mittaukset sekä muut

parametrit, joita sen halutaan noudattavan, kuten esimerkiksi pelaajan tähtäyspiste. Makrosta käsin pelin kehittäjät pystyvät halutessaan muuttamaan linjajäljittimen käyttämiä arvoja. Jos linjajäljitin aktivoituessaan osuu npc-hahmoon, aloittaa se pelaajan tervehtimiseen luodun animaatioliikettä, sekä ääntä yhdistelevän montaasin. Kuvassa 6 näkyy, kuinka kaikki edellä mainitut toiminnot löytyvät yhdestä Blueprint-solmusta. Solmusarja alkaa vasemmalta toimintanäppäimen näppäinkehotteesta, josta käsky etenee linjajäljittimelle. Myös tehty makro tuo etäisyysmäärittelynsä linjajäljitinsolmulle. Tästä komento siirtyy varmistamaan, että linjajäljitin osuu hyväksyttävään kohteeseen. Jos kohde löytyy, solmusarja päättyy montaasin aloittamiseen.



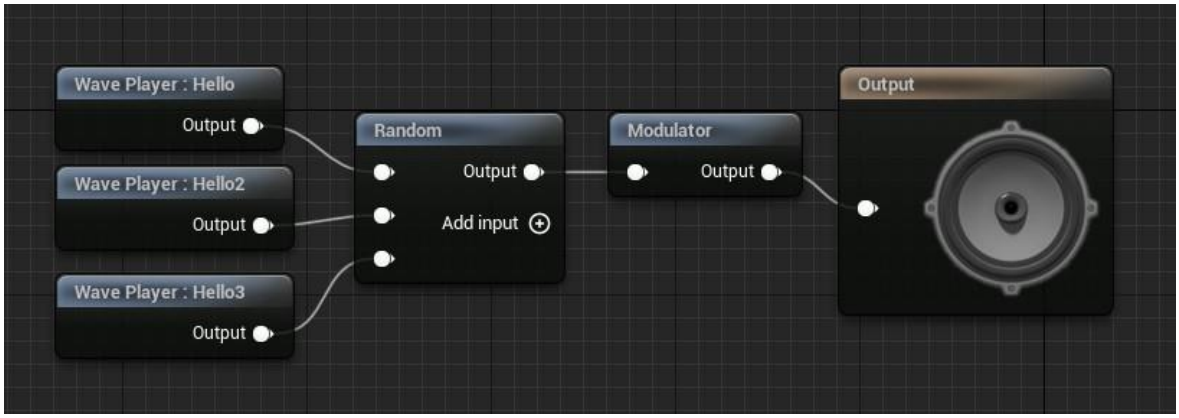
Kuva 5. Näkyvä linjajäljitin testimaailmassa (Jari Mäenpää)



Kuva 6. Linjajäljitin ja tervehtimismontaasi samassa solmussa (Jari Mäenpää)

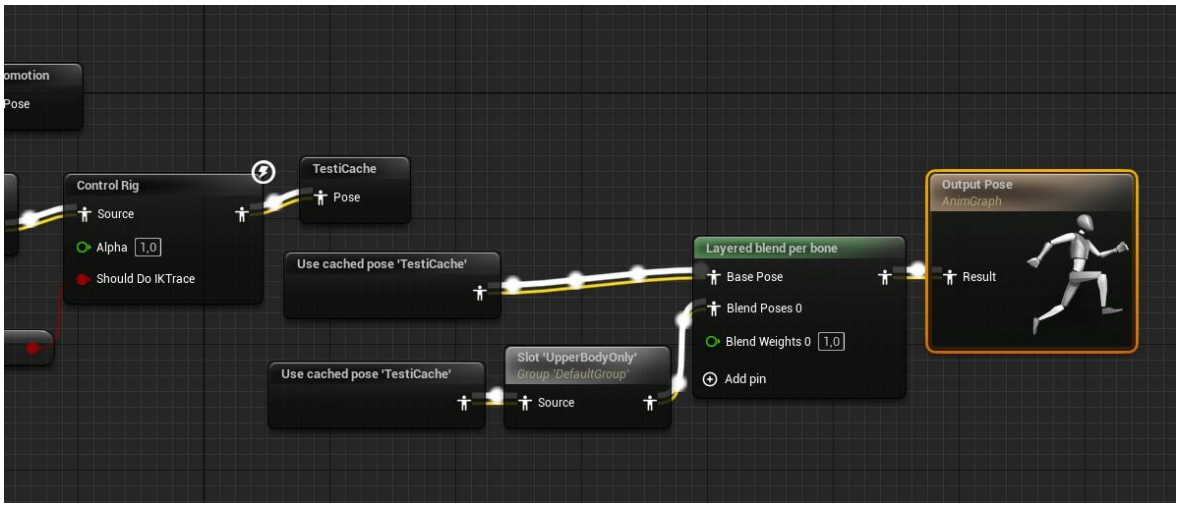
4.2.2 Tervehtimisanimaatio

Tervehtimisanimaatiota varten Unreal Engineen luodaan audioraitoja hallinnoiva Sound Cue -objekti (kuva 7). Sound Cue on toiminto, johon voidaan koota pelissä käytettäviä ääniä ja muokata niitä käyttäytymään halutulla tavalla. Tähän Sound Cue -objektiin lisätään tarvittavat käyttötarkoitukseen tehdyt tervehdyssäänet ja niiden valikoitumismuodoksi asetetaan satunnaisuus, jotta tervehdittäessä pelaajahahmo ei vain toista yhtä ja samaa audioraitaa. Tehty Sound Cue lisätään haluttuun kohtaan käytetyn tervehtimisanimaatioliikkeen aikajanaa, sillä animaatioiden Blueprinteissä pystytään itse liikkeen lisäksi lisäämään ja muokkaamaan myös muita liikkeeseen sisältyviä materiaaleja, kuten ääniä. Tätä kautta animaatiot käynnistävät itsenäisesti Sound Cue -objektin ja aktivoivat liikkeen lisäksi myös tervehtimisen audio-osuuden.



Kuva 7. Esimerkki Sound Cue -objektista (Jari Mäenpää)

Itse tervehtimisanimaatiossa hyödynnetään Layered Blend Per Bone toimintoa (kuva 8). Layered Blend Per Bone lisätään hahmon tervehtimisanimaation Blueprinttiin. Layered blend Per Bone -toiminto mahdollistaa määrittelyn sille, mikä osa hahmon luurangosta tai ruumiista suorittaa määritellyn animaation, pitäen muut luurangon osat vapaina suorittamaan muita toimintoja. Monia eri Layered Blend Per Bone -toimintoja pystytään tarvittaessa lisäämään samaan animaatioon näin mahdollistaen tilannekohtaisesti monien eri toimintojen yhtäaikaisen tapahtumisen. Tervehtimisanimaatioissa vain hahmon yläruumiille animoidaan liikettä, jotta tervehtimisanimaatiot pystytään toteuttamaan myös hahmon liikkuessa. Lopuksi luodaan montaasipohja, johon tervehtimisanimaatio liitetään. Tämä montaasi kytketään pelaajahahmon ThirdPersonCharacteriin linjajäljitintoiminnon jatkeeksi, jolloin se aktivoituu, kun sopiva kohde on löytynyt. Kun tervehtiminen on suoritettu, lähettää pelimoottori tiedon tervehtimisestä pelaajan mainetta hallinnoivalle CharacterAI C++-oliolle.



Kuva 8. Layered Blend Per Bone (Jari Mäenpää)

4.3 Ryhmittymät

Pelimaailman eri kaupunginosille tulee olemaan omat ryhmittymänsä, jotka hallitsevat omia alueitaan. Tietyt pelialueet tulevat suhtautumaan lähtökohtaisesti positiivisemmin pelaajaan kuin toiset. Tervehtimisen mukanaan tuoma hyvä maine tulee antamaan pelaajalle tiettyjä etuuksia. Hän pystyy esimerkiksi ostamaan tavaroita halvemmalla kauppiailta, sekä tietyt tehtävät aukeavat mahdollisesti vasta kun pelaajan ja ryhmän välinen maine on saavuttanut tietyn tason. Pelaajan tervehtiessä npc-hahmoja, jotka kuuluvat pelin antagonistien ryhmään, suuttuvat hahmot pelaajalle. Tämä johtaa pelaajan maineen laskuun kyseisen ryhmittymän silmissä.

C++-puolella luotu CharacterAI hallinnoi ryhmittymäjärjestelmän toimivuutta. CharacterAI saa tiedon tervehtimisestä ThirdPersonCharacter Blueprintiltä ja sen oleellisin toiminto on määritellä ryhmittymät ja luoda vastauspohja npc-hahmojen reagointiin puhetekstin, animaation tai näiden yhdistelmän muodossa. Nämä vastaukset määräytyvät sen mukaan, onko pelaaja maine ryhmään positiivinen, neutraali vai negatiivinen.

CharacterAI.h-olioon luodaan UENUM-lista, johon kaikki pelinsisäiset ryhmittymät kirjataan (kuva 9). Tätä listaa tullaan kutsumaan pelihahmojen Blueprint-luokissa ja sen avulla voidaan määritellä jokaiselle halutulle npc-hahmolle rytmittymä. Kuva 10 näyttää kirjatut EditDefaultsOnly rajoitteiset UPROPERTY-muuttujat, jotka mahdollistavat hahmojen tervehtimisvastauksen muuttamisen Blueprintissä ja joita AIControl tulee käyttämään välittäessään npc-hahmon vastatervehdyksen pelaajalle.

```
// Add all the factions

FT_Player UMETA(DisplayName = "Player"),
FT_None UMETA(DisplayName = "None"),
FT_Faction1 UMETA(DisplayName = "Faction 1"),
FT_Faction2 UMETA(DisplayName = "Faction 2"),
FT_Faction3 UMETA(DisplayName = "Faction 3")
```

Kuva 9. Lista ryhmittymistä (kuva: Jari Mäenpää)

```
UPROPERTY(EditDefaultsOnly)
    UTextRenderComponent* ConversationText;

UPROPERTY(EditDefaultsOnly)
    TArray<EFactionType> FriendlyFactions;

UPROPERTY(EditDefaultsOnly)
    TArray<EFactionType> NeutralFactions;

UPROPERTY(EditDefaultsOnly)
    TArray<EFactionType> HostileFactions;

UPROPERTY(EditDefaultsOnly)
    TArray<FString> FriendlyConversationStrings;

UPROPERTY(EditDefaultsOnly)
    TArray<FString> NeutralConversationStrings;

UPROPERTY(EditDefaultsOnly)
    TArray<FString> HostileConversationStrings;
```

Kuva 10. UPROPERTY-muuttujat (kuva: Jari Mäenpää)

StatusCharacter-olio hakee CharacterAI:lta ryhmittymän määrittelytiedot, jotta pelaajahahmolle pystytään asettamaan myös ryhmittymä (kuva 11). Pelaajahahmo kuuluu omaan ryhmittymäänsä. Tämä helpottaa määrittelemään pelaajan ja ryhmittymän suhteen.

```
// Faction
void AStatusCharacter::SetFaction(EFactionType NewType)
{
    Faction = NewType;
}
```

Kuva 11. Pelaajan ryhmittymän määrittelyn alustus StatusCharacterissa (Jari Mäenpää)

4.4 Mainejärjestelmä

Kun CharacterAI on käsitellyt tiedon tervehtimisestä, lähettää se tiedon eteenpäin StatusCharacterille ilmoittaen, että maineeseen pitää tehdä muutos. StatusCharacter C++-olio sisältää pelaajahahmon omaavia attribuutteja, kuten esimerkiksi hänen voimansa, kestävyytensä sekä rahamääränsä. Näiden joukkoon luodaan myös pohja pelaajan maineeseen eri ryhmittymien sisällä. StatusCharacter pitää sisällään pelaajan maineiden aloitus- ja nykyarvot ryhmittymissä, sekä lasku- ja muuntotoimitukset mainepisteiden ja -tasojen muuttamiseen. Kun StatusCharacter on tehnyt muutokset ryhmittymän maineeseen, lähettää se tiedon päivitetystä maineesta CharacterAI:lle vastatervehdystä varten.

StatusCharacter.h-olioon määritellään kaikki arvostuksen eri tasot UENUM-luokkaan, sekä tasojen pistemäärät bit shift -tekniikalla. Bit shift, eli bittisiirto, on tekniikka, missä bittejä siirretään oikealle tai vasemmalle jakamalla tai kertomalla luku 2 potenssilla. Tämä on tehokas tapa toimintojen suorittamiseksi. Tätä käytetään hyväksi mainetason muutoksissa (kuva 12).

```
UENUM(BlueprintType)
enum ECharacterReputationState {
    CS_HATE = -1 << 1        UMETA(DisplayName = "CS_RS_Hate"),
    CS_DISLIKE = -1 << 0     UMETA(DisplayName = "CS_RS_Dislike"),
    CS_NEUTRAL = 0           UMETA(DisplayName = "CS_RS_Neutral"),
    CS_LIKE = 1 << 0         UMETA(DisplayName = "CS_RS_Like"),
    CS_LOVE = 1 << 1         UMETA(DisplayName = "CS_RS_Love")
};
ENUM_CLASS_FLAGS(ECharacterReputationState);
```

Kuva 12. Arvostuksen tasot (kuva: Jari Mäenpää)

USTRUCT-luokan sisään luodaan maineen vaihdossa käytetyt parametrit (kuva 13). Kuvassa 14 näkyy tasojen määritellyt pistearvot. Näiden arvojen pisteytys kulkee välillä 0–100, jossa alin arvo 0 kuvastaa huonointa mainetta ja ylin arvo 100 korkeinta. Tämän avulla funktio tietää, milloin sen pitää muuttaa arvostuksen tasoa, kun se nousee tai laskee tietyn pisterajan ohitse pelaajan kerättyä tai menetettyä riittävästi tervehtimisessä keräämiään tai menettämiään pisteitä. Kun StatusCharacter muuttaa ryhmittymän mainepisteitä, tämä toiminto tapahtuu getter ja setter -funktoiden avulla, jossa funktio hakee uuden tiedon pistemuutoksesta, muuttaa ryhmittymän pisteitä laskutoimituksen kautta ja asettaa uuden pistemäärän pelaajan maineeksi ryhmittymässä.

```
USTRUCT(BlueprintType)
struct FCharacterReputationChangeParams
    /*
    Reputation state
    */
{
    GENERATED_BODY()

    UPROPERTY(EditAnywhere, BlueprintReadWrite)
    TEnumAsByte<ECharacterReputationState> CS_RepState;
    int32 RepValue;

    FCharacterReputationChangeParams() :
        RepValue(0),
        CS_RepState(ECharacterReputationState::CS_NEUTRAL)
    {}
    FCharacterReputationChangeParams(ECharacterReputationState status, int32 value) :
        RepValue(value),
        CS_RepState(status)
    {}
};
```

Kuva 13. Pelaajahahmon mainetason vaihtoparametrit (Jari Mäenpää)

```
UPROPERTY(VisibleAnywhere, BlueprintReadOnly, meta = (AllowPrivateAccess = "true"))
    float StartingKarma = 50.0;

UPROPERTY(VisibleAnywhere, BlueprintReadOnly, meta = (AllowPrivateAccess = "true"))
    float CurrentKarma;

UPROPERTY(VisibleAnywhere, BlueprintReadWrite, meta = (AllowPrivateAccess = "true"))
    int MinKarmaToNeutral = 26.0;

UPROPERTY(VisibleAnywhere, BlueprintReadWrite, meta = (AllowPrivateAccess = "true"))
    int MaxKarmaToNeutral = 74.0;

UPROPERTY(VisibleAnywhere, BlueprintReadWrite, meta = (AllowPrivateAccess = "true"))
    float MinKarmaToLikeValue = 75.0;

UPROPERTY(VisibleAnywhere, BlueprintReadWrite, meta = (AllowPrivateAccess = "true"))
    float MinKarmaToLoveValue = 99.0;

UPROPERTY(VisibleAnywhere, BlueprintReadWrite, meta = (AllowPrivateAccess = "true"))
    float MinKarmaToDislikeValue = 25.0;

UPROPERTY(VisibleAnywhere, BlueprintReadWrite, meta = (AllowPrivateAccess = "true"))
    float MinKarmaToHateValue = 1.0;
```

Kuva 14. Arvostuksen tasojen määritellyt pistearvot (kuva: Jari Mäenpää)

StatusCharacter.cpp sisältää ainoastaan määrittelyparametrit maineen vaihtoon. Parametrit kertovat StatusCharacter.h:lle mihin uuteen maineen tasoon pelaajan maine pitää päivittää muutoksen ollessa edessä (kuva 15).

```
void AStatusCharacter::ChangeCharacterReputation(FCharacterReputationChangeParams params)
{
    CurrentKarma += params.RepValue;

    if (CurrentKarma <= MinKarmaToHateValue)
        CS_ReputationState = ECharacterReputationState::CS_HATE;
    else if (CurrentKarma <= MinKarmaToDislikeValue)
        CS_ReputationState = ECharacterReputationState::CS_DISLIKE;
    else if (CurrentKarma <= MaxKarmaToNeutral && CurrentKarma >= MinKarmaToNeutral)
        CS_ReputationState = ECharacterReputationState::CS_NEUTRAL;
    else if (CurrentKarma >= MinKarmaToLikeValue)
        CS_ReputationState = ECharacterReputationState::CS_LIKE;
    else if (CurrentKarma >= MinKarmaToLoveValue)
        CS_ReputationState = ECharacterReputationState::CS_LOVE;
}
```

Kuva 15. Maineen tason vaihto (kuva: Jari Mäenpää)

4.5 Vastatervehdys

CharacterAI:n C++-olio ja Unreal Enginessä CharacterAI:n pohjalta luotu Blueprint-luokka mahdollistavat määrittelyn sille, mihin ryhmittymään npc-hahmo kuuluu ja miten tietokoneen ohjaama hahmo vastaa tervehdykseen puheen, puhetekstin ja animaation avulla.

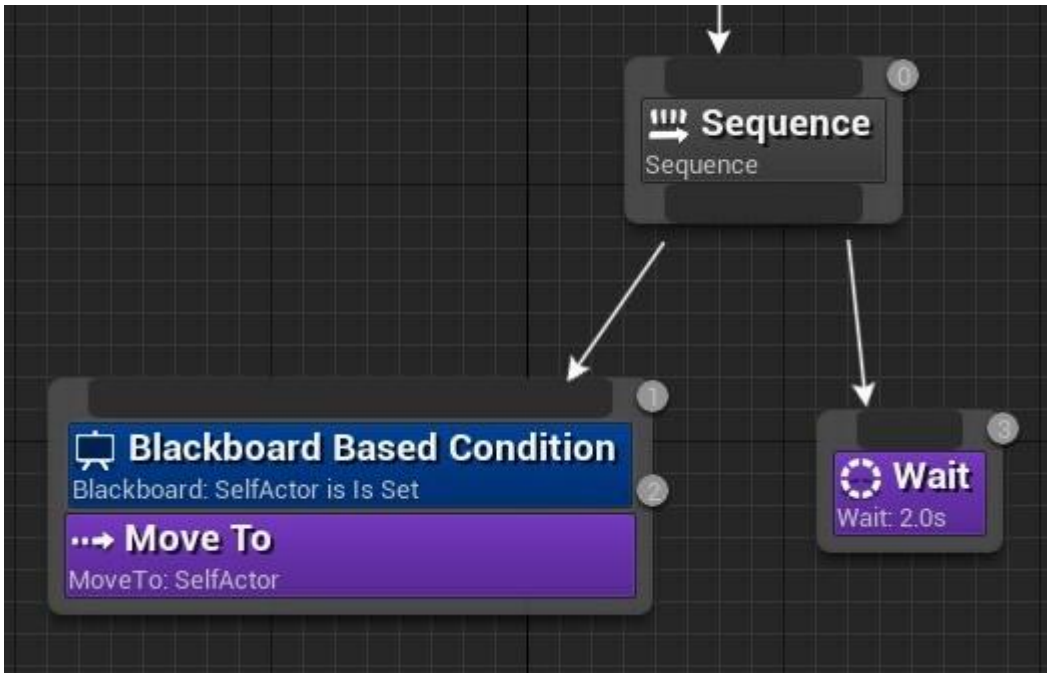
Kun pelaaja on suorittanut tervehtimisanimaationsa ja saatuaan päivitetyn tiedon pelaajan maineesta StatusCharacterilta, ilmoittaa CharacterAI tästä AIControl:lle. AIControl:n tehtävä on hallinnoida npc-hahmoja tervehtimisen aikana ja mahdollistaa vastatervehtimiset.

AIControl.h kontrolloi npc-hahmoja OnPossess toiminnon avulla. OnPossess toiminto ohittaa jo ohjelmoidut toiminnot ja pakottaa määrätyn hahmon hallintaansa. Oliossa on referenssit Blackboard- ja BehaviorTree -komponentteihin, jotta Blueprintin puolella pystytään määrittelemään miten hahmot käyttäytyvät.

AIControl.cpp:n tehtävä on antaa tekoälylle pääsyn tehtyihin Blackboard ja BehaviorTree -komponentteihin. Se alustaa edellä mainitut komponentit ja OnPossess toiminto antaa npc-hahmoille pääsyn komponentteihin ensin varmistaen, että ohjelmoidut käskyt ovat paikkansapitäviä. AIControl:ssa myös luodaan Blackboardiin pelaajan hahmoon viittaava merkintä, jotta sitä voidaan hyödyntää BehaviorTreen sisällä editorissa.

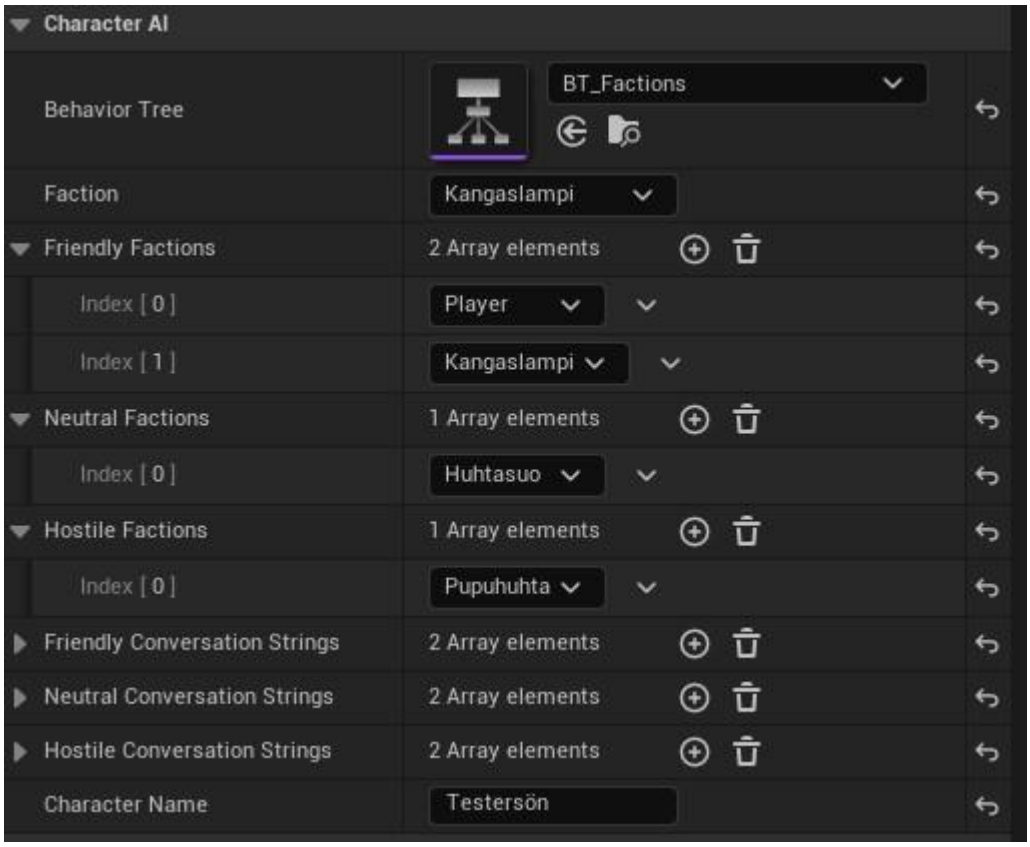
CharacterAI sekä AIControl -olioiden sisällä pohjustettuihin BehaviorTree- ja Blackboard -komponentteihin pystytään tekemään Blueprintissä muutokset npc-hahmon käyttäytymiseen. Blackboard on BehaviorTreen sisarkomponentti, johon määritellään tietoja, mitä BehaviorTree pystyy käyttämään omassa toiminnassaan. Tässä tapauksessa Blackboard-komponenttiin luodaan objektiavain nimeltään Player ja asetetaan sen luokaksi Actor, jotta Blueprint ymmärtää C++-olioissa määritellyt toiminnot. BehaviorTree on

toimintokomponentti, jolla voidaan kertoa tekoälylle, mitä sen pitää tehdä missäkin tilanteessa. Vastatervehdystä varten BehaviorTree ohjelmoidaan niin, että npc-hahmo liikkuu pelaajaa kohti ja odottaa määritellyn ajan ennen kuin se vastaa pelaajalle (kuva 16).



Kuva 16. Npc-hahmon yksinkertainen BehaviorTree (Jari Mäenpää)

Hahmot käyttävät erilaisiin tilanteisiin tarkoitettuja animaatioita ja repliikkejä riippuen siitä, vastaavatko he positiiviseen vai negatiiviseen sävyyn pelaajalle. Kuvassa 17 näkyy, kuinka Blueprintissä on määritelty hahmon ystävälliset, neutraalit sekä vihatut ryhmittymät. Samaan osioon pystytään kirjaamaan eri tilanteisiin sopivia vastauksia. Positiiviseen sävyyn tapahtuneet tervehdykset aiheuttavat vastatervehdyksen kädenheilautuksen ja tervehdysaudion tai -tekstin muodossa, kun taas negatiiset reaktiot laittavat npc-hahmon huutamaan pelaajalle törkeyksiä ja näyttämään esimerkiksi keskisormea. Samoin kuin pelaajahahmon tervehdystoiminnossa, npc-hahmojen omiin Blueprinttiin ohjelmoidaan montaasi, joka aktivoituu pelaajan tervehtiessä heitä. Montaaseja on jokaiselle erilaiselle tervehtimistyypille omansa ja Blueprint saa tiedon siitä, miten hänen pitää tervehdykseen vastata CharacterAI-oliossa.



Kuva 17. CharacterAI ikkuna npc-hahmon Blueprintissä (Jari Mäenpää)

5 Yhteenveto ja pohdinta

Opinnäytetyön tavoitteena oli luoda Pixtell oy:n kehittämään Last Drop: Legend of Seppo -tietokonepeliin pelaajan mainetta hallinnoiva operointijärjestelmä. Osana laajempaa mainejärjestelmää luotiin myös tervehtimisestä sekä ryhmittymistä vastaavat järjestelmät. Työssä tutkittiin pelin alustana toimivan Unreal Enginen historiaa, pelimoottorin omaa visuaaliseen koodaukseen perustuvaa Blueprint-järjestelmää ja sen ominaisuuksia sekä pelimoottorille tärkeimpiä ominaisuuksia sen käyttämässä C++-ohjelmointikielessä.

Työssä pyrittiin luomaan mahdollisimman vähän pelimoottoria rasittava järjestelmäkokonaisuus, jota olisi myös helppo muokata ja käyttää. Järjestelmien tulisi toimia myös hyvin yhdessä muun pelikokonaisuuden kanssa, luoden näin parhaan mahdollisen osan isompaa kokonaisuutta.

Työn käytännön osuus jakautui kolmeen osaan, tervehtimis-, ryhmittymä- sekä mainejärjestelmiin. Tervehtimisjärjestelmä toteutettiin Blueprintillä ja siinä pelaaja pystyy tervehtimään tietokoneen ohjaamaa pelihahmoa animaatioiden sekä audioiden avulla. C++-koodilla tehty ryhmittymäjärjestelmä hallinnoi pelimaailmassa olevia pelihahmoista koostuvia ryhmittymiä ja itse pelihahmoja, kun he vastaavat pelaajan tervehdykseen. Työn pääasiallinen tavoite eli mainejärjestelmä toteutettiin C++-koodeilla ja se pitää sisällään pelaajan maineet eri ryhmittymien sisällä ja nostaa tai laskee mainetta pelaajan tervehtiessä muita pelihahmoja.

Järjestelmien lopullista toimivuutta ei pystytty testaamaan peliympäristössä, sillä osa koodeista oli kirjoitettu StatusCharacter-nimiseen C++-olioon, jota ei vielä ole jalkautettu itse peliin. Kuitenkin ne osat, mitkä toimivat ilman StatusCharacteria, toimivat tarkoitetulla tavalla. Mainejärjestelmäkokonaisuuden kehitystyö jatkuu opinnäytetyön kirjoittamisen jälkeen ja se saatetaan päätökseen mahdollisimman nopealla aikataululla.

Projektissa luodut järjestelmät ovat helposti luotavissa ja muokattavissa sopimaan myös muihin Unreal Engineä hyödyntäviin peliprojekteihin. Jo opinnäytetyön ideointivaiheessa todettiin, että The Last Drop: Legend of Seppo -pelin mainejärjestelmään olisi tarkoitus kiinnittää muita ominaisuuksia, kuten esimerkiksi kaupankäynnissä saatavia alennuksia. On myös mahdollista, että pelin kehityksen edetessä ja kehittyessä, mainejärjestelmääkin muokataan ja kehitetään entisestään pelin kehityksen edetessä.

Lähteet

Cheema, A. 2022. Advantages and Disadvantages of C++. Viitattu 8.3.2023. Saatavissa https://techvidvan.com/tutorials/cpp-pros-and-cons/

Cplusplus. History of C++. Viitattu 8.3.2023. Saatavissa https://cplusplus.com/info/history/

Doucet, L. & Pecorella, A. 2021. Game engines on Steam: The definitive breakdown. Viitattu 4.3.2023. Saatavissa https://www.gamedeveloper.com/business/game-engines-on-steam-the-definitive-breakdown

iD Tech. 2016. 8 Reasons Why Unreal Engine is Great for Beginners & Aspiring Game Designers. Viitattu 3.3.2023. Saatavissa https://www.idtech.com/blog/8-reasons-why-unreal-engine-is-great-for-aspiring-game-designers

Epic Games a. 2022. Mixing Blueprints and C++. Viitattu 27.3.2023. Saatavissa https://dev.epicgames.com/community/learning/tutorials/YDpo/mixing-blueprints-and-c%2B%2B

Epic Games b. 2022. Interfaces (BP + C++). Viitattu 12.4.2023. Saatavissa https://dev.epicgames.com/community/learning/tutorials/bLXe/interfaces-bp-c

Hu, S. 2017. Continuity and Discreteness: How Input Methods Influence Gaming Experiences. Viitattu 1.4.2023. Saatavissa https://www.gamedeveloper.com/design/continuity-and-discreteness-how-input-methods-influence-gaming-experiences

Rush, R. 2022. What is the difference between a .cpp file and a .h file? Viitattu 10.4.2023. Saatavissa https://copyprogramming.com/howto/what-is-the-difference-between-a-cpp-file-and-a-h-file

Schultz, W. Unreal Engine 4 – First Look. Viitattu 6.3.2023. Saatavissa https://web.archive.org/web/20120524062935/http://gameindustry.about.com/od/trends/a/Unreal-Engine-4-First-Look.htm

Sirani, J. 2017. Unreal Engine4 is Free for Everyone. IGN. Viitattu 3.3.2023. Saatavissa https://www.ign.com/articles/2015/03/02/unreal-engine-4-is-free-for-everyone

Thomsen, M. 2012. History of the Unreal Engine. IGN. Viitattu 2.3.2023. Saatavissa https://www.ign.com/articles/2010/02/23/history-of-the-unreal-engine

Unity. General subscription FAQ. Viitattu 6.5.2023. Saatavissa https://unity.com/faq

Unreal Engine a. UnrealScript Language Reference. Viitattu 6.3.2023. Saatavilla https://docs.unrealengine.com/udk/Three/UnrealScriptReference.html#Design%20goals%20of%20_UnrealScript

Unreal Engine b. Frequently Asked Questions (FAQs). Viitattu 6.5.2023. Saatavissa https://www.unrealengine.com/en-US/faq

Unreal Engine c. Blueprint Overview. Viitattu 5.3.2023. Saatavissa https://docs.unrealengine.com/4.27/en-US/ProgrammingAndScripting/Blueprints/Overview/

Unreal Engine d. Nodes. Viitattu 11.5.2023. Saatavissa https://docs.unrealengine.com/4.26/en-US/ProgrammingAndScripting/Blueprints/UserGuide/Nodes/

Unreal Engine e. Blueprint Editor Reference. Viitattu 30.4.2023. Saatavissa https://docs.unrealengine.com/4.27/en-US/ProgrammingAndScripting/Blueprints/Editor/

Unreal Engine f. Macros. Viitattu 3.4.2023. Saatavissa https://docs.unrealengine.com/5.0/en-US/macros-in-unreal-engine/

Unreal Engine g. Components. Viitattu 3.4.2023. Saatavissa https://docs.unrealengine.com/5.0/en-US/components-in-unreal-engine/

Unreal Engine h. Input Overview. Viitattu 2.4.2023. Saatavissa https://docs.unrealengine.com/5.0/en-US/input-overview-in-unreal-engine/

Unreal Engine i. Programming Quick Start. Viitattu 11.4.2023. Saatavissa https://docs.unrealengine.com/5.0/en-US/unreal-engine-cpp-quick-start/

Unreal Engine j. Balancing Blueprint and C++. Viitattu 31.3.2023. Saatavissa https://docs.unrealengine.com/4.27/en-US/Resources/SampleGames/ARPG/BalancingBlueprintAndCPP/

Unreal Engine k. Interfaces. Viitattu 12.4.2023. Saatavissa https://docs.unrealengine.com/4.27/en-US/ProgrammingAndScripting/GameplayArchitecture/Interfaces/