LAB University of Applied Sciences

Käyttäytymislogiikka NPC-hahmolle

LAB-ammattikorkeakoulu
Insinööri (AMK), tieto- ja viestintätekniikka
2025
Niko Salminen

Tiivistelmä

Tekijä(t)	Julkaisun laji	Valmistumisaika
Niko Salminen	Opinnäytetyö, AMK	2025
	Sivumäärä 28	

Työn nimi

Käyttäytymislogiikka NPC-hahmolle

Tutkinto ja koulutusala

Insinööri (AMK), tieto- ja viestintätekniikka

Toimeksiantajaorganisaatio (jos opinnäytetyöllä on toimeksiantaja)

Lohkare Games Oy

Tiivistelmä

Opinnäytetyön tavoitteena oli kehittää NPC-hahmolle käyttäytymislogiikkaa. Työ toteutettiin Unreal Engine 5 pelimoottorissa hyödyntäen sen Blueprints Visual Scripting, Behavior Tree ja AI Perception järjestelmiä.

Opinnäytetyössä perehdyttiin esimerkkeihin olemassa olevien pelien NPC-hahmoista, sekä pohdittiin niiden merkitystä pelattavuuden kannalta. Työssä tutkittiin myös Unreal Enginen ominaisuuksia ja vertailtiin sitä kilpailijaansa Unityyn.

Opinnäytetyön lopputuloksena oli prototyyppi NPC-hahmo, joka vaeltaa ympäriinsä ja pelaajahahmon nähdessään ryhtyy jahtaamaan tätä. NPC palaa vaeltamaan jahtaamisen jälkeen, mikäli näköyhteys pelaajahahmoon katkeaa määritellyksi ajaksi.

Asiasanat

Unreal Engine, Blueprint, Behavior Tree, NPC

Abstract

Author(s)	Type of Publication	Published
Niko Salminen	Thesis, UAS	2025
	Number of Pages	
	28	

Title of Publication

Behavior Logic for an NPC

Degree, Field of Study

Bachelor of Engineering, Information and Communication Technology

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

Lohkare Games Oy

Abstract

The goal of the thesis was to develop behavior logic for a non-player-character (NPC). The project was done in Unreal engine 5, utilizing Blueprints Visual Scripting, Behavior Tree and AI Perceptions systems.

Examples of other games' NPCs and their effects on gameplay were contemplated. Unreal Engine's features were researched and compared to its competitor Unity.

The result of the thesis was a prototype NPC, which wanders around until it spots a player character. After successfully sensing a player character, the NPC starts chasing it. If visual contact is broken for a time, the NPC gives up chasing and returns to wandering around.

Keywords

Unreal Engine, Blueprint, Behavior Tree, NPC

Sisällys

1 Johdanto

Lyhenne NPC tulee englannin kielen termistä non-player-character ja sillä tarkoitetaan videopeleissä etukäteen ohjelmoituja hahmoja, joita pelaaja ei suoraan hallitse. Peleissä NPC:t ovat tärkeitä pelin tarinan ja sen maailman eloon tuomisen välineitä. Ne voivat myös olla osa pelin pelattavuutta esimerkiksi vastustajien, liittolaisten tai neutraalien osapuolien rooleissa. Tässä opinnäytetyössä keskiössä tulee olemaan NPC-hahmot pelattavuuden näkökulmasta.

Työn toimeksiantaja on maaliskuussa 2024 perustettu pelialan yritys Lohkare Games, jonka kehittämään peliin opinnäytetyön lopputulosta voidaan soveltaa. Työn tavoitteena on suunnitella Unreal Enginessä logiikka NPC-hahmolle, joka käyttäytyy pelaajan hahmon ottaman velan mukaan.

Opinnäytetyön tarkoitus ei ole toteuttaa täysin valmista NPC-hahmoa, vaan työssä keskitytään yksittäiseen toiminnallisuuteen, jota toimeksiantaja voi itse soveltaa haluamallaan tavalla. Hahmon käyttäytyminen toteutetaan Unreal Engine 5 pelimoottorissa hyödyntäen Blueprints Visual Scripting, Behavior Tree ja AIPerception järjestelmiä.

2 Esimerkkejä eri pelien NPC-hahmoista

2.1 Rockstar Games

Yksi tunnetuimmista esimerkeistä realistisesti käyttäytyvistä NPC-hahmoista on ehdottomasti Rockstar Gamesin suunnattoman suosituissa Grand Theft Auto -pelisarjassa, jossa elävältä tuntuva maailma on oleellinen osa sarjan pelejä. Pelit pyrkivät imitoimaan oikean elämän kaupunkeja hyvinkin tarkasti, ja kaupunkilaiset ovat luonnollisesti tärkeä osa kokonaisuutta.

Sarjan tuoreimmassa pelissä Grand Theft Auto V (2013) kaupunkilaiset reagoivat pelaajan tekemiin asioihin esimerkiksi juoksemalla karkuun, menemällä suojaan tai soittamalla hätänumeroon. Jotkin NPC-hahmot voivat myös käyttäytyä aggressiivisesti pelaajahahmoa kohtaan esimerkiksi jatkuvasti tönityksi tullessaan tai kolarin sattuessa. NPC-hahmoja on lukuisia erilaisia, kodittomista henkilöistä jengien jäseniin. Samaan luokkaan kuuluvat NPC:t ovat usein samankaltaisia keskenään yleisen käytöksen ja vaarallisuuden tason kannalta. NPC-hahmot voivat myös kantaa aseita, ja tällaisten hahmojen ärsyttäminen voi olla pelaajan hahmolle hyvinkin vaarallista. (GTA Wiki.)

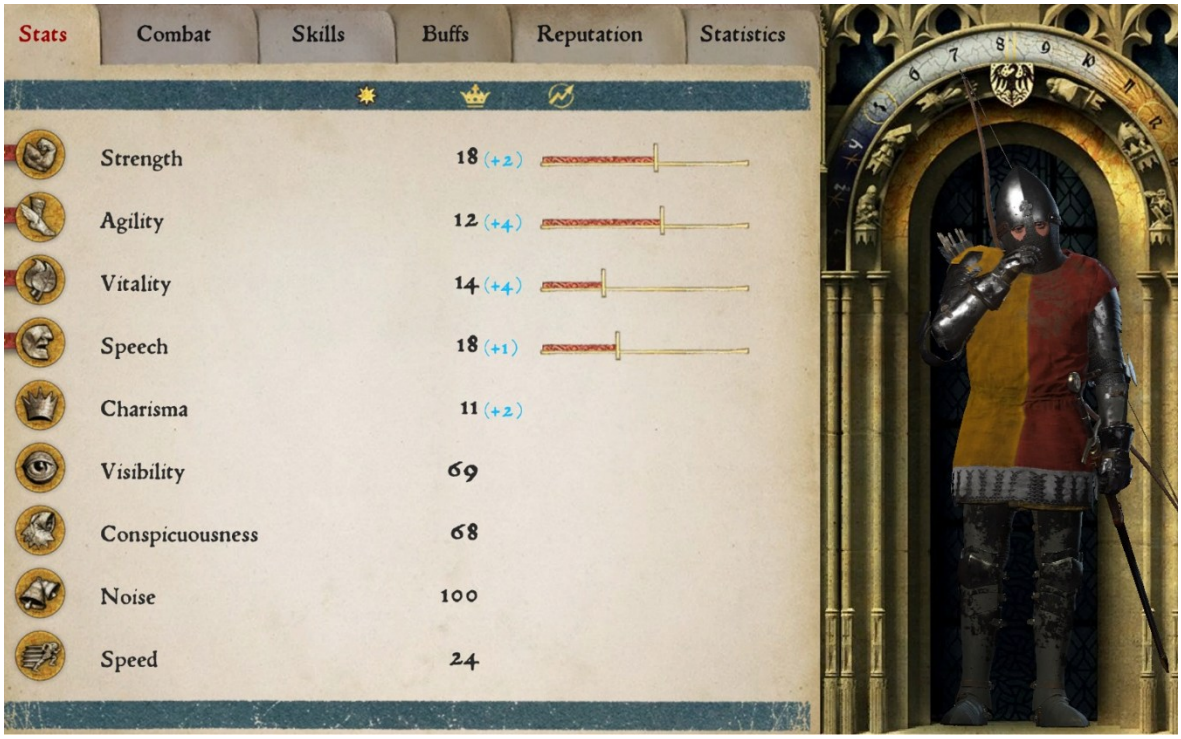
Toinen suosittu Rockstar Gamesin peli Red Dead Redemption 2 (2018), joka sijoittuu 1800- ja 1900-luvun vaihteeseen, on myös loistava taidonnäyte NPC-hahmojen luonnista. Pelinkehittäjät käyttivät 1200 näyttelijää hahmojen luomiseen ja kirjoittivat jopa 80 sivuisia käsikirjoituksia, jotta jokainen NPC tuntuisi moniulotteiselta ja aidolta henkilöltä. Red Dead Redemption 2:ssa on valtavasti NPC-hahmoja, joilla on henkilökohtainen päivittäinen rutiini, persoonallisuus ja mieliala. Hahmoilla on kyky muistaa menneitä tapahtumia ja reagoida pelaajaan niiden mukaisesti. (Velardo 2019.)

Rockstar Gamesin NPC-hahmot ovat merkittävä osa heidän pelejänsä, joissa pelaaja voi parhaimmillaan tuntea olevansa osa pelin elävää maailmaa. Englannin kielen termillä ”emergent gameplay” kuvataan peleissä tilanteita, jotka eivät ole suoraan kehittäjien kirjoittamia tarinoita. Rockstar Gamesin peleissä NPC-hahmojen käytöksestä johtuvat yllättävät tilanteet ovat hyvä esimerkki tämänkaltaisista kokemuksista, ja usein pelaajat saattavat jonkin NPC-interaktion kautta kehittää itselleen pelattavia tilanteita.

2.2 Kingdom Come: Deliverance

Kingdom Come: Deliverance on Warhorse Studiosin vuonna 2018 julkaistu keskiaikaan sijoittuva avoimen maailman roolipeli (IMDb 2025). Pelissä on perinteisten NPC-hahmojen toiminnallisuuksien lisäksi järjestelmä, joka määrittelee miten pelaajan asusteet vaikuttavat NPC-hahmojen käytökseen. Vaatetus vaikuttaa pelaajahahmon näkyvyyteen,

silmiinpistävyyteen, äänekkyyteen ja karismaan. Kuvassa 1 on esimerkki pelaajahahmon parametreista ja päällä olevista asusteista.



Kuva 1. Esimerkki pelaajan parametreista ja asusta Kingdom Come: Deliverance pelissä

Pelaajahahmoon kiinnitetty parametri ”conspicuousness”, eli silmäänpistävyys vaikuttaa kuinka helposti pelaajahahmo erottuu joukosta normaalissa ympäristössä (Kingdom Come: Deliverance Wiki 2018a). Erityisesti räikeän väriset tai muuten erikoiset vaatteet nostavat silmiinpistävyyttä, jolloin NPC-hahmot huomaavat pelaajahahmon epäilyttävät teot erityisen helposti. Kuvan 1 pelaajahahmolla on päällä täysi haarniska ja värikäs vaakuna, jolloin silmpiinpistävyys on kohtuullisen korkea.

Silmiinpistävyys kulkee lähes käsi kädessä ”visibility”, näkyvyys, parametrin kanssa. Näkyvyys määrittelee, kuinka helppo pelaaja on nähdä ylipäänsä (Kingdom Come: Deliverance Wiki 2018b). Näkyvyyteen vaikuttaa päällä olevien vaatteiden väri, tummiin vaatteisiin pukeutunut henkilö on luonnollisesti vaikeampi huomata pimeästä kulmasta. Vaaleat vaatteet eivät välttämättä ole silmiinpistäviä, jolloin silmiinpistävyyden arvo ei nouse, mutta ne ovat kuitenkin selkeästi näkyviä, jolloin näkyvyyden arvo nousee. Kuva 1 hahmolla on vaaleahkot asusteet ja metallinen haarniska, joten näkyvyys on myös melko korkea.

Lisäksi pelissä on parametri ”noise”, äänekkyys, joka kertoo, kuinka paljon ääntä pelaajan vaatetuksesta lähtee (Kingdom Come: Deliverance Wiki 2018c). Äänekkyys vaikuttaa erityisesti hiiviskelyyn. Kuvan 1 hahmolla on täysi haarniska päällä, joten äänekkyyden arvo on täydet sata pistettä.

Viimeisenä ulkonäön parametrina on ”charisma”, eli pelaajan karisma. Karismaattiset vaatteet auttavat tekemään pelaajahahmosta hyvän vaikutelman (Kingdom Come: Deliverance Wiki 2022). Pelaajahahmo voi esimerkiksi uskotella kaupungin vartijalle olevansa korkea-arvoinen henkilö, jolloin voi välttää seuraamuksia sääntöjen rikkomisesta. Pelaajahahmon karismaan vaikuttaa myös vaatteiden likaisuus, joka lisää kannustimen käyttää pelin kylpylöitä. Esimerkiksi kuvan 1 pelaajahahmon haarniska on melko likainen, joka laskee sen karisman arvoa.

Nämä kaikki järjestelmät yhdessä rakentavat peliin toiminnallisuutta, joka kannustaa pelaajaa vaihtamaan asusteita tilanteen mukaan. Mikäli pelaaja esimerkiksi haluaa pelata varasta, kannattaa hahmon päällä olla tummia ja hiljaisia kankaisia vaatteita. Tämä kuitenkin luo suuren riskin erityisesti vihamielisten NPC-hahmojen läheisyydessä, sillä tällaista pelityyliä varten ei voi pitää kovin suojaavia vaatteita päällä. Suojaavassa haarniskassa ei voi hiiviskellä, halvoissa vaatteissa ei ole helppo vakuuttaa NPC-hahmoja ja hiljaiset vaatteet eivät suojaa taistelussa.

2.3 Mafia III

Monet pelit, joissa hyödynnetään NPC-hahmoja, toteuttavat järjestelmänsä kohtuullisen samankaltaisesti keskenään. Onhan tarkoituksena kuitenkin imitoida oikeita ihmisiä. Mafiaelämään perustuva Hangar 13 studion kehittämä peli Mafia III (2016) on lisännyt mielenkiintoisen interaktion, jossa NPC-hahmojen toiminta pelaajahahmon tekemien rikoksien suhteen vaihtelee alueesta alueeseen.

Mikäli alueella on runsasta rikollista toimintaa, paikalliset siviilit eivät soita poliisille todistaessaan laittomuuksia. Poliiseina toimivat NPC-hahmot kuitenkin puuttuvat näkemiinsä rikkeisiin näillä alueilla. Pelaaja voi myös halutessaan ottaa yhteyttä henkilöön, joka estää puheluiden soittamisen tietyltä alueelta joksikin aikaa, jolloin siviilit eivät kykene soittamaan poliisille, vaikka näkisivätkin pelaajan tekevän rikoksia. (Mafia Wiki.)

Tämä järjestelmä tekee pelin maailmasta uskottavan sen teeman sisällä ja vaikuttavat sen pelattavuuteen. Pelaajan täytyy miettiä hahmonsa toimia sijaintinsa kannalta, sillä varomattomuus väärällä alueella saattaa kostautua helposti, mutta syrjäisemmällä alueella voi toimia huolettomammin.

2.4 Hitman 3

Hiiviskelypeleissä, joissa pelaajahahmon tarkoituksena on pysytellä pääosin näkymättömissä NPC-hahmoilta, usein käytetään jonkinlaista mittaria NPC-hahmojen huomaamisen tason näyttämiseksi. Esimerkiksi IO-Interactiven Hitman-pelisarjassa käytetään epäilymittaria, josta esimerkki näkyy keltaisena ympäröitynä kuvassa 2.

Kuva 2. Epäilymittari Hitman 3 pelissä (Gamepressure 2021)

Epäilymittari täyttyy jonkin NPC:n epäillessä pelaajahahmoa, esimerkiksi pelaajan ollessa paikassa, jossa hänen ei kuuluisi olla. Mittari muuttuu keltaiseksi, kun NPC tulee tutkimaan, ja punaiseksi pelaajan jäädessä kiinni. Kuvan tapauksessa oikealla näkyvällä vartijahahmolla on päällään valkoinen pallo, joka kertoo sen olevan tavallista tarkkaavaisempi NPC (Gamepressure 2021). Kyseinen tarkkaavainen vartija tunnistaa pelaajan, koska hänellä on vartijan valeasu päällä, eikä vartija tunnista pelaajahahmoa tuttavakseen. Esimerkiksi siivoojan valeasussa olevaa pelaajahahmoa tämä NPC ei välttämättä tunnistaisi, jos alueella siivoojien on sallittua liikkua.

Nämä pelin järjestelmät luovat syitä käyttää erilaisia valeasuja ja lähestymistapoja tehtäviä suorittaessa. Pelin NPC-hahmot ovat myös uskottavampia, kun jokaisella on oma roolinsa ja mikä tahansa valeasu ei huijaa joka ikistä henkilöä. Epäilymittari antaa myös pelaajalle varoituksen ja mahdollisuuden välttää epäonnistumisia yllättävissä tilanteissa. Mittaria on myös mahdollista käyttää taktisesti houkuttelemalla NPC:n tutkimaan jotain aluetta.

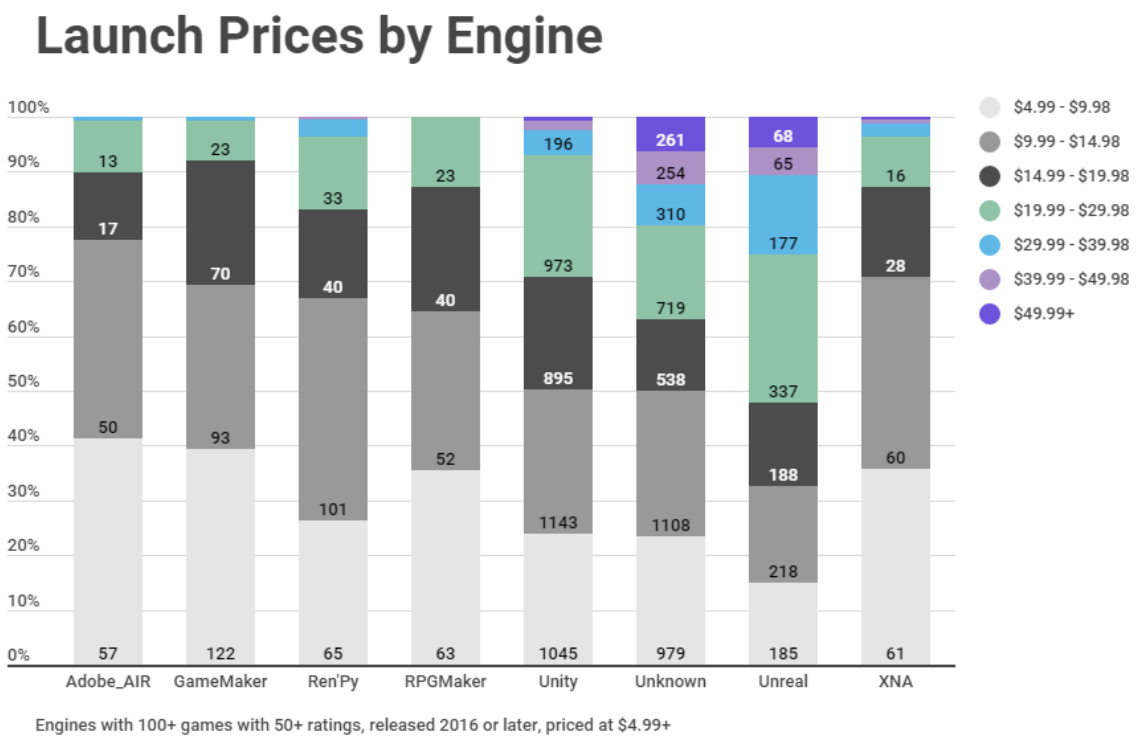
3 Unreal Engine

3.1 Unreal Engine 5

Unreal Engine 5 on uusin versio Epic Gamesin kehittämästä pelimoottorista. Virallisesti Unreal Engine 5 julkaistiin huhtikuussa 2022 monen vuoden kehityksen jälkeen. Alkuperäinen versio Unreal Enginestä oli ensimmäisen kerran käytössä Epic Gamesin kehittämässä vuoden 1998 pelissä Unreal, josta pelimoottorikin sai nimensä. Epic Games on ajan myötä madaltanut pelimoottorilla kehittämisen rahallisia vaatimuksia, mikä on nostanut sen saavutettavuutta merkittävästi. (Doucet & Pecorella 2021.)

Unreal Enginen pääasiallinen ohjelmointikieli on C++. Moottorin lähdekoodi on saatavilla Epic Gamesin verkkosivuilta ilmaiseksi (Epic Games 2025a). Tämä tekee Unreal Enginestä käytännössä avoimen lähdekoodin moottorin, vaikka lakiteknisesti sitä ei sellaiseksi määritelläkään. Saatavilla olevan lähdekoodin ansiosta kuka tahansa voi luoda moottoriin lisäyksiä hyvinkin syvälle moottorin rakenteeseen projektin tarpeita varten. Erityisen tunnettu Unreal Engine on huippuluokan grafiikoista ja loistavista fysiikoista. Esimerkkejä tunnetuista Unreal Enginellä kehitetyistä peleistä on Fortnite, Star Wars Jedi: Fallen Order ja Hellblade: Senua’s Sacrifice. (Johns 2025.)

Vaikka Unreal Engine on merkittävästi parantanut saavutettavuuttaan ajan myötä, se on silti useammin käytössä suuremman luokan projekteissa verrattuna esimerkiksi kilpailijaansa Unityyn. Tätä näkemystä tukee Doucetin & Pecorellan (2021) tutkimus, joka kertoo 25 prosenttia Unreal Enginellä kehitetyistä peleistä lähtöhinnan olevan yli 30 dollaria, kun puolestaan Unitylla kehitettyjen pelien prosentti samalle hintaluokalle on vain 6. Tämän tutkimuksen tulos näkyy kuviossa 1, jossa Unreal Enginen vaaleansinisen ja sen yllä olevien palkkien osuus on huomattavasti suurempi kuin muiden pelimoottorien.



Kuvio 1. Eri pelimoottoreilla kehitettyjen pelien julkaisuhinnat (Doucet & Pecorella 2021)

3.2 Bisnesmalli

Unreal Enginen käyttäminen on täysin ilmaista kaikissa tapauksissa, joissa sitä käyttävä taho ei ole saanut yli miljoonaa dollaria tuloja viimeisen 12 kuukauden aikana. Mikäli tämä raja ylittyy, vaihtoehtoja on kaksi: rojalti- ja käyttäjäpohjainen. (Epic Games 2025b.)

Rojaltipohjaista lisenssiä sovelletaan peli- ja applikaatiokehitykseen, kun kehitetty tuote käyttää pelimoottorin koodia ajon aikana ja tuote lisensoidaan kolmannen osapuolen loppukäyttäjille. Tähän kategoriaan menevät kaikki pelit ja sovellukset, joita myydään suoraan yksityisille asiakkaille tai yrityksille. Tällöin kaikesta yhden miljoonan dollarin yli menevästä tuotosta maksetaan viiden prosentin rojalti Epic Gamesille. Mikäli kehittäjä myy tuotettaan Epic Games Storessa, siel saaduista tuotoista ei kuitenkaan tarvitse maksaa kyseistä rojaltia. (Epic Games 2025b.)

Käyttäjäpohjaista (seat-based) lisenssiä puolestaan käytetään muissa tapauksissa, kuten esimerkiksi televisioon tarkoitetun sisällön luomisessa. Tällöin Unreal Engineä käyttävä taho maksaa jokaisesta käyttäjästä 2162,36 € vuodessa. Mikäli käyttäjäpaikkoja on ostettu yli kymmenen, on myös mahdollista ostaa Epic Direct Support avuksi kehitykseen. (Epic Games 2025b.)

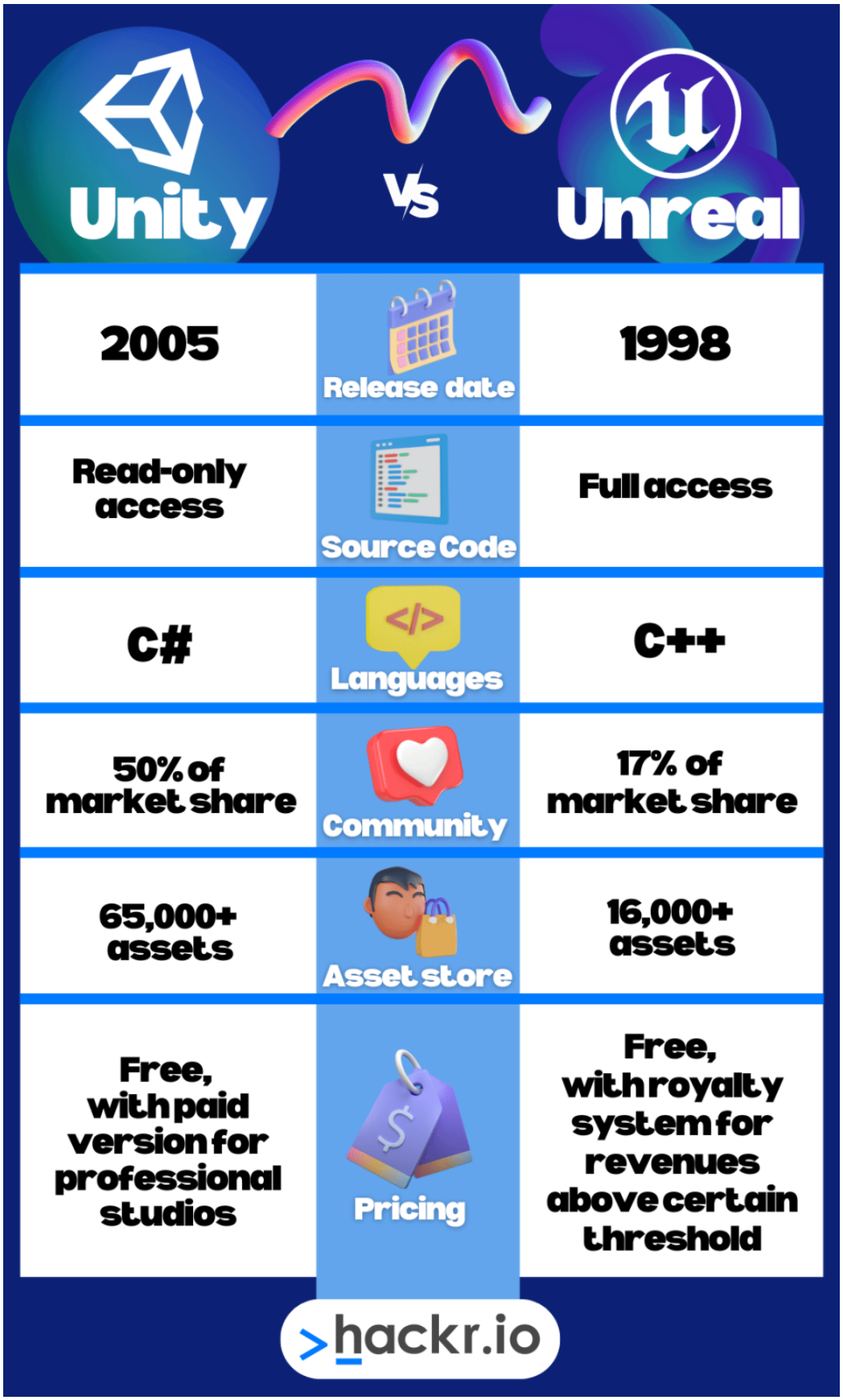
3.3 Vertailukohteena Unity

Unreal Enginen pääasiallinen kilpailija on Unity Technologiesin Unity pelimoottori, ja niitä vertaillaan jatkuvasti toisiinsa samankaltaisuuksiensa vuoksi. Unity julkistettiin ensimmäistä kertaa Applen World Developer Conference:ssa vuonna 2005, ja uusin versio moottorista on julkaistu 17. lokakuuta 2024 nimellä Unity 6. Unity käyttää pääasiallisena ohjelmointikielenä C#:ia. Toisin kuin Unreal Enginen tapauksessa, Unityn lähdekoodia ei lähtökohtaisesti ole mahdollista ladata muokattavaksi. Unity sekä Unreal Engine tukevat kumpikin laajalti alustoja tietokoneista konsoleihin ja puhelimiin, sekä virtuaalitodellisuuteen. (Johns 2025.) Kuvassa 3 seuraavalla sivulla on pikainen vertailu pelimoottorien välillä.

Unityn ajatellaan olevan merkittävästi aloittelijaystävällisempi Unreal Engineen verrattuna, sillä Unityn käyttämä C# on helpompi ohjelmointikieli oppia kuin Unreal Enginen C++, joka tarjoaa enemmän hallintaa, mutta on vaikeampi. Lisäksi Unityn 2D pelien kehittämiseen erikoistuneet työkalut ovat merkittävästi parempia kuin Unreal Enginen, vaikka Unreal Enginessäkin on mahdollista luoda 2D projekteja. Kummankin moottorin yhteydessä on saatavilla kauppapaikka, josta voi ostaa ja ladata assetteja projekteihin käytettäväksi. Unityn kauppapaikassa Unity Asset Store:ssa on laaja valikoima assetteja, työkaluja ja plugineja. Unreal Enginen Marketplace on puolestaan pienempi, mutta keskittyy korkealaatuisiin assetteihin ja materiaaleihin. (Johns 2025.)

Suurin etu Unreal Enginellä on kuitenkin sen realistisissa grafiikoissa ja fysiikoissa, jonka vuoksi moottori onkin hyvin suosittu AAA kehittäjien keskuudessa. Myös Unreal Enginen renderöinti on nopeampaa kuin Unityn. Lisäksi Unreal Enginen moninpelin työkalut ovat kehittyneempiä kuin Unityn, jossa tarvitsee tehdä enemmän manuaalista työtä toiminnallisuuden implementoimiseksi. (Johns 2025.)

Unitylla lisenssivaihtoehtoja on lukuisia, ilmaisesta henkilökohtaisesta lisenssistä käyttäjäpohjaisiin lisensseihin. Ilmaista Unity Personal lisenssiä saa käyttää, mikäli pelimoottoria käyttävä taho ei ole saanut yli 200 000 dollaria tuloja viimeisen 12 kuukauden aikana. Mikäli tämä raja ylittyy, täytyy käyttäjän siirtyä käyttäjäpohjaiseen Unity Pro lisenssiin. Yli 25 miljoonaa dollaria 12 kuukaudessa tienanneet yritykset käyttävät puolestaan Unity Enterprise lisenssiä, jonka ominaisuudet neuvotellaan Unityn kanssa yrityskohtaisesti. Unity Enterprise on myös ainut lisenssi, joka antaa pääsyn Unityn lähdekoodiin. (Unity Technologies 2024.)



Kuva 3. Yksinkertainen vertailu Unityn ja Unreal Enginen välillä (Johns 2025)

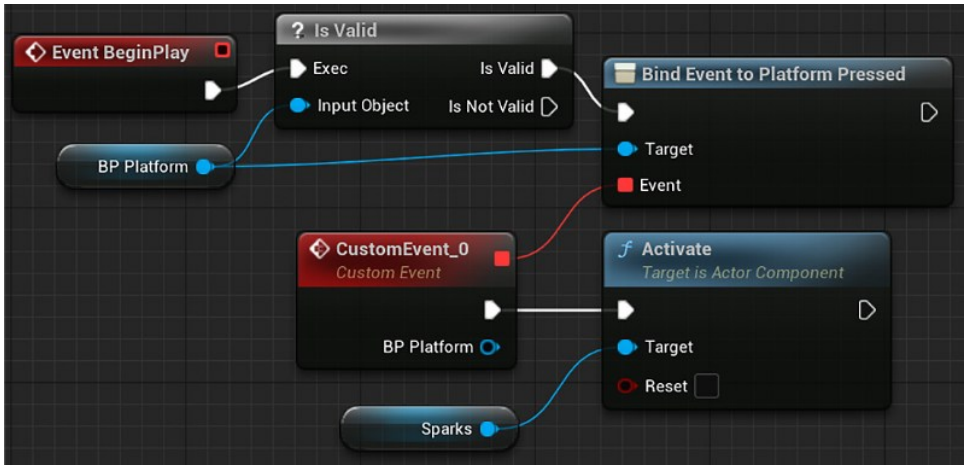
3.4 Blueprints Visual Scripting

Blueprints Visual Scripting on Unreal Enginen visuaalinen ohjelmointityökalu, jolla ohjelmoijat voivat luoda toiminnallisia perustoja suunnittelijoiden jatkettavaksi (Romero & Sewell 2022, XIII). Unreal Enginessä sana Blueprint voi tarkoittaa sekä työkalussa käytettävää visuaalista ohjelmointikieltä, että kyseisellä kielellä luotua pelin osaa (Cataldi ym. 2022, 8). Pelin osiksi luodut Blueprintit voivat käytännössä olla mitä tahansa näkymättömistä pelillisyyden elementeistä rakennuksiin ja hahmoihin.

Siinä, missä valtaosa ohjelmointikielistä ovat tekstipohjaisia, Unreal Enginen Blueprint perustuu solmupohjaiseen käyttöliittymään (Cataldi ym. 2022, 21). Kokonaisuudessaan Unreal Engine on rakennettu C++ ohjelmointikielellä, jota myös Blueprint järjestelmä käyttää taustalla. Blueprintin osioita on myös mahdollista luoda kirjoittamalla itse C++ kielellä, joka tekee järjestelmästä hyvinkin mukautuvan.

Blueprint-tyyppejä on kahdenlaisia: Level Blueprint ja Blueprint Class. Jokaisella pelin tasolla on oma Level Blueprint ja niitä ei voi luoda erikseen ilman uutta tasoa. Blueprint Class:ia käytetään, kun luodaan interaktiivisia osia peliin. Toisin kuin Level Blueprint:ia, Blueprint Class:ia voi käyttää uudelleen missä tahansa tasossa. (Cataldi ym. 2022, 8.)

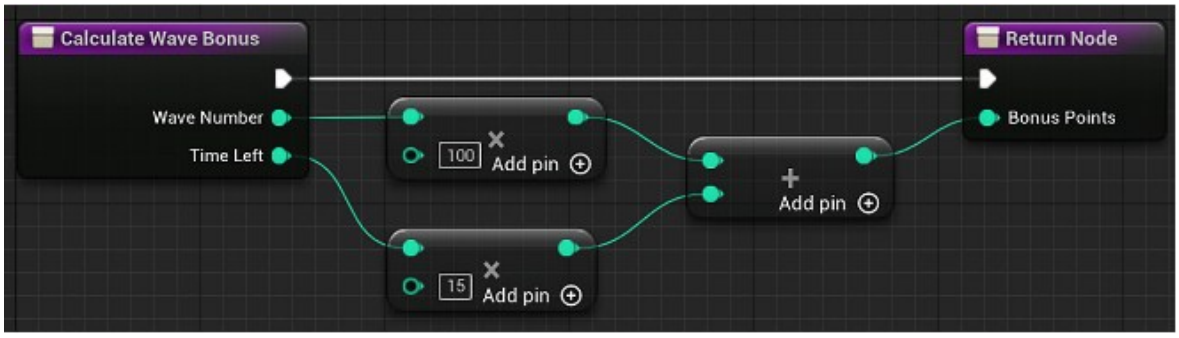
Blueprinteissä suorittaminen alkaa aina tapahtumista (Event), joita löytyy pelimoottorista itsestään lukuisia (Epic Games 2025c). Näitä tapahtumia ovat esimerkiksi kahden objektin törmäys tai jokin syöttötapahtuma (Cataldi ym. 2022, 27). Tapahtumia voi myös luoda itse (Custom Event), jolloin suorittamisen ehdot voi määritellä itse. Jotta mukautettua tapahtumaa voi käyttää, täytyy sen aktivointi ohjelmoida ensin. Tämän voi toteuttaa joko Event Dispatcher -solmulla tai sitomalla. Esimerkki sitomalla toteutetusta mukautetusta tapahtumasta näkyy kuvassa 4.



Kuva 4. Esimerkki mukautetun tapahtuman käyttämisestä sitomalla (Cataldi ym. 2022, 92)

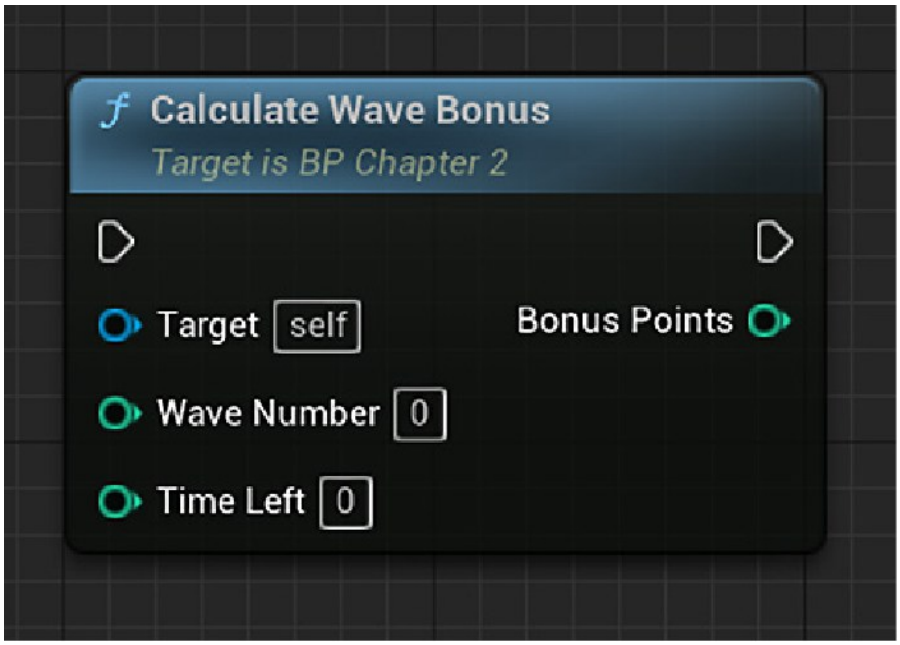
Kuvan 4 esimerkissä sidotaan mukautettu tapahtuma CustomEvent_0 alustan painautumiseen Bind Event to Platform Pressed solmulla. Kun alusta painautuu ja mukautettu tapahtuma suoritetaan, Sparks niminen objekti aktivoituu.

Mikäli joitakin osioita Blueprintistä tarvitsee käyttää useammin kuin kerran, on niistä mahdollista luoda makroja tai funktioita (Cataldi ym. 2022, 32). Tämä on erityisen hyödyllistä, mikäli makrojen tai funktioiden toimintoihin tarvitsee tehdä muutoksia, sillä uudelleen käytettyjä osioita ei tarvitse etsiä erikseen joka paikasta, jossa niitä on käytetty (Cataldi ym. 2022, 32). Makrot ovat myös hyödyllisiä Blueprinttien rakenteen selkeyttämiseksi. Funktiot ovat pintapuolisesti hyvin samankaltaisia kuin makrot, mutta toisin kuin makroja, niitä on mahdollista käyttää muissakin Blueprinteissä (Cataldi ym. 2022, 36). Kuvassa 5 on esimerkki funktiosta.



Kuva 5. Esimerkki funktiosta, jossa on kaksi sisääntuloa ja yksi ulostulo (Cataldi ym. 2022, 38)

Kuvassa 5 näkyy vasemmalla funkion alku, jossa on funktion sisääntulon parametrit Wave Number ja Time Left. Oikealla kuvassa puolestaan on funktion palautussolmu, jossa näkyy funktion ulostuloparametri Bonus Points. Kuvan funktio kertoo Wave Number luvun sadalla, Time Left luvun viidellätoista, lisää molempien tuloksen yhteen ja palauttaa sen Bonus Points ulostuloon. Kuvassa 6 näkyy, miltä funktio näyttää valmiiksi käytettävänä Blueprintissä.

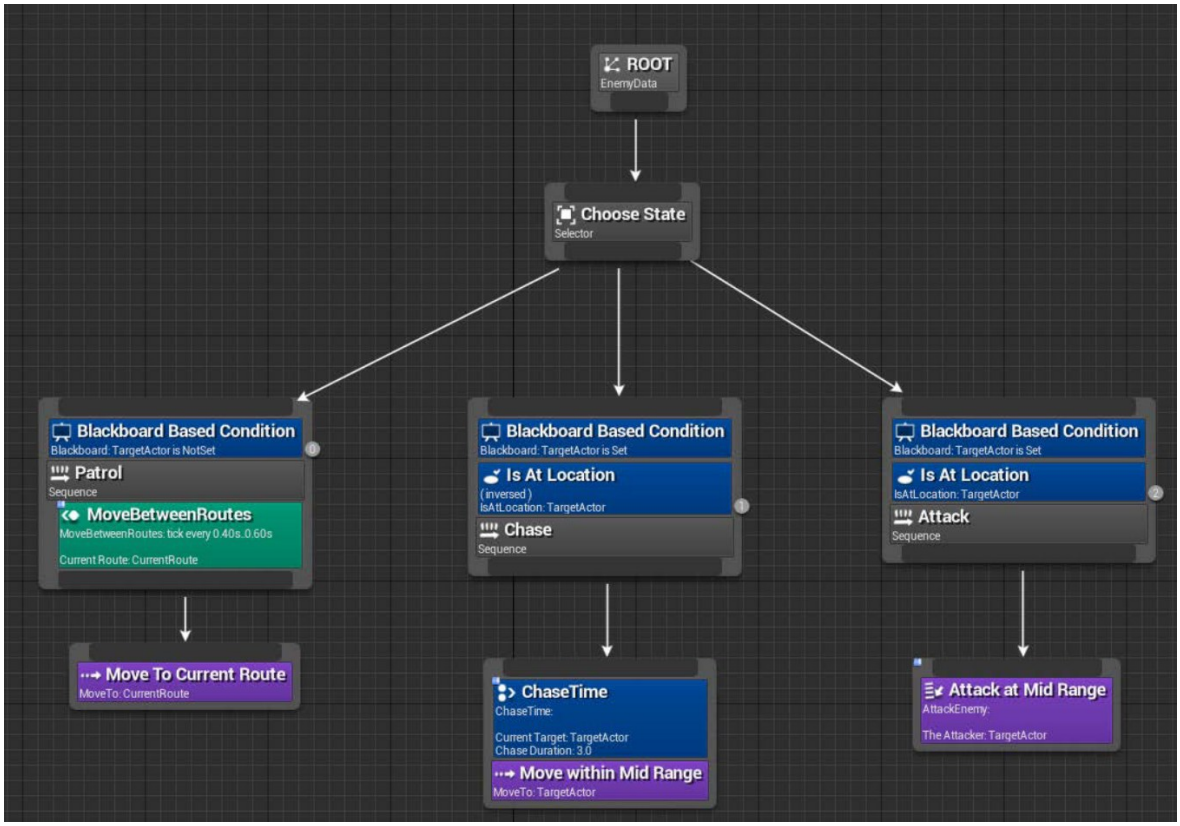


Kuva 6. Funktio sijoitettuna Blueprinttiin (Cataldi ym. 2022, 38)

3.5 Behavior Tree

Behavior Tree on assetti Unreal Enginessä, jota käytetään tekoälyn luomiseen NPC-hahmoille (Epic Games 2025d). Oleellinen Behavior Treen toiminnalle on Blackboard assetti, joka toimii ikään kuin Behavior Treen aivoina (Epic Games 2025d). Feng & Newton (2016, 5) kuvailevat Behavior Tree rakenteen olevan kuin tilakone, mutta joustavampi. He käyttävät termiä hierarkkinen tilakone, joka lisää toisen tason päätöksiä tilakoneen tilan määrittelemiseksi.

Behavior Tree koostuu juuresta, komposiiteista, decorator-lauseista, palveluista ja tehtävistä. Juuri on aina Behavior Treen ensimmäinen solmu, josta lähtee signaali seuraavaan solmuun. Juurta täytyy seurata aina komposiittisolmu, joita on kolmen tyyppisiä: valitsin, sekvenssi ja yksinkertainen rinnakkaisuus. Valitsin suorittaa kaikki sitä alemmat solmut vasemmalta oikealle, kunnes joku niistä onnistuu. Sekvenssi puolestaan suorittaa alempia solmuja vasemmalta oikealle, kunnes joku niistä epäonnistuu. Yksinkertainen rinnakkaisuus mahdollistaa sekä tehtävien, että muun Behavior Treen yhtäaikaisen suorittamisen. Decorator-lauseet ovat ehdollisia lauseita, joilla voi hallita Behavior Treen solmujen suorittamista. Palvelut kiinnitetään komposiitteihin, ja ne ajavat itseään annetuin väliajoin niin kauan kuin komposiitti pysyy aktiivisena. Tehtävät ovat usein viimeisinä omissa haaroissaan, ja suorittavat varsinaisen työn Behavior Treessä. (Feng & Newton 2016, 6–9.) Nämä kaikki solmut näkyvät kuvan 7 esimerkissä. Juuri ja komposiitit ovat harmaita, decorator-lauseet sinisiä, palvelu vihreä ja tehtävät purppuroita.



Kuva 7. Esimerkki Behavior Treen rakenteesta (Feng & Newton 2016, 6)

Blackboard, jota aiemmin luonnehdittiin kuin Behavior Treen aivoiksi, on assetti, jossa varastoidaan muuttujia avaimiin Behavior Treen käytettäväksi. Blackboardissa voi varastoida esimerkiksi pisteitä reititykselle, viittauksia kohteiden tunnistamiselle tai tiloja NPC-hahmon tilaa varten.

3.6 AI Perception

AI Perception on Unreal Enginen järjestelmä, joka mahdollistaa lähteiden luoda erilaisia ärsykkeitä, joihin kuuntelijat voivat reagoida (Feng & Newton 2016, 95). Tällä järjestelmällä on mahdollista luoda realistisesti ärsykkeisiin reagoivia NPC-hahmoja.

AI Perception on komponentti, joka lisätään NPC-hahmon AIController Blueprinttiin. Komponentilla määritellään mitä aisteja hahmon tulee kuunnella, mitä parametreja näillä aisteilla on ja miten hahmon kuuluu reagoida, kun aistiminen tapahtuu. Hahmoille voi määritellä kuulon, näön, kosketuksen, ennustamisen ja vahingon ottamisen aistit. Lisäksi hahmot voi määritellä tunnistamaan, kun toinen samaan joukkueeseen kuuluva hahmo on lähellä. (Epic Games 2025e.)

4 Case

4.1 Toimeksianto

Opinnäytetyön toimeksiantona on luoda Lohkare Gamesin Last Drop peliin logiikka hahmolle, joka pelaajan nähdessään ryhtyy jahtaamaan tätä ja pyrkii perimään tämän velat. Tarkoituksena ei ole tehdä täysin valmista hahmoa, vain yleisen käyttäytymisen logiikka, joten varsinainen velan keräämisen toiminnallisuus jää opinnäytetyön toimeksiantajan toteutettavaksi.

Toteutus tapahtui Unreal Engine 5 editorissa, joka on saatavilla Unreal Enginen verkkosivuilta. Versionhallintaan käytettiin GitHubia ja kommunikointiin Discordia.

4.2 Suunnitteleminen

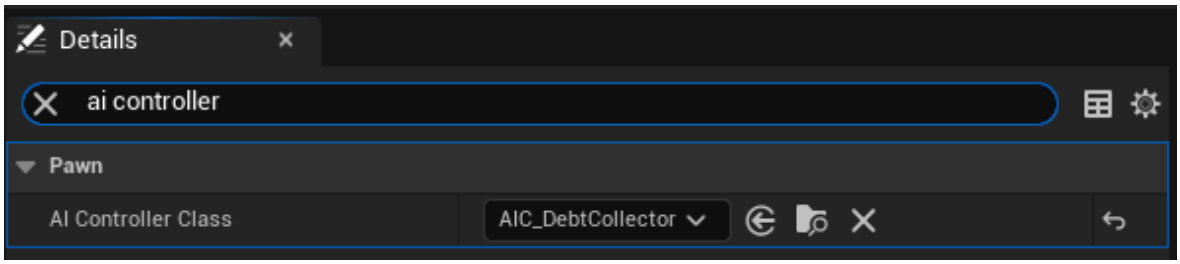
Teoriaosuudessa käsiteltyjen pelien esimerkit ovat suurien studioiden ja tuhansien työtuntien tuotoksia, joten samankaltaisten tuotosten luominen opinnäytetyön puitteissa oli mahdotonta. Täytyi siis miettiä, mikä on mahdollista aika- ja taitorajoitteiden mukaan.

NPC-hahmon haluttiin mukautuvan jollain tavalla pelaajahahmon toimiin, jotta hahmo ei tuntuisi aivan yksiulotteiselta. Päädyttiin ratkaisuun, että NPC-hahmolla olisi jonkinlainen kiukkuisuuden taso, jonka mukaan hahmo liikkuu nopeammin. Lisäksi hahmolle piti luoda perustoiminnot vaeltamiselle ja jahtaamiselle pelaajahahmon ollessa näkökentässä. Kiukkuisuuden tason voisi esimerkiksi määritellä pelaajahahmon velan määrä.

4.3 Assettien luominen

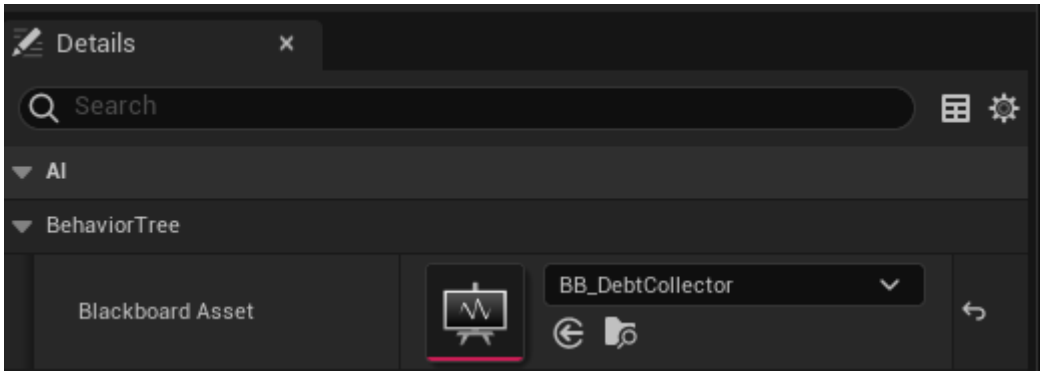
Pohjana esimerkkihahmolle käytettiin projektiin jo luotua NPC Blueprinttiä, jolloin hahmon 3D-malli, animaatiot ja fysiikat olivat jo valmiina, jolloin voitiin keskittyä vain toiminnallisuuteen. Tämä onnistui yksinkertaisesti kopioimalla projektissa olemassa oleva assetti BP_NPC ja nimeämällä se uudelleen nimellä BP_DebtCollector. Blueprintin sisältä myös poistettiin olemassa olevat solmut. Tämä Blueprint on se assetti, joka asetetaan tasoon NPC-hahmoksi.

Jotta hahmoa voidaan hallita NPC-hahmona, sille tarvitaan AI controller Blueprint, joten projektiin luotiin assetti AIC_DebtCollector. Hahmon omassa kopioidussa Blueprintissä oli vielä linkki BP_NPC assetin AI controlleriin, joten se täytyi kuvan 8 mukaisesti korvata uudella assetilla.



Kuva 8. AI Controller Blueprint määriteltynä BP_DebtCollector assetin tiedoissa

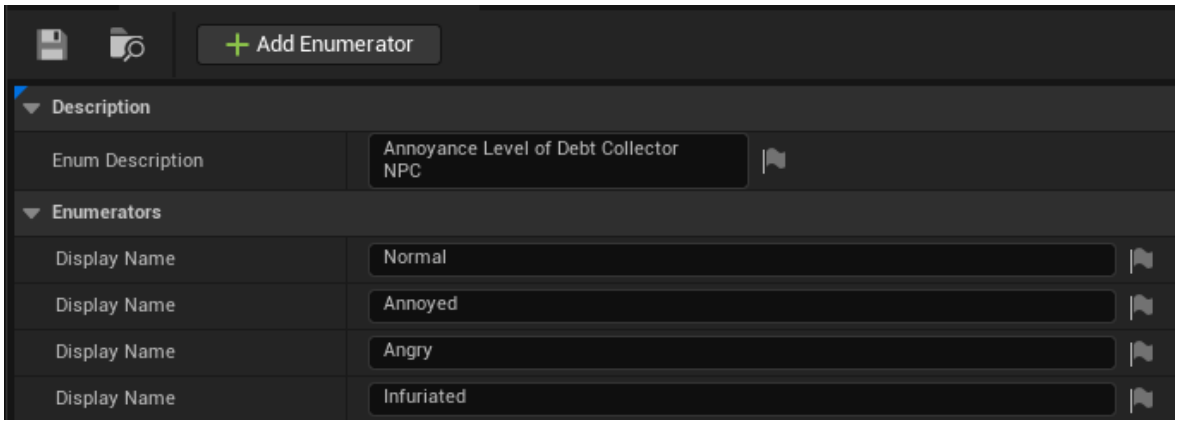
Behavior Treen integroimiseksi luotiin sekä Blackboard, että Behavior Tree assetit. Blackboard nimeltään BB_DebtCollector ja Behavior Tree BT_DebtCollector. Behavior Treen tiedoissa asetettiin Blackboard assetiksi BB_DebtCollector kuvan 9 mukaisesti, jotta Behavior Tree osaa käyttää oikean Blackboardin tietoja.



Kuva 9. BB_DebtCollector määriteltynä Blackboardiksi Behavior Treen tiedoissa

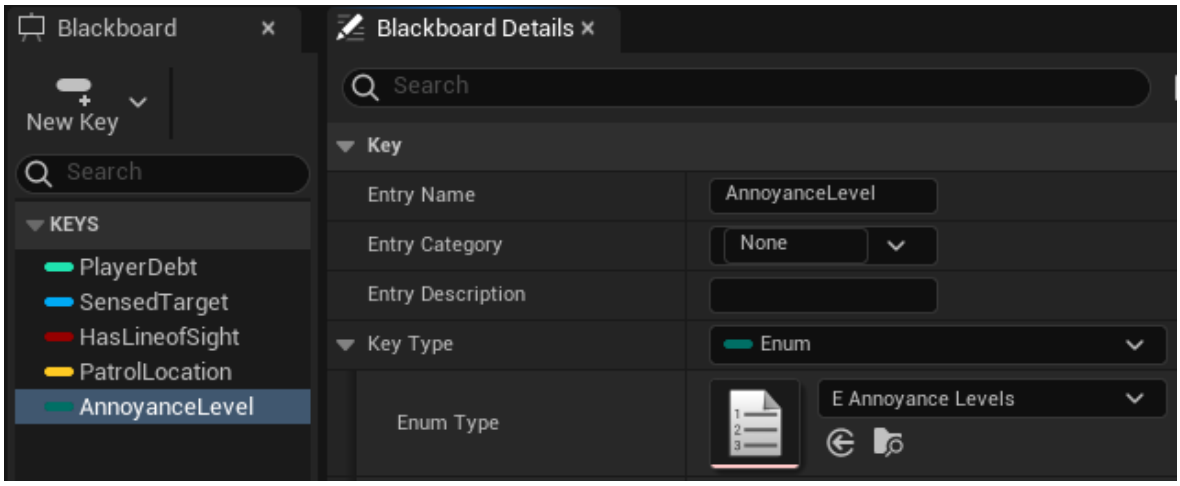
NPC-hahmon liikkumisen toiminnallisuuden toteuttamista varten luotiin Blackboardiin avaimiksi objektityypin ja Actor-luokan SensedTarget, vektorityypin PatrolLocation ja boolean-tyypin HasLineOfSight. Blackboard avain SensedTarget varastoi NPC-hahmon havaitseman kohteen tiedot sen varmistamista varten, PatrolLocation varastoi sijainnin, johon NPC liikkuu vaeltaessaan ja HasLineOfSight varastoi tiedon, onko havaittu kohde pelaajahahmo vai jokin muu.

Ärsyyntymisen tason on tarkoitus vaikuttaa NPC-hahmon nopeuteen jahdatessa ja vaeltaessa. Tätä varten luotiin Enumeraatio tyypin assetti E_AnnoyanceLevels kuvan 10 mukaisesti. Assettiin listattiin neljä enumeraatiota: Normal, Annoyed, Angry ja Infuriated.



Kuva 10. Enumeraatio assetti E_AnnoyanceLevels

E_AnnoyanceLevels lisättiin myös Blackboardiin avaimeksi nimellä AnnoyanceLevel. Lisäksi luotiin Integer tyypin avain PlayerDebt, mikäli sitä olisi tarvittu myöhemmin. Valmis Blackboard ja avain AnnoyanceLevel näkyy kuvassa 11. Avainten värit kuvastavat niiden tyyppejä.



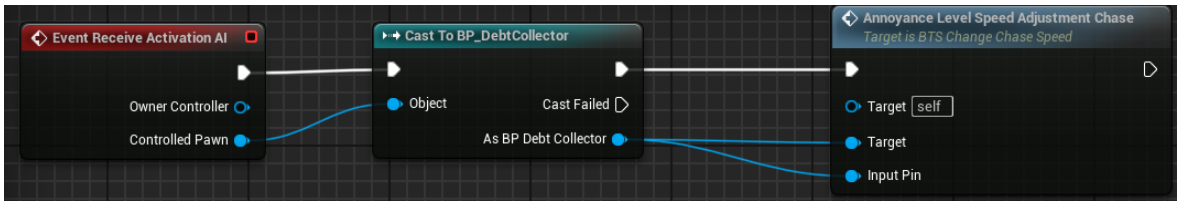
Kuva 11. Valmis Blackboard ja AnnoyanceLevel avaimen tiedot avattuna

4.4 Vaeltamisen ja jahtaamisen toteutus

Behavior Treen toimintaa varten luotiin kaksi Behavior Tree Service Blueprint assettia, BTS_ChangeChaseSpeed jahtaamista ja BTS_FindPatrolPoint vaeltamista varten. Ensimmäistä käytetään seuraamisen ja jälkimmäistä vaeltamisen toiminnallisuuteen.

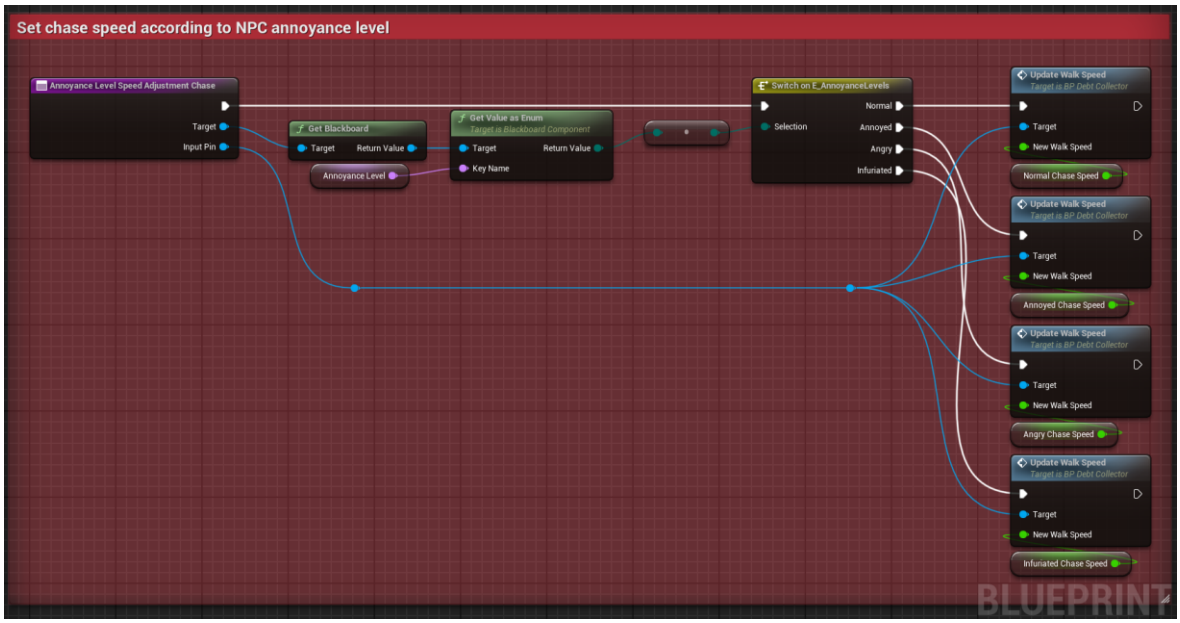
4.4.1 BTS_ChangeChaseSpeed

BTS_ChangeChaseSpeed service assetin sisältö luotiin kuvan 12 mukaisesti. Kun service suoritetaan, se muuttaa NPC-hahmon jahtaamisen nopeutta Blackboardista haetun AnnoyanceLevel enumeraattorin mukaisesti.



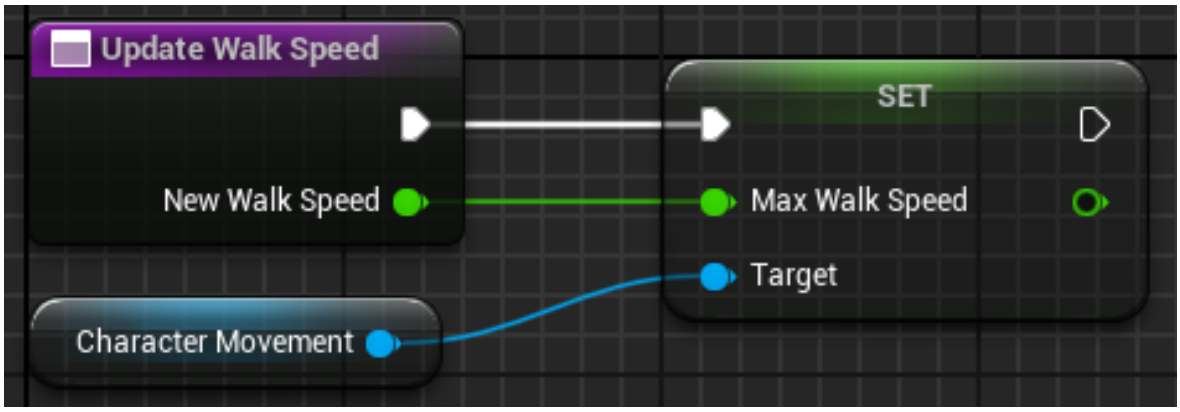
Kuva 12. BTS_ChangeChaseSpeed service hahmon liikenopeuden asettamiseksi

Kun service saa aktivointikomennon, se yrittää päästä NPC-hahmon Blueprinttiin (Cast to BP_DebtCollector) ja toteuttaa perässä olevan funktion Annoyance Level Speed Adjustment Chase, lähettäen funktioon NPC Blueprintin tiedot. Mikäli Behavior Treetä käytetään oikein, tämä onnistuu aina sillä ohjattu Blueprint on BP_DebtCollector. Viimeisin solmu on kuvan 13 mukainen itse toteutettu funktio, joka säätää NPC hahmon nopeuden jahdatessa ärsyyntyneisyyden tason mukaan.



Kuva 13. Funktio liikenopeuden muuttamiseen ärsyyntyneisyyden tason mukaan

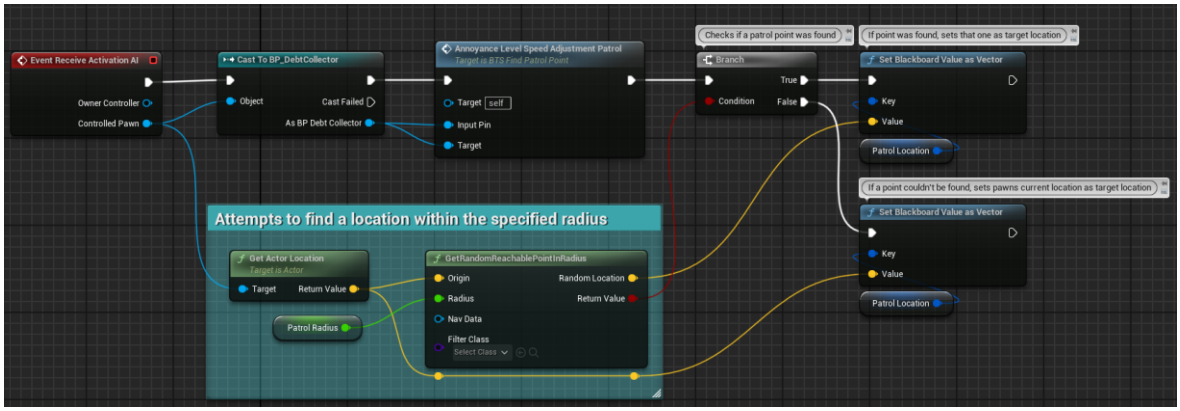
Funktio hakee ohjatun Blueprintin Blackboardista tiedon nykyisestä AnnoyanceLevel arvosta, ja suorittaa switchin avulla arvoon määritellyn Update Walk Speed funktion. Kuvan kirkkaan vihreät solmut ovat julkisia muuttujia, joita voidaan säätää Behavior Treessä, kun service on liitettynä solmuun. Update Walk Speed on hyvinkin yksinkertainen funktio NPC-hahmon kävelynopeuden säätämiseksi, joka ottaa funktiolle annetun numeroarvon ja asettaa sen Character Movement komponentin sisällä olevaan Max Walk Speed muuttujaan kuvan 14 mukaisesti. Cast to BP_DebtCollector mahdollistaa kyseisen funktion suorittamisen AI controllerissa, vaikka funktio on eri Blueprintissä.



Kuva 14. Funktio kävelynopeuden muuttamiseen BP_DebtCollector Blueprintissä

4.4.2 BTS_FindPatrolPoint

BTS_FindPatrolPoint service assetin sisältö puolestaan luotiin kuvan 15 mukaisesti. Servicen tarkoituksena on muuttaa kävelynopeus hitaammaksi kuin seuratessa, sekä etsiä ja määritellä pisteitä maailmassa vaeltamista varten.

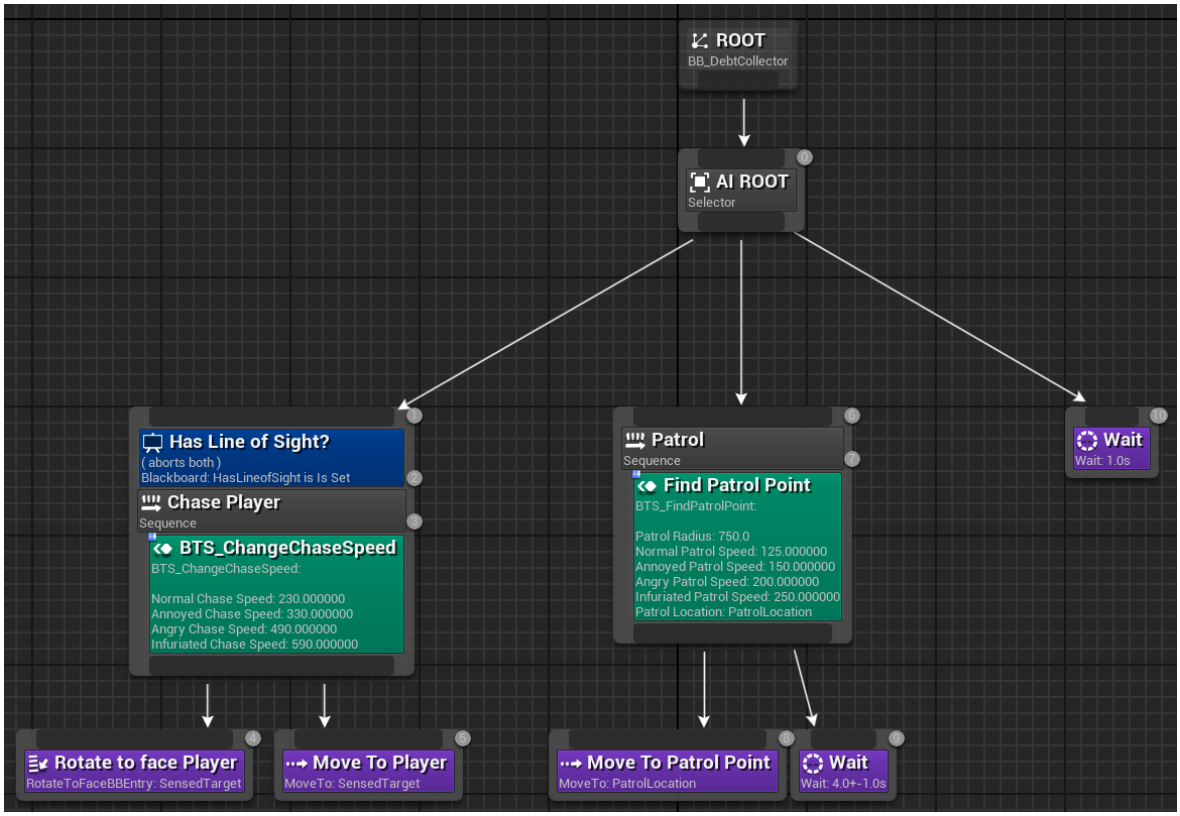


Kuva 15. BTS_FindPatrolPoint service

BTS_FindPatrolPoint on hyvin samanlainen BTS_ChangeChaseSpeed servicen kanssa. Kävelynopeuden määrittelyn lisäksi service pyrkii löytämään pelin maailmasta pisteen määritellyn säteen sisältä, ja löytäessään sellaisen asettaa sen Blackboardiin Behavior Treen luettavaksi. Funktio Annoyance Level Speed Adjustment Patrol on identtinen samannimisen Chase function kanssa (kuva 12), erona vain sisään menevät arvot, jotka säädetään myös Behavior Treessä. Mikäli service ei löydä maailmasta validia pistettä, se asettaa Blackboardiin NPC:n nykyisen sijainnin, jolloin hahmo pysyy paikallaan.

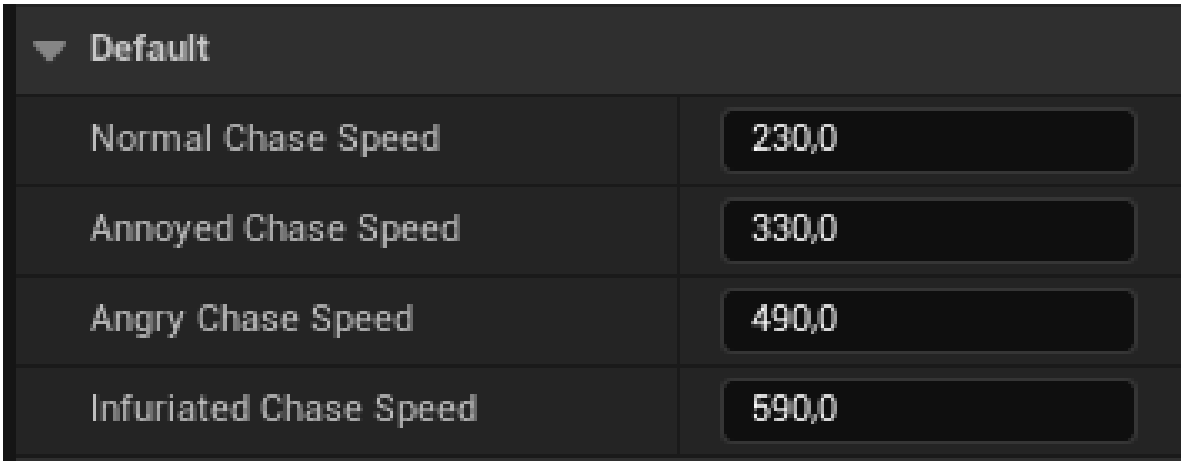
4.4.3 Behavior Tree

Behavior Treessä toteutettiin NPC-hahmon käyttäytymislogiikka yhdistämällä Blackboardin tiedot ja Behavior Tree servicet. Tätä varten luotiin juuresta valitsinsolmu AI ROOT, jossa on kiinni kaksi sekvenssisolmua ja yksi solmu odottamiselle kuvan 16 osoittamalla tavalla. Valitsinsolmu suorittaa solmuja vasemmalta oikealle, kunnes joku niistä onnistuu. Sekvenssit suorittavat alempia solmuja järjestyksessä ja lopettaa vasta, kun joku niistä epäonnistuu.



Kuva 16. Valmis Behavior Tree

Chase Player solmuun kiinnitettiin aiemmin luotu vihreä BTS_ChangeChaseSpeed service ja sininen “Has Line of Sight?” decorator. Decorator on asetettu keskeyttämään itsensä ja alemman prioriteetin solmut, mikäli Blackboardin avain HasLineofSight muuttuu. Behavior Treessä solmujen tärkeysjärjestys on ylhäältä alas, vasemmalta oikealle, joten kun Chase Player-solmu on vasemmanpuoleisin solmu, se on tärkeysjärjestyksessä korkeimmalla. HasLineofSight avaimen ollessa tosi, alkaa Chase Player toteuttamaan alempia toimintoja, jotka kääntävät ja liikuttavat NPC:tä SensedTarget avaimessa olevaa objektia kohti. HasLineofSight avaimen muuttuessa epätodeksi, suoritus ei onnistu, jolloin AI ROOT valitsin alkaa suorittamaan Patrol sekvenssiä. BTS_ChangeChaseSpeed servicessä näkyy myös arvot nopeuksille, joita voi servicen tiedoista säätää vapaasti kuvan 17 osoittamassa ikkunassa.



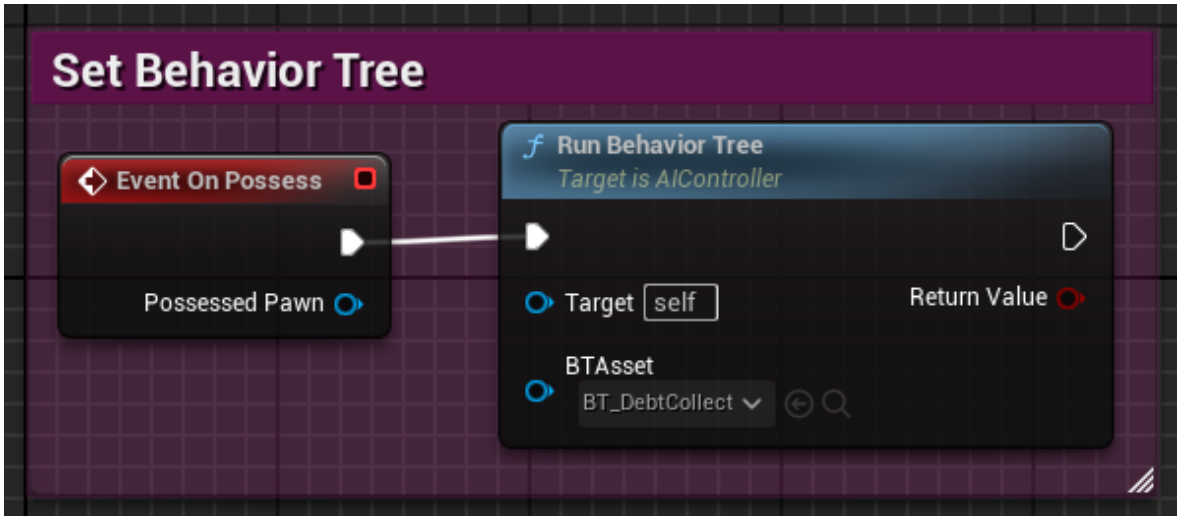
Kuva 17. BTS_ChangeChaseSpeed servicen tiedot hahmon liikkumisen nopeuksille

Patrol sekvenssiin kiinnitettiin BTS_FindPatrolPoint service, jossa voi säätää nopeuksien lisäksi vaeltamisen sädettä ja asettaa Blackboard avain servicelle käytettäväksi. Sekvenssistä lähtee kaksi toimintoa, Move To Patrol Point ja Wait. Ensimmäinen liikuttaa NPC-hahmon PatrolLocation avaimen osoittamaan sijaintiin, ja jälkimmäinen odottaa 3–5 sekuntia ennen kuin sekvenssi suoritetaan uudestaan.

Valmis Behavior Tree siis suorittaa Patrol sekvenssiä, kunnes HasLineofSight Blackboard avain muuttuu todeksi, jolloin Chase Player sekvenssi alkaa. Chase Player sekvenssi puolestaan loppuu avaimen muuttuessa epätodeksi, jolloin sekvenssi epäonnistuu ja AI ROOT valitsin suorittaa seuraavan onnistuvan solmun, joka on Patrol.

4.4.4 AI controller

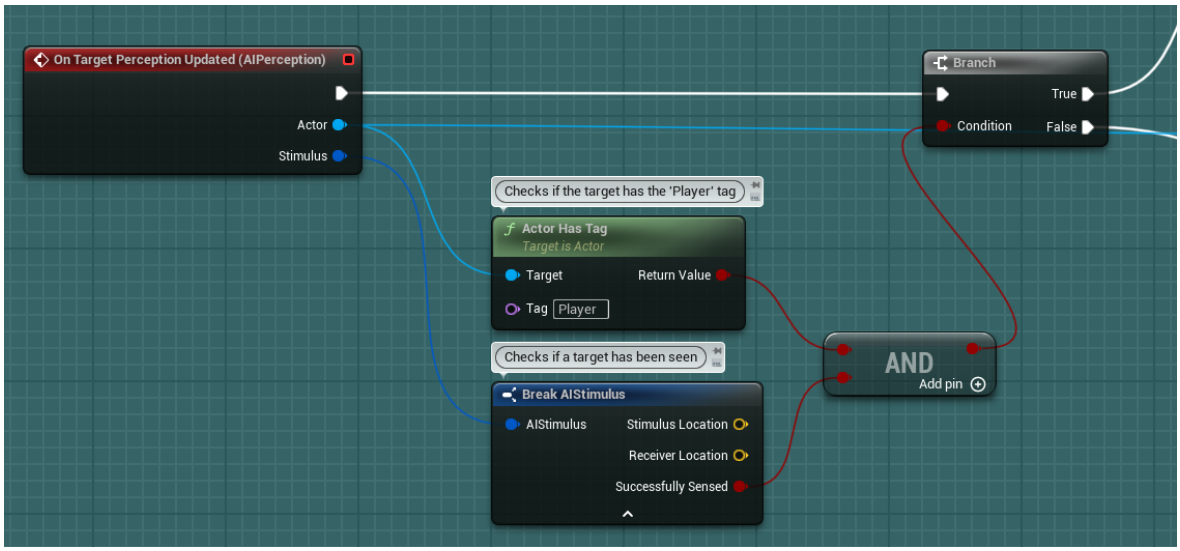
Behavior Treen logiikan toteuttamiseksi NPC-hahmossa täytyi Behavior Tree asettaa halutun NPC:n AI controlleriin kuvan 18 osoittamalla tavalla. AI controllerin ottaessa hahmon hallintaansa, Run Behavior Tree komento toteutuu, jossa on määritelty haluttu Behavior Tree BTAsset valikossa.



Kuva 18. Behavior Treen ajaminen NPC-hahmoon AI controllerissa

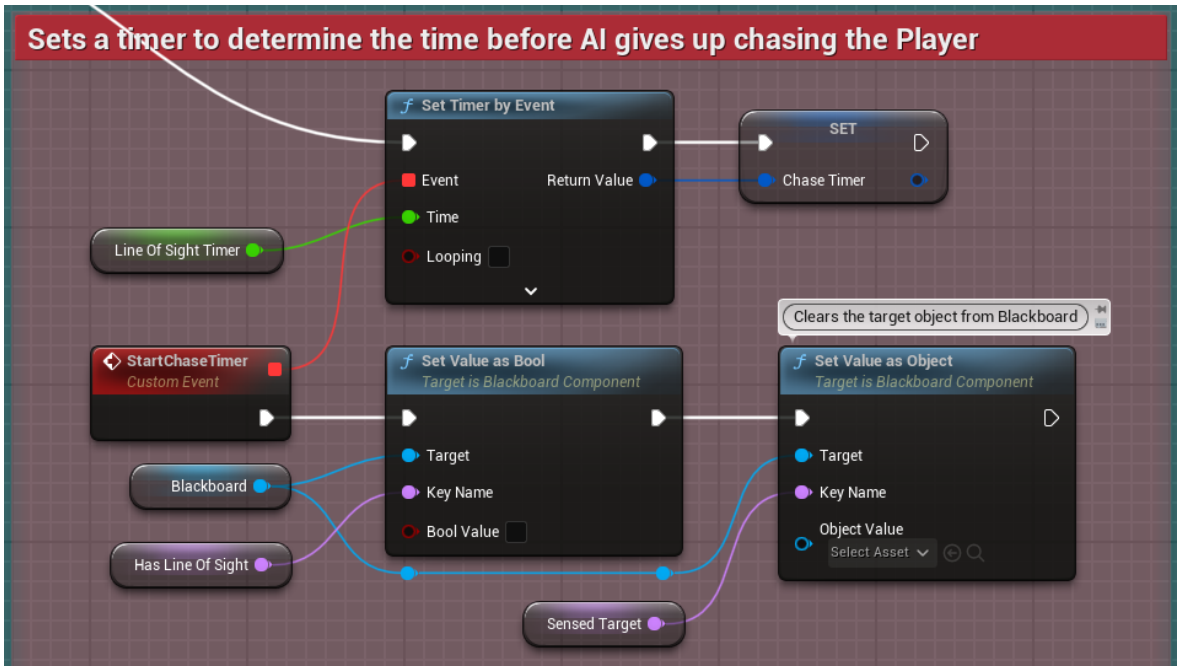
Jotta Behavior Tree voi käyttää HasLineofSight Blackboard avainta hyödykseen, täytyy hahmolle toteuttaa näköaisti. Tätä varten täytyy AI controlleriin lisätä komponentti AIPerception. AIPerception komponentissa suurin osa tarvittavasta toiminnallisuudesta löytyy automaattisesti ja esimerkiksi näkemisen etäisyyttä sekä kulmaa voi säätää tahtonsa mukaan.

AIPerception komponentissa on tapahtuma On Target Perception Updated, joka suoritetaan NPC-hahmon näkökentässä tapahtuvan muutoksen myötä. Tapahtuma hakee nimenomaan Actor tyyppisiä objekteja, jolloin se etsii käytännössä vain muita hahmoja. Ensimmäinen osio näköaistin toteutuksesta on kuvassa 19.



Kuva 19. Ensimmäinen osio näköaistin toteutuksesta AI controllerissa

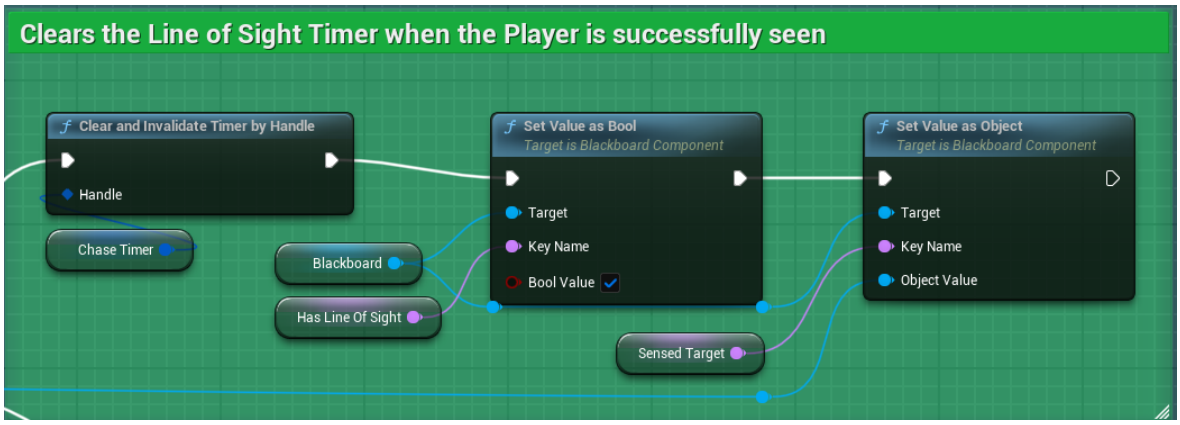
Tapahtuman käynnistyessä tarkistetaan, onko havaitulla kohteella merkki ”Player” ja onko näkemisen päivitys tapahtunut näköyhteyden muodostumisesta vai sen katkeamisesta. Mikäli kohteella ei ole oikeaa merkkiä, vaikka se olisi näkökentässä, tai näköyhteyttä ei ole, suoritetaan haaran epätoden puoli. Epätoden puoli johtaa kuvan 20 mukaisiin solmuihin.



Kuva 20. AI controllerin solmut epäonnistuneelle näköyhteydelle

Kuvan mukainen osuus on oleellinen lähinnä, kun näköyhteys on menetetty aikaisemmin tapahtuneesta jahtaamisesta. Näköyhteyden menettäminen johtaa neljän sekunnin ajastimen, ”Chase Timer”, asettamiseen ja sen aloittamiseen. Ajastimen käydessä loppuun Blackboardin HasLineofSight avain muutetaan epätodeksi, ja SensedTarget tyhjäksi. Ajastimen aika määritellään vihreässä Line Of Sight Timer muuttujassa. Pelaajan näkökulmasta NPC luovuttaa neljän sekunnin näköyhteyden menetyksen jälkeen.

Kun NPC onnistuneesti havaitsee ”Player” merkillä merkityn kohteen, suoritetaan kuvan 21 mukaiset solmut, jossa keskeytetään mahdollisesti käynnissä oleva ajastin ”Chase Timer”, asetetaan Blackboardin HasLineofSight avain todeksi ja SensedTarget havaituksi kohteeksi.

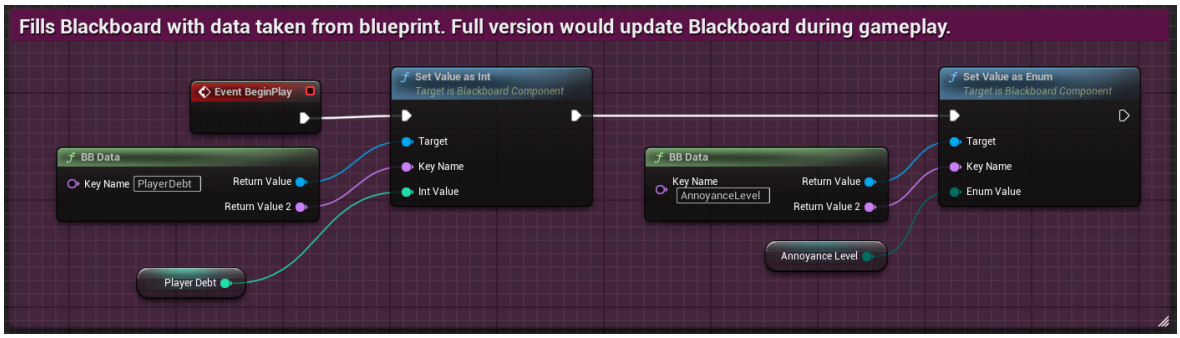


Kuva 21. AI controllerin solmut onnistuneelle näköyhteydelle

Onnistuneen näköyhteyden solmut ovat samat kuin epäonnistuneen näköyhteyden StartChaseTimer tapahtuman jäljessä olevat solmut, mutta päinvastaisin tarkoituksin. HasLineofSight avain asetetaan todeksi, jolloin Behavior Treen Chase sekvenssi toteutuu, ja SensedTarget avaimeksi asetetaan nähdyn kohteen tiedot. Kuvan 21 alin vaaleansininen viiva tulee kuvan 18 ensimmäisen solmun Actor pisteestä.

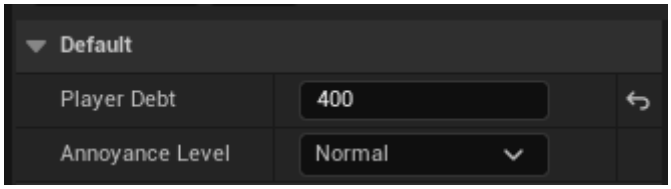
4.4.5 Blackboardin täyttäminen NPC Blueprintin tiedoista

Blackboardin avaimien PlayerDebt ja AnnoyanceLevel täyttämistä varten haetaan tiedot BP_DebtCollector Blueprintin omista tiedoista kuvan 22 osoittamalla tavalla.



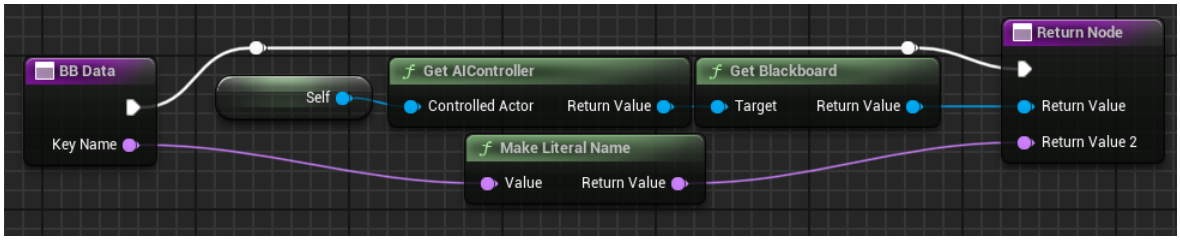
Kuva 22. Blackboardin loppujen tietojen täyttämisen solmut BP_DebtCollector Blueprintissä

Pelin alkaessa solmut hakevat BP_DebtCollector Blueprintin tiedoista vastaavat muuttujat, ja lisäävät ne määriteltyyn Blackboard avaimeen. PlayerDebt ja AnnoyanceLevel ovat julkisia muuttujia NPC:n omassa Blueprintissä, jolloin niitä voi säätää suoraan editorista kuvassa 23 näkyvällä tavalla.



Kuva 23. BP_DebtCollector Blueprintin tiedoissa PlayerDebt ja AnnoyanceLevel parametrit

BB Data on itse luotu funktio, johon syöttämällä halutun avaimen nimen hakee Blueprinttiin liitetyn Blackboardin ja palauttaa sen avaimen nimen kanssa. Funktio on kätevä, mikäli Blackboardin joutuu hakemaan useamman kerran eri avainten kanssa. Tämän funktion solmut näkyvät kuvassa 24.



Kuva 24. BB Data funktion solmut Blackboardin täyttämiselle

5 Yhteenveto ja pohdinta

Opinnäytetyön tavoitteena oli kehittää pelaajahahmon velkoja keräävälle NPC-hahmolle käytöksen logiikkaa hyödyntäen Unreal Enginen Blueprint, Behavior Tree ja AIPerception järjestelmiä. Opinnäytetyössä käsiteltiin esimerkkejä erilaisista NPC-hahmoista eri peleissä sekä Unreal Enginen ominaisuuksia ja verrattiin sitä kilpailijaansa Unityyn. Opinnäytetyön teoriassa käsiteltyjä asioita sovellettiin työn tuloksen toteuttamisessa.

Lopputuloksena syntyi prototyyppi NPC-hahmosta, joka vaeltaa pelin maailmassa 3–5 sekunnin pysähdyksin ja pelaajahahmon nähdessään ryhtyy jahtaamaan tätä. NPC palaa vaeltamaan jahtaamisen jälkeen, mikäli näköyhteys pelaajahahmoon katkeaa määritellyksi ajaksi. Näköyhteys voi katketa joko fyysisen esteen vuoksi, tai pelaajahahmon ollessa tarpeeksi kaukana NPC:stä. NPC-hahmon vaeltamis- ja jahtaamisnopeus muuttuu AnnoyanceLevel enumeraattorin mukaan, jota voi testaamista varten säätää editorista.

Tarkoituksena oli säätää NPC-hahmon nopeuden lisäksi näkemisen etäisyyttä AnnoyanceLevel enumeraattorin mukaan, ja tätä testattiinkin onnistuneesti itse tehdyllä funktiolla. NPC näki pelaajan progressiivisesti pidemmältä ärsyyntyneisyyden tason noustessa, mutta jahtaamisen luovuttaminen tapahtui vain ja ainoastaan näköyhteyden katketessa maksimiarvossa tai fyysisen esteen vuoksi. NPC-hahmon AIPerception komponentissa määriteltyä näkemis- ja näkemisen menettämisetäisyyttä ei ole mahdollista säätää pelin käydessä, joten koko hahmon toiminnallisuus pitäisi rakentaa aivan eri tavalla kunnollista toteutusta varten. Vaikka huomaamisen etäisyyden säätäminen periaatteessa toimi, pelaajan näkökulmasta hahmo näki pienimmällä ärsyyntymisen asteella vasta aivan vierestä, mutta ei luovuttanut millään ilman fyysisiä esteitä. Tämä ei vaikuttanut kovin reilulta pelikokemuksen puolesta, joten päätettiin olla toteuttamatta kyseistä toimintoa. Funktio on kuitenkin vielä projektissa, mikäli toimeksiantaja sitä haluaa iteroida.

Luotu NPC-hahmo luovuttaa jahtaamisen kanssa, mikäli sen näkökenttään vaeltaa jahtaamisen aikana muita hahmoja kuin pelaaja. Todennäköisesti jostain tuntemattomasta syystä StartChaseTimer tapahtuma laukeaa NPC-hahmon nähdessä minkä tahansa hahmon, jolla ei ole ”Player” merkkausta, vaikka pelaajahahmo olisi vielä näkökentässä. Ongelman voisi esimerkiksi ratkaista merkitsemällä pelaaja viholliseksi ja muut hahmot neutraaleiksi tiimijärjestelmän avulla, jolloin NPC-hahmon voisi merkitä havaitsemaan vain vihollisiksi merkatut hahmot. Tämän toteuttaminen kuitenkin menee omaa rooliani syvemmälle projektissa, joten toimeksiantaja saa itse päättää miten ratkaisee kyseisen haasteen.

Seuraavina askelina lopputuloksen kehittämisessä on logiikan toteuttaminen kokonaisessa NPC-hahmossa. Tämä edellyttää vähintäänkin AnnoyanceLevel enumeraattorin

muuttamista pelin aikana esimerkiksi pelaajalle kerääntyneen velan mukaan ja kiinni jäämisen toiminnallisuuden toteuttamista. Pelissä on jo valmiina lukuisia muita toimintoja NPC-hahmoille, joten niiden lisääminen velkojahahmolle tarpeen mukaan on mahdollista. On myös mahdollista muuttaa vaeltamisen toiminto toimimaan ennalta määriteltyjen pisteiden mukaan, mikäli NPC-hahmon halutaan kulkevan vain tietyillä alueilla.

Lähteet

Cataldi, L., Romero, M. & Sewell, B. 2022. Blueprints Visual Scripting for Unreal Engine 5. Birmingham: Packt Publishing, Limited.

Doucet, L. & Pecorella, A. 2021. Game engines on Steam: The definitive breakdown. Game Developer. Viitattu 25.10.2024. Saatavissa https://www.gamedeveloper.com/business/game-engines-on-steam-the-definitive-breakdown

Epic Games. 2025a. Downloading Unreal Engine Source Code. Viitattu 26.3.2025. Saatavissa https://dev.epicgames.com/documentation/en-us/unreal-engine/downloading-source-code-in-unreal-engine?application_version=5.4

Epic Games. 2025b. Licensing. Viitattu 26.3.2025. Saatavissa https://www.unrealengine.com/en-US/license

Epic Games. 2025c. Blueprints Quick Start Guide. Viitattu 1.4.2025. Saatavissa https://dev.epicgames.com/documentation/en-us/unreal-engine/blueprints-quick-start-guide?application_version=4.27

Epic Games. 2025d. Behavior Trees. Viitattu 13.3.2025. Saatavissa https://dev.epicgames.com/documentation/en-us/unreal-engine/behavior-trees-in-unreal-engine

Epic Games. 2025e. AI Perception. Viitattu 1.4.2025. Saatavissa https://dev.epicgames.com/documentation/en-us/unreal-engine/ai-perception-in-unreal-engine

Feng, J., Newton, P. 2016. Unreal engine 4 AI programming. Birmingham, Packt Publishing.

Gamepressure. 2021. Hitman 3: Stealth Guide. Viitattu 29.3.2025. Saatavissa https://www.gamepressure.com/hitman-iii/stealth/zde21e

GTA Wiki. Pedestrians. Fandom. Viitattu 19.11.2024. Saatavissa https://gta.fandom.com/wiki/Pedestrians

IMDb. 2025. Kingdom Come: Deliverance. Viitattu 29.3.2025. Saatavissa https://www.imdb.com/title/tt3414510

Johns, R. 2025. Unity vs Unreal: Which Game Engine? [2025 Update]. Hackr.io. Viitattu 26.3.2025. Saatavissa https://hackr.io/blog/unity-vs-unreal-engine

Kingdom Come: Deliverance Wiki. 2018a. Conspicuousness. Wiki.gg. Viitattu 29.3.2025. Saatavissa https://kingdomcomedeliverance.wiki.gg/wiki/Conspicuousness

Kingdom Come: Deliverance Wiki. 2018b. Visibility. Wiki.gg. Viitattu 29.3.2025. Saatavissa https://kingdomcomedeliverance.wiki.gg/wiki/Visibility

Kingdom Come: Deliverance Wiki. 2018c. Noise. Wiki.gg. Viitattu 29.3.2025. Saatavissa https://kingdomcomedeliverance.wiki.gg/wiki/Noise

Kingdom Come: Deliverance Wiki. 2022. Charisma. Wiki.gg. Viitattu 29.3.2025. Saatavissa https://kingdomcomedeliverance.wiki.gg/wiki/Charisma

Mafia Wiki. Police Officers (Mafia III). Fandom. Viitattu 5.11.2024. Saatavissa https://mafiagame.fandom.com/wiki/Police_Officers_(Mafia_III)

Unity Technologies. 2024. Plans and pricing. Viitattu 27.3.2025. Saatavissa https://unity.com/products?c=unity+engine

Velardo, V. 2019. The Reality of Red Dead Redemption 2's AI (Part 1). Medium. Viitattu 22.11.2024. Saatavissa https://medium.com/the-sound-of-ai/the-reality-of-red-dead-redemption-2s-ai-part-1-c276e9da2763