Samu Haaja

Goal-Oriented Action Planning ja Unreal Engine

Tradenomi
Tietojenkäsittely
Syksy 2022

KAMK • University of Applied Sciences

Tiivistelmä

Tekijä(t): Haaja Samu

Työn nimi: Tavoitteellinen toiminnansuunnittelu ja Unreal Engine

Tutkintonimike: Tradenomi, Tietojenkäsittely

Asiasanat: videopelit, tavoitteellinen toiminnansuunnittelu, GOAP, Unreal Engine, tekoäly, simulaatio, projekti, blueprint, c++, agent, worldstate, suunnittelu, toiminta

Videopelien kehityksessä tekoälyn osuus on jo merkittävän suuri, ja sille asetetut vaatimukset kasvavat jatkuvasti. Tekoälyn toteutukseen on olemassa lukuisia erilaisia tekniikoita, joista myös useita on jo valmiiksi saatavilla implementoitavaksi. Pelimoottorit kuten Unreal ja Unity tuovat yksinkertaiset tekoälyn työkalut suoraan käyttäjälle. Pelinkehityksessä on kuitenkin myös erittäin tärkeää, että tekoälyn toteutus soveltuu oikein sen käyttötarkoitukseen ja monimutkaisempien tekoälyn ratkaisujen löytäminen onkin jo huomattavasti haasteellisempaa.

Opinnäytetyön tavoitteena olikin tutustua itselleni tuntemattomaan tekoälytekniikkaan sekä punnita sen mahdollisuuksia ja käytännöllisyyttä etenkin monimutkaisemman tekoälyn rakentamisessa. Tähän tutustumiseen kuului olennaisena osana tekniikan logiikan sisäistäminen ja toimivan esimerkin rakentaminen. Esimerkkiprojekti rakennettiin Unreal Enginen C++ kielellä, jolloin moottorin muita ominaisuuksia voitiin käyttää visualisoimaan tekoälyn toimintaa.

Vertailun vuoksi opinnäytetyössä käsitellään myös kevyesti muita tunnettuja AI tekniikoita, jotta nähdään minkälaisia eroavaisuuksia ja käyttötarkoituksia näillä on.

Valmiin projektin toimintatapaa ja logiikka tarkastellaan myös omassa teknisemmässä kappaleessa lähemmin. Tämän jälkeen käsitellään projektin tekijälleen tarjoamia opetuksia.

Abstract

Author(s): Haaja Samu

Title of the publication: Goal-Oriented Action Planning and Unreal Engine

Degree title: Bachelor of Business Administration, Business Information Technology

Keywords: video games, Goal-oriented action planning, GOAP, Unreal Engine, AI, artificial intelligence, simulation, project, blueprint, c++, agent, worldstate, planning, action

The significance of artificial intelligence in game development is enormous, and the set requirements for it proceed to grow. There are numerous different techniques for the implementation of artificial intelligence, several of which are already readily available for implementation. Game engines such as Unreal and Unity bring simple artificial intelligence tools directly to the user. However, in game development it's also very important that the implementation of AI is suitable for its purpose and finding readily usable complex artificial intelligence solutions is considerably more challenging.

The objective of the thesis was to become familiar with an AI technique formerly unknown to the author, and to weigh its possibilities and practicality, especially in building more complex AI behavior. An essential part of this familiarization was internalizing the logic of the technique and building a working example. The example project was built in Unreal Engine's C++, which enables the use of the engine's other features to visualize the operation of the artificial intelligence.

The thesis also lightly glosses over other well-known AI techniques to analyze the differences and purposes they serve in comparison to Goal-oriented action planning.

The finished project's operation methods and logic are examined more closely in their own separate technical chapter. After this, the experience and insight gained during the project are discussed.

Sisällys

1 Johdanto

Videopelien tekoäly on nykyään olennainen osa pelikokemusta. Jos pelistä löytyy eläviä hahmoja, jokainen pelaaja osaa jo olettaa näiden hahmojen käyttäytyvän vähintäänkin loogisesti. Tämä oletus voi vaikuttaa pieneltä tekijältä, mutta pelihahmot ja niiden tekoäly ovat pelien alusta alkaen olennainen osa pelimaailmanrakennusta. Pelaajan kokemus pelistä ja sen tekoälystä on usein neutraali, niin kauan kun älykkyyden illuusio säilyy, jos tekoäly ”hajoaa” kesken pelin, tulee pelaajalle epämiellyttävä muistutus siitä, että hahmot eivät ole oikeasti älykkäitä.

Opinnäytetyön aiheena on tutustuminen tekoälyn erilaisiin tekniikoihin, ja kuinka niiden avulla voidaan rakentaa uskottavia pelikokemuksia pelaajalle. Opinnäytetyö voidaan jakaa kahteen osaan: teoreettiseen tarkasteluun ja käytännön projektiin, jossa toteutetaan ja esitellään yksi monimutkaisempi tekoälyn toimintaperiaate. Teoreettinen tarkastelu vertailee yleisesti käytettyjä tekniikoita sekä korostaa niiden hyviä ja huonoja puolia. Käytännön projektin tekoälyn teoria käsitellään myös tarkasti läpi, jotta toiminnan logiikka on ymmärrettävissä, ilman edeltävää kokemusta.

Käytännön projektin on tarkoitus toimia esimerkkinä tekniikan toimintatavasta ja samalla myös arviona sen käytännöllisyydestä.

2 Tekoälyn toteutuksesta peleissä

Tekoäly on erittäin laaja käsite ja tästä syystä termillä ei usein tarkoiteta peleissä nähtyjä tietokonehahmojen tekoälyä. Tekoäly on nykypäivänä osana lähes kaikkia tekniikan aloja, ja sen pitkälle edistynyt kehitys on mahdollistanut ennennäkemättömiä saavutuksia. Uudenlaiset tekoälyn tekniikat, kuten esimerkiksi itsestään oppiva tekoäly ovat laajentaneet tekoälyn käytännön mahdollisuuksia.

Tekoäly käsiteenä on niin laaja, että peleissä käytettävä tekoäly on syytä jakaa täysin omaan kastiinsa. Peleissä käytettävä tekoäly on usein tarkoituksella ennalta-arvattava, pelkistetty ja helposti lähestyttävä, sillä sen pääsääntöisenä tehtävänä on toimia viihteenä pelaajalle. Haetaan haastavaa vastusta, ilman turhauttavaa vaikeutta. Tällaisen tekoälyn kehityksessä voidaan puhua ”älykkyyden illuusiosta” (Buckland, 2005)

Tekoälyn merkitys pelinkehityksessä on myös kasvanut huomattavasti ja sen kehityksestä on nopeasti tullut yksi pelinkehityksen tärkeimmistä kulmakivistä. Oikeanlainen käyttäytyminen ja luontevat tekoälyhahmot voivat syventää pelaajan kokemusta pelimaailmasta. Pelimaailmoissa oleskelevat hahmot ovat tärkeä osa tarinan ja maailmanrakentamista pelaajalle. Tämän vuoksi tällaisen luontevan tekoälyn kehityksestä ja käyttämisestä voi löytää projektiin uudenlaista syvyyttä ja rakennetta.

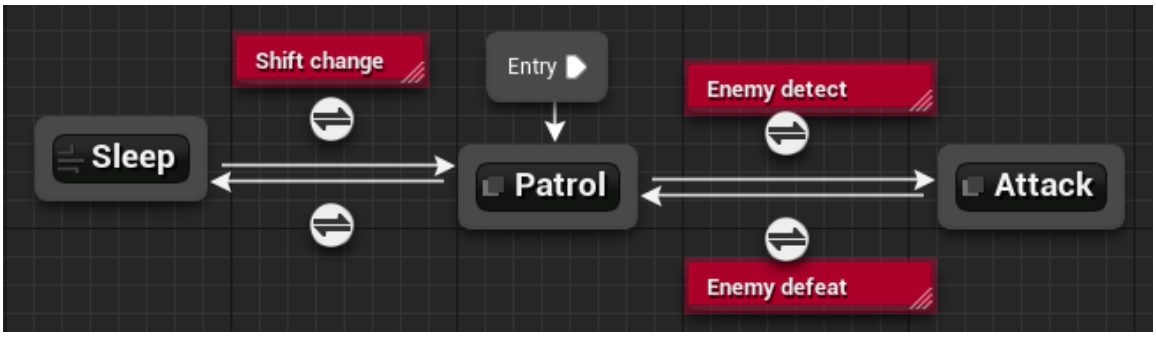
3 Tunnettuja tekoälyn toteutuskeinoja

Tekoälyn toteutukseen on kehittynyt useita tekniikoita ja näitä lähes standardeiksi muodostuneita kaavoja voidaan helposti muovata omaan käyttöön sopivaksi. Käyttöönoton avuksi löytyy helposti esimerkkejä, ja pelimoottoreista löytyy jo valmiita implementointeja.

3.1 Äärellinen tila-automaatti (FSM)

Äärelliset tila-automaatit (FSM) ovat käsitteellisesti yksinkertaisia ja nopeita koodattavia, mikä johtaa tehokkaaseen ja joustavaan tekoälyrakenteeseen vähäisellä suorituskyvyn lisäkululla. Ne ovat intuitiivisia ja helppo visualisoida, mikä helpottaa kommunikointia vähemmän teknisten tiimin jäsenten kanssa. (Rabin, 2014, 48.)

FSM hajottaa tekoälyhahmon kokonaisvaltaisen tekoälyn pienempiin erillisiin osiin, joita kutsutaan tiloiksi. Jokainen tila edustaa tiettyä käyttäytymistä tai sisäistä konfiguraatiota ja vain yhtä tilaa pidetään "aktiivisena" kerrallaan. Tilat yhdistetään siirtymillä, suunnatuilla linkeillä, jotka ovat vastuussa vaihtamisesta uuteen aktiiviseen tilaan aina, kun tietyt ehdot täyttyvät. (Rabin, 2014, 48.).



Kuva 1. Esimerkki tekoälyn äärellisestä tila-automaatista Unreal Enginessä.

Voidaan siis havaita, että FSM on erinomainen tekniikka toteuttamaan yksinkertaisia toimintasarjoja, jossa tilamuutoksille voidaan määrittää selkeät säännöt.

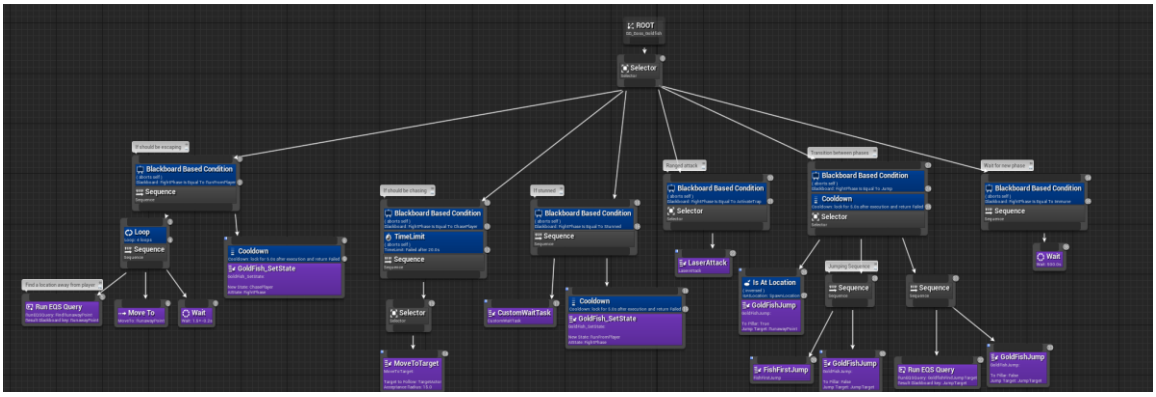
Kuitenkin vakavana puutteena on laajentuvuus, sillä kun eri tilamuutoksia ja toimintasarjoja lisätään hahmolle, niin vaadittu työmäärä kasvaa eksponentiaalisesti. Käyttäen Kuva 1. esimerkkiä

apuna; jokainen uusi lisätty tila vaatii siirtymän ”Attack” tilaan, sillä hahmo, jonka tarkoitus on hyökätä kohdatessaan vihollisen, täytyy loogisesti pystyä hyökkäämään monessa muussakin tilanteessa. Tämän jälkeen täytyy myös olla selvitettävissä mihin tilaan hahmo palautuu hyökkäyksen jälkeen. Jokainen uusi käytös vaatii useamman, kun yhden siirtymän, ja FSM:n toiminta muuttuu entistä epäselvemmäksi.

3.2 Käytöspuu (Behavior Tree)

Käytöspuu on AI-tekniikka, jonka konsepti on yksinkertainen. Käytöspuun isänä voidaan pitää aiemmin mainittua FSM tekniikkaa, käytöspuun on tarkoitus säilyttää tekniikan yksinkertaisuus, mutta poistaa laajentuvuuden ongelmat. Käytöspuu pyrkii säilyttämään FSM:n selkeän ”tila” logiikan, ja keskittyy korjaamaan ongelmat siirtymissä.

Käytöspuu on visuaalisesti nimensä mukainen. Käytöspuusta löytyy juuri (root), josta hahmon toimintalogiikka jaetaan erillisiin oksiin ja puun omaisesti nämä oksat voivat jakaantua. Aivan oksien kärjistä löytyy tekoälylle halutut suoritettavat toiminnot, eli ”lehdet”. Näitä edeltävät hajaantuvat oksanhaarat sisältävät ehtoja, jotka määrittävät onko tätä polkua mahdollista tai syytä seurata.



Kuva 2. Käytöspuu Unreal Enginessä. Siniset ovat ehtoja, violetit toimintoja

Visuaalisesti käytöspuuta on huomattavasti yksinkertaisempi seurata, kuin edeltävää FSM mallia, etenkin jos ehtoja ja erilaisia käytöksiä on useita. Käytöspuuhun tehtävät muutokset ovat myös selkeämpi toteuttaa, sillä toimintojen välisiä siirtymäehtoja tarvitaan huomattavasti vähemmän.

Käytöspuun toimintaperiaatteen vuoksi tehtävät ovat aina myös lukitussa tärkeysjärjestyksessä, sillä ensimmäinen mahdollinen tehtävä on hyväksyttävä ja tehtävien tarkastus tapahtuu aina loogisesti vasemmalta oikealle. Tämä itsessään luo ennalta-arvattavaa käyttäytymistä, jonka hyvä-puoli on hahmojen käyttäytymisen helppo implementointi.

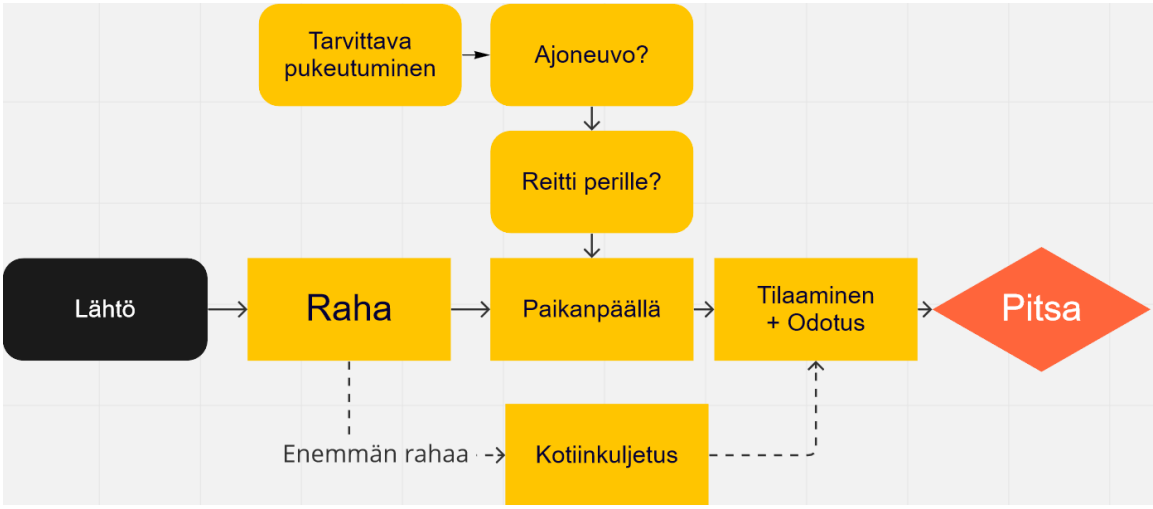
Käytöspuu tekniikalla on kuitenkin useita puutteita. Erittäin suurilla käytöspuilla koko puun arvioinnin suorituskustannukset voivat olla kohtuuttomat. Tämän seurauksena ajatus alipuista on otettu käyttöön. Ajatuksena on, että alipuu voi jatkaa suorittamista kutsumatta koko puuta, kunnes jokin ehto alipuusta poistumiselle täyttyy. Tämä kuitenkin palauttaa osan monimutkaisuudesta, joka liittyy tilojen välisen siirtymisen ohjaukseen ja virheenkorjaukseen Finite State Machinesissa. (Rasmussen, 2016)

4 Tavoitteellinen toiminnansuunnittelu (Goal-Oriented Action Planning (GOAP))

Tavoitteellinen toiminnansuunnittelu, lyhyemmin GOAP on edellisiin verrattuna täysin erilainen lähestymistapa pelien tekoälyyn, jonka tarkoitus on siirtää toiminnansuunnittelu tekoälylle itselleen. Tällöin tekoälylle ei ole enää tarvetta määrätä kaavamaisia siirtymisiä toimintojen välille, tämän sijaan annetaan päämäärä, sekä toimia, joita yhdistämällä tekoäly pyrkii itse saavuttamaan halutun lopputuloksen.

Tällainen tekniikka mahdollistaa itsenäisen päätöksenteon, jota voidaan säädellä muuttamalla tehtävien ja erilaisten toimintojen ”hintaa”. Itsenäinen päätöksenteko voi tehdä pelihahmoista maailmassa toisistaan erottuvia, sillä suunnitelmat eroavat toisistaan. Jokainen pelihahmo toimii oman suunnitelmansa mukaisesti, jonka muodostamiseen hahmon oma tilanne on otettu huomioon.

GOAP pohjimmiltaan muistuttaa paljon ihmiselle ominaista ajattelutapaa ja ongelmanratkontaa. Yksinkertaisena esimerkkinä: Päämääränä pitsan syöminen



Kuva 3. Esimerkki tarvittavista suunnitelmista, jos tavoitteena on pitsan syönti.

Tiivistettynä GOAP antaa tekoälylle tavoitteen: Pitsan syöminen, ja sen yksinkertainen suoritusehto on, että hahmolla on pitsaa. Tämän toteutumiseksi sellainen täytyy tilata, ja tilaamisen ehtona voi olla tietty sijainti. Tavoite siis puretaan takaperin yksinkertaisiin askeliin, joita seuraamalla ongelma on ratkaistu.

Tällaisella suunnittelujärjestelmällä on kolme selvää hyötyä. Ensimmäinen hyöty on sen kyky erottaa tavoitteet ja toimet, jolloin erityyppiset hahmot voivat saavuttaa tavoitteet eri tavoin. Toinen suunnitelmajärjestelmän etu on helpotus yksinkertaisten käyttäytymismallien kerrostamiseen

monimutkaiseksi havaittavan käyttäytymisen tuottamiseksi. Kolmas hyöty on hahmojen vahvistaminen omalla dynaamisella ongelmanratkaisukyvyllä. (Orkin, 2006, 7.)

Tavoitteellisella toiminnansuunnittelulla on siis mahdollista luoda monimutkaisiakin tapahtumaketjuja, ilman että ne täytyy ennalta määrätä. Tällainen tekoäly sopii parhaiten kuitenkin projekteihin, joissa sen monimuotoisuudelle ja itsenäisyydelle on tarvetta. Valmis toiminnansuunnittelu voi jopa vähentää tarvittua työmäärää, jos hahmot usein osaavat toimia pelin tarkoitusten mukaisesti, joka vähentää tarvetta niin sanotulle käsikirjoitetulle käytökselle.

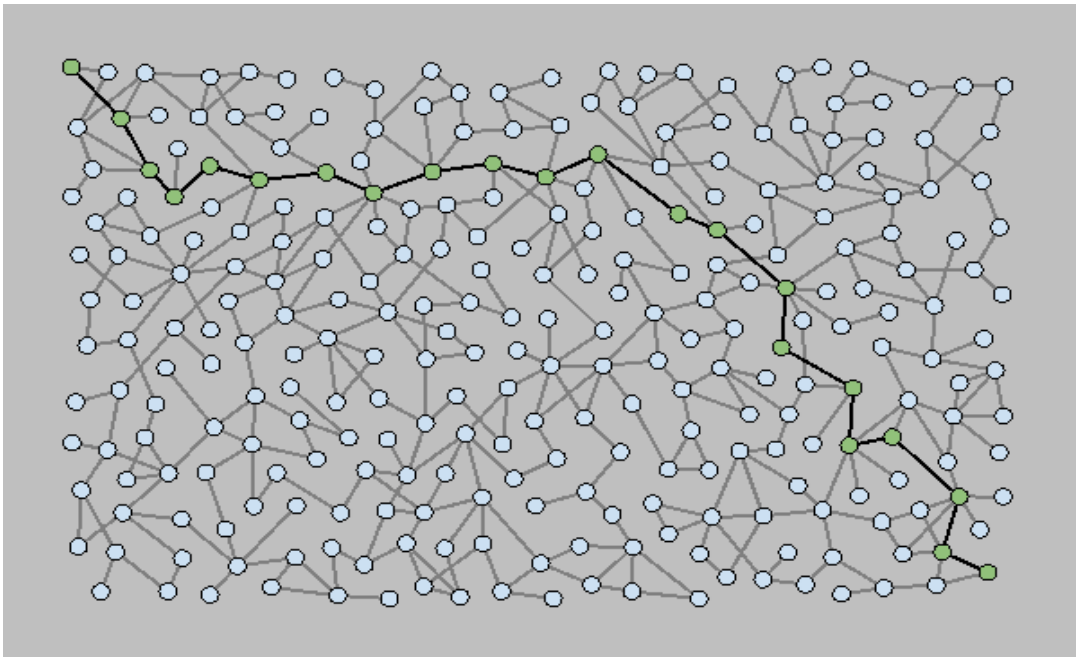
4.1 Kuinka GOAP yhdistää tehtävät ja oman tavoitteen suorittamisen

Tavoitteelliseen toiminnansuunnitteluun ei siis kuulu ennestään minkäänlainen käytöksen käsikirjoitus, kaikki toiminta ja päätökset ovat orgaanisia. GOAP:iin kuuluu myös yleisesti parhaan mahdollisen suunnitelman luominen. Päätöksenteossa on tärkeää, että eri vaihtoehtoja voidaan vertailla ja punnita toisiaan vastaan. Tämän vuoksi erilaisille toimille on syytä määrittää painoarvo, eli hinta.

Jos määritämme kustannuksen per toimenpide, voimme pakottaa hahmon valitsemaan yhden toimen toisen sijaan. Tässä kuvaan tulee vanha ystävämme A*. Nyt kun meillä on kustannusmittari, voimme käyttää tätä hintaa ohjaamaan A*-hakuamme kohti edullisinta toimintosarjaa jonkin tavoitteen saavuttamiseksi. (Orkin 2006, 11.)

4.2 A* algoritmi

Normaalisti A*:ta ajatellaan keinona löytää navigointipolku, ja tähän sitä käytetään myös F.E.A.R - pelissä, eli polkujen löytämiseksi navigointiverkon läpi. Tosiasia on kuitenkin, että A* on todella yleisesti käytetty hakualgoritmi. A*:ta voidaan käyttää lyhimmän polun etsimiseen minkä tahansa solmugraafin läpi, kun solmut ovat toisiinsa yhdistettyinä reunoista. Navigoinnin tapauksessa on intuitiivista ajatella navigointiverkon monikulmiot solmuina ja monikulmioiden reunat reunoina kaaviossa, jotka yhdistävät yhden solmun toiseen. (Orkin 2006, 11.)



Kuva 4. Visuaalinen esimerkki solmugraafista (Happy Coding, n.d)

Toiminnansuunnittelussa solmut ovat maailmantiloja ja polku etsitään tavoite maailmantilan löytämiseksi. Reunat, jotka yhdistävät eri maailmantiloja ovat toimia, jotka johtavat maailmantilan muuttumiseen yhdestä toiseen. A*:ta siis käytetään sekä navigointiin että suunnitteluun F.E.A.R.:issa, ja molemmissa tapauksissa hakua käytetään täysin eri tietorakenteissa. (Orkin 2006, 11.)

5 Unreal Engine

Unreal Engine on Epic Gamesin vuonna 1998 kehittämä pelimoottori, joka oli alun perin kehitetty yhtiön omia FPS-pelejä varten. Tämän jälkeen pelimoottoria ja sen sisältämiä ominaisuuksia on kehitetty jatkuvasti. Unrealin käyttö pelinkehityksessä on yleistynyt huomattavasti ja etenkin Unreal Engine 5:n julkaisun jälkeen pelimoottori on saanut erittäin paljon huomiota myös mediassa.

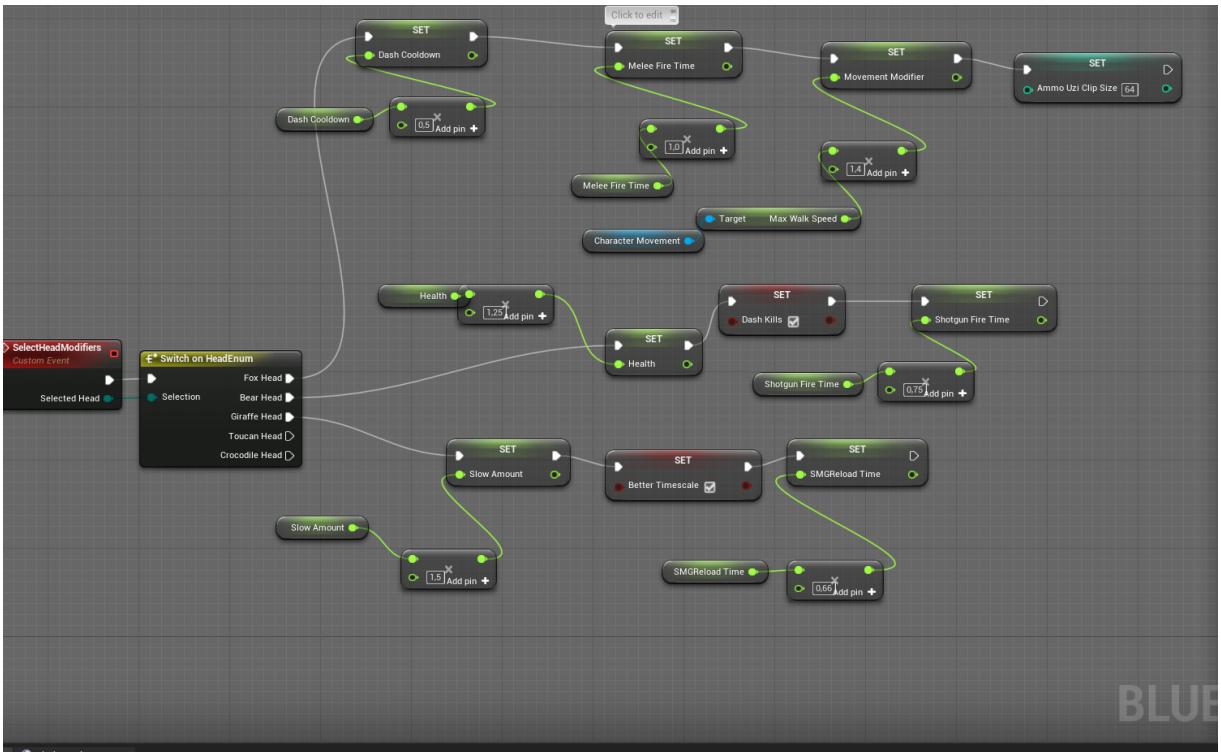
Epic Games julkaisi vuonna 2009 Unreal Engine 3- pelimoottorista kokonaisen valmiin pelinkehitysalustan ”Unreal Deveploment Kit” (UDK), joka sisälsi myös useita pelinkehitykseen olennaisia työkaluja. (IGN, 2009)

Pelinkehitysympäristö on tämän jälkeen ollut jatkuvasti kehityksessä, ja uusin versio Unreal Engine 5 julkaistiin virallisesti huhtikuussa 2022. (Unreal Engine, 2022)

Unreal Engine mahdollistaa nopeamman pelinkehityksen, sillä sen tarjoamilla työkaluilla prototyyppien ja projektien aloittaminen on nopeampaa. Alusta tarjoaa valmiiksi kaiken tarvittavan yksinkertaisen prototyypin valmistamiseen. Pelimoottorin ohjelmointikielenä toimii C++ sekä Unrealin oma visuaalinen ohjelmointikieli ”Blueprint”.

5.1 Visuaalinen Blueprint-ohjelmointi

Visuaalinen ohjelmointikieli Blueprint yksinkertaistaa tavanomaista ohjelmointikieltä käyttämällä selkeitä laatikoita, joita yhdistämällä toiminallisuuden yhteydet saadaan visualisoitua. Aivan kaikkea pelimoottorissa ei ole mahdollista toteuttaa Blueprinttejä käyttämällä, mutta yksinkertaisiin toteutuksiin kieli on erinomainen ja helppokäyttöinen.



Kuva 5. Esimerkki Unreal Enginen blueprint ohjelmoinnista.

5.2 Tekoälytyökalut

Unreal Engine sisältää myös tekoäly ohjelmointia helpottavia työkaluja. Pelimoottorista löytyy valmiit Finite-state machine ja Behavior Tree implementaatiot ja näitä voi käyttää myös yhtäaikaisesti hahmojen käyttäytymislogiikan luomiseksi. Hahmoille on olemassa myös valmiiksi implementoitavia inhimillisiä ominaisuuksia, kuten tuntoaisti, liikkeen ennakointi, näkö, kuulo ja joukkuetunto.

5.3 C++ -rajapinta

Unreal Engine on rakennettu pääasiallisesti C++ kielellä ja sen käyttö on edelleen tärkeä osa pelimoottoria ja projektien rakentamista. C++ on mahdollista käyttää yhdessä Blueprint järjestelmän kanssa, ja tämän avulla on mahdollista rakentaa monimutkaisia toiminnallisuuksia kirjoitettuna C++ koodina, jonka jälkeen niiden toiminnallisuus ja käyttö voidaan siirtää yksinkertaisiin Blueprint funktioihin.

C++ toimii kuitenkin aina pohjana, josta Blueprint voi noutaa tietoa, sillä Blueprintin tietoja ei ole suunniteltu siirrettäväksi C++ kieleen.

Unreal Engine C++ on itsessään hyvin erilaista perinteiseen C++ kieleen, ja aiheutti projektin aikana useita ongelmatilanteita käyttäytyessään eri tavalla, kun pelimoottorin dokumentaatio antaa olettaa.

6 AI tekniikan suunniteltu toteutus Unreal Enginessä

6.1 Projektin lähtökohdat

Projektin alustavana tavoitteena oli kauppasimulaatio, jossa pelissä on asiakkaita ja kaupan työntekijöitä, ja ainakin kaupan työntekijöille voi asettaa useita erilaisia tehtäviä kuten hyllyjen täyttö, siivous ja kassalla työskentely. Tavoitteena olisi myynnin maksimointi.

Tämän toteuttamiseksi aivan ensimmäisenä lähtökohtana projektissa on valmiin Goal-oriented action planning pohjan rakentaminen.

Tämä osa vaati eniten suunnittelua, sekä tuntemusta pelimoottorin ja GOAP tekniikan rajoituksista. Ennalta tuntemattomia rajoituksia ja ongelmia projektin aikana tuli arvattavasti lisää, osittain oman rajallisen kokemuksen vuoksi.

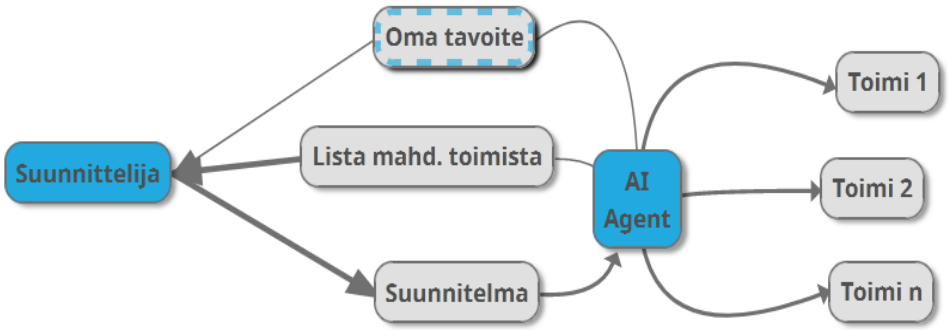
Goal-Oriented Action Planningin implementoinnissa ensimmäisenä askeleena on rakentaa kaikki tarvittavat osat, joita logiikka tarvitsee erilaisten toimien ja niiden vaikutusten tunnistamiseksi. Näiden yhteisvaikutuksen avulla voidaan luoda pohja suunnittelulle.

Tätä on kuitenkin mahdotonta tiivistää selkeästi, joten käyn työssä läpi kaikki GOAP:in olennaiset palaset ja niiden toiminnallisuuden tavoitteet.

6.2 AI agenttien tavoitteet

GOAP:issa hahmojen tavoitteen tarkoitus on merkata hahmon haluamaa maailmantilaa; jokin tapahtuma, tai tilanne, jonka tavoittamista varten voidaan muodostaa selkeä suunnitelma.

Projektissa hahmoille lisätään myös omia alatavoitteita, kun sellaisen suorittamiselle on tarvetta päätavoitteeseen pääsemiseksi, tämän avulla voidaan paloitella suunnitellun kokoa. Hahmolla voi olla esimerkiksi alatavoite hankkia tarvittava työkalu tai esine, jonka työtehtävän suorittaminen vaatii.



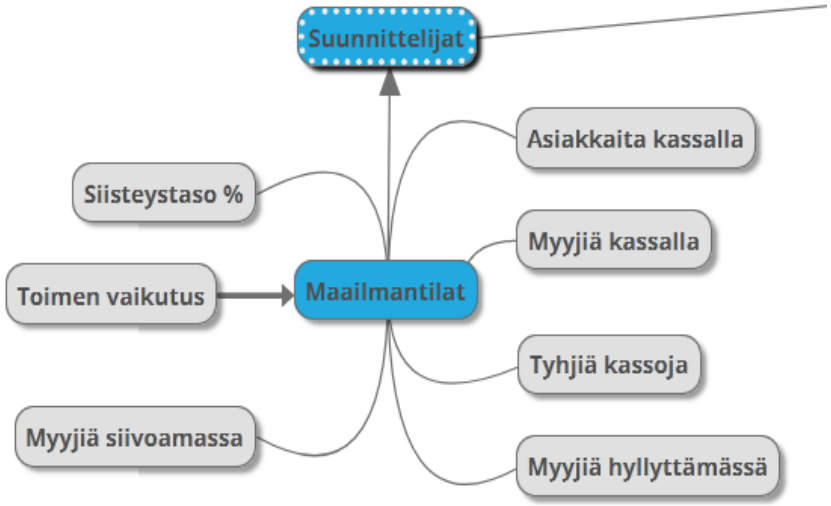
Kuva 6. GOAP Tavoitteen käyttö

Projektissa hahmojen tavoite on aina muokattavissa ja vaihdettavissa, on kuitenkin otettava huomioon että, hahmon eri toimet voivat mahdollistaa tavoitteen saavuttamisen, sillä muuten suunnitelmaa ei ole mahdollista kehittää. Tässä projektissa myyjähahmojen päätavoite on pitää kaupan tuottavuus mahdollisimman korkealla, kun taas hahmojen omat alatavoitteet määrittelevät ovatko hahmot suunnitelmiensa mukaisissa sijainneissa ja onko suunnitelma vielä mahdollista suorittaa.

6.3 Pelimaailman tilat (worldstates)

Maailmantilat käsittävät eri muuttujia, jotka kuvastavat pelissä tapahtuvia asioita sekä sen hetkistä tilannetta. Nämä arvot ovat olennaisesti sellaisia, joita seuraamalla on mahdollista määrittää mitkä toimet ovat tarvittavia, tai mitä toimia ei ole mahdollista suorittaa. Yksittäinen ”maailmantila” siis kuvastaa maailmassa olevaa yksittäistä asiaa, ja tätä seuraamalla on mahdollista nopeasti saada tarvittava tieto toiminnansuunnitteluun.

Maailmantilat ovat siis merkittävä osa suunnitelman luontia ja koko GOAP:in toimintatapaa. Tämän vuoksi maailmantilat päivitetään jatkuvasti, kun niille olennaisia muutoksia tapahtuu. Hahmot luovat jatkuvasti uusia suunnitelmia maailmantiloja hyväksikäyttäen.



Kuva 7. GOAP Maailmantilojen toiminta

Maailmantila muuttujaa käytetään myös hahmon sisäisesti, kuvaamaan hahmon omaa tilannetta. Muuttujan avulla voidaan seurata hahmon omaa tilannetta, joka vaikuttaa suunnitelman muodostukseen ja saattaa kokonaan evätä tietyt toimet, jos hahmolla ei ole toimen vaatimaa tilannetta.

Tärkeimpiin maailmantiloihin projektissa kuuluu tilat, jotka määrittävät eri toimien tärkeys. Maailmantilan ”asiakkaiden määrä kassalla” muutokset muuttavat myös kassatyöntekijöiden tavoitelukumäärää ja näin aiheuttaa suunnittelussa käytettyjä painoarvoja. Suurempi määrä asiakkaita varaa työntekijöitä, pienempi määrä taas laskee. Suunnittelussa otetaan kuitenkin huomioon muut tehtävät, joten kassa muutokset tapahtuvat vain, jos niiden tärkeys muuttuu myyjähahmojen suunnittelijoissa muita tehtäviä suuremmaksi.

Muita esimerkkejä maailmantiloista projektissa on: Tuotteen x määrä hyllyssä, tai myymälän siisteystaso.

Hahmon suunnitelman muodostuksessa yksi tärkeimmistä pilareista on tietyn maailmantilan tavoittelu. Kuinka saadaan tilanne, jossa kassalla ei ole liikaa jonoa, tuotteitta on riittävästi, ja siisteystaso ei ole laskenut liian matalalle. Etsitään siis maailmantilaa, jossa myymälän tuottavuus on korkein mahdollinen.

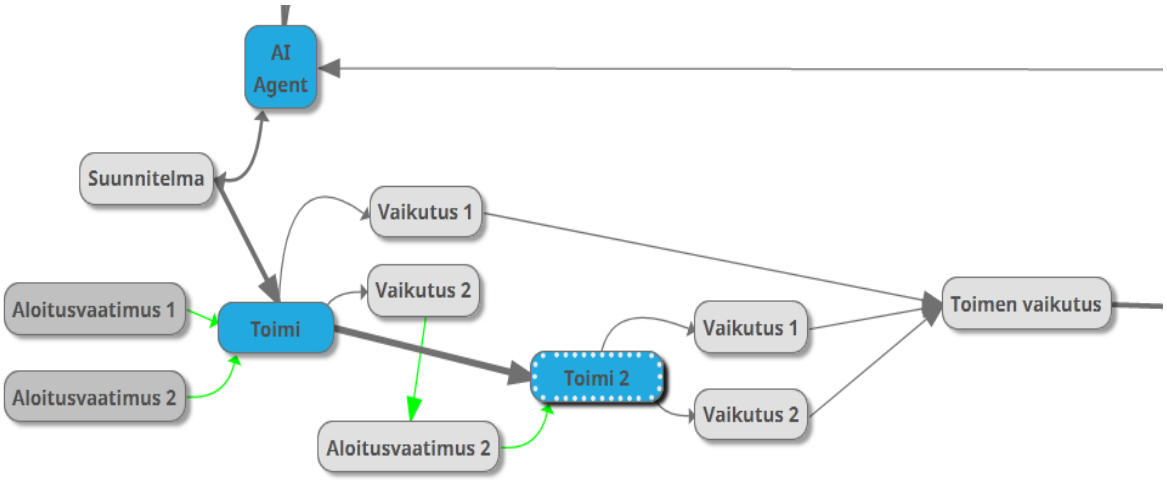
6.4 AI agenttien toimet

Agentin toimella tarkoitetaan yksittäistä hahmon omaavaa toimea, jolla on selkeä aloitus ja lopetus. Kaikki GOAP hahmot omistavat useita eri toimia, sillä näiden toimien ja niiden ennakkoehtojen ja vaikutusten yhdistäminen selkeiksi toimintaketjuiksi on koko suunnitelman perusta.

Toimien perustaan kuuluu oma hinta, eli kuinka raskasta toimi on suorittaa, sillä osa toimista voi olla vähemmän haluttuja. Tämä hinta on aina liukuva, sillä siihen voidaan vaikuttaa maailman tai hahmon tilanteella. Esimerkiksi hahmon etäisyys kahteen mahdolliseen toimeen voi ääritapauksessa vaikuttaa siihen kumman suoritus on kannattavampaa.

Suunnittelun mahdollistamiseksi toimille on tärkeää myös asettaa selkeät aloitusvaatimukset. Nämä aloitusvaatimukset voivat liittyä sen hetkiseen maailmantilanteeseen, tai myös hahmon omaan tilanteeseen. Esimerkkinä voidaan esittää, että hahmo ei voi aloittaa täyttämään tuotetta hyllyyn, jos ei ole hyllyn luona, eikä hahmo voi palvella asiakasta, jos asiakkaita ei ole kassalla.

Hahmon suorittaman toimella on myös syytä olla selkeä loppuvaikutus, joka kuvastaa millaisen muutoksen kyseinen toimi tekee maailmaan, ja hahmon omaan tilanteeseen. Tämä on olennainen osa maailmantilanteen päivittämistä ja hahmon luoman suunnitelman edistämistä seuraavaan vaiheeseen. Esimerkiksi kun hahmo palvelee asiakkaan kassalla, maailmantila, joka seuraa jonon pituutta vähenee yhdellä.



Kuva 8. Toimet ja niiden vaatimukset ja vaikutukset

Kassan ruuhkautumistilanne voi aiheuttaa myyjähahmolle suunnitelman muutoksen, jossa hahmo siirtyy kassatyöskentelyyn. Tämän toimen aloitusvaatimuksia ovat vapaan kassan olemassaolo, asiakkaat kassalla ja hahmon sijoitus kassalle. Toimen aloitus vähentää muualla myymälässä olevien työntekijöiden määrää, ja jokainen palveltu asiakas vähentää jonottavien kokonaismäärää. Suunnittelija valitsee kannattavamman toimen hahmolle, kun sellainen löytyy.

Kaupassa asioivat asiakkaat aiheuttavat jatkuvasti myymälän siisteystason vähenemistä, ja myyjähahmoilla on mahdollisuus siivota, jonka avulla siisteystasoa saadaan nostettua. Tämän toimen aloitusvaatimuksena on, että myymälässä on siivottavaa, ja hahmolla on siivousvälineet. Siisteystason prioriteetti ei ole aluksi suuri, mutta sen laskiessa huomattavan alhaalle sen arvo ylittää lähes kaikki muut mahdolliset toimet.

Asiakkaiden toimet aiheuttavat myös jatkuvasti hyllyissä olevien tuotteiden vähenemistä ja tuotteet voivat myös loppua. Tähän myyjähahmot voivat vastata keräämällä tuotteita varastosta ja tuomalla niitä myyntiin. Tuotteiden määrän lisäämisen tärkeys ei ole suuri, ellei kyseisen tuotteen määrä laske huomattavan vähäiseksi.

6.5 Tehtävien suunnittelun toiminta (Action planning)

Edellisessä kappaleessa käsitellyt toimet ovat kriittisin osa suunnittelua, sillä näiden yhdistäminen on tavoitteellisen toiminnansuunnittelun perusta.

Suunnittelijan toiminta muistuttaa useasti pelinkehityksessä käytettävää polun etsintää, aivan ensimmäisenä tarkastellaan haluttua tavoitetta; Mikä maailmantilanne on määränpäänä. Tämän jälkeen tarkastellaan hahmon omistamia toimia, ja aiemmin mainittuja toimien aiheuttamia loppuvaikutuksia; Edistävätkö jotkin näiden toimien loppuvaikutuksista maailmantilaa haluttuun suuntaan?

Jos tällaisia toimia löytyy, on tarkasteltava näiden toimia aloitusvaatimuksia. Tämä prosessi toimii samalla tavalla kuin määränpään tarkastelu; Mitkä toimet auttavat täyttämään vaadittavat aloitusvaatimukset?

Tällainen prosessi toistuu niin kauan, kunnes vastaan tulee toimia, joiden suorittamisen hahmo voi jo aloittaa. Tässä vaiheessa voidaan tutkia edeltäneitä vaihtoehtoja, joka ovat muodostaneet mahdollisesti useita erilaisia tapoja päästä haluttuun lopputulokseen, ja seuraavana askeleena on valita niistä hahmolle *halvin* vaihtoehto, vertailemalla toimien aiemmin mainittuja hinta-arvoja.

Suunnitelman valinta käyttää yleisesti tunnettua ja monikäyttöistä A* algoritmia halvimman toimintaketjun valitsemiseen. A* tunnetaan parhaiten sen ominaisuudesta löytää aina lyhyin reitti määränpäähän, tässä tapauksessa etäisyyksien sijaan vertaillaan eri toimien hintoja ja mahdollisen toimintasuunnitelman kokonaiskustannusta, joten halvimman reitin löytäminen tapahtuu samalla periaatteella.

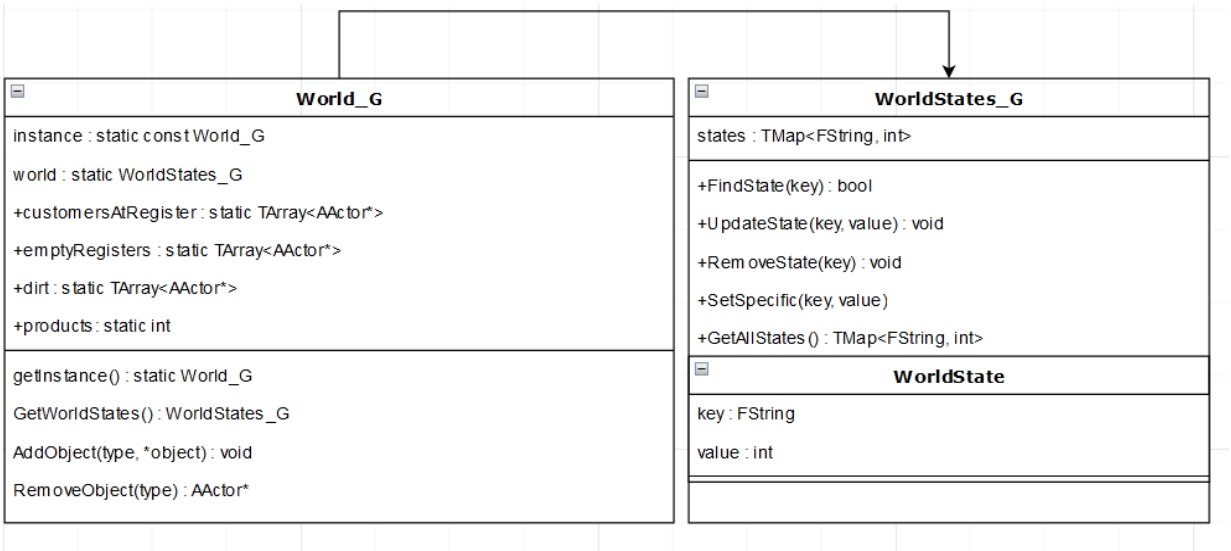
Suunnitelman valmistuttua hahmo alkaa suorittaa toimintoja suunnitelman mukaisessa järjestyksessä, mutta tekoälyn oikeanlaisen toiminnan varmistamiseksi hahmot varmistavat usein suunnitelman pitävyyden, sillä maailman tilanteisiin vaikuttaa myös hahmon ulkopuoliset tekijät. Tarvittaessa hahmo luo uuden suunnitelman, mikäli nykyinen suunnitelma ei ole enää mahdollinen, tai vaihtoehtoinen suunnitelma on parempi.

7 Tekninen toteutus

Tässä kappaleessa kuvataan tarkemmin, kuinka tavoitteellinen toiminnansuunnittelu on toteutettu tässä opinnäytetyössä. Tarkoituksena on tuoda esille huomattavasti teknisemmin koko toiminnallisuuden toteutumisketju.

7.1 Pelimaailma ja maailmantilat

Toiminnansuunnitteluun kuuluu olennaisena osana yksittäinen luokka, joka seuraa kaikkia suunnitteluun vaadittavia muuttujia pelimaailmassa. Projektissa tämä on staattinen World luokka ja sen toimintoihin kuuluu näiden muuttujien päivittäminen. World luokka pitää myös sisällään pelimaailman tiloja seuraavan staattisen maailmantila eli Worldstates luokan. Näiden luokkien staattisuus on tärkeää, koska maailmanmuuttujat ja -tilat ovat pelimaailmassa yksittäinen kokonaisuus.



Kuva 9. Maailma ja maailmantila luokkakaaviot

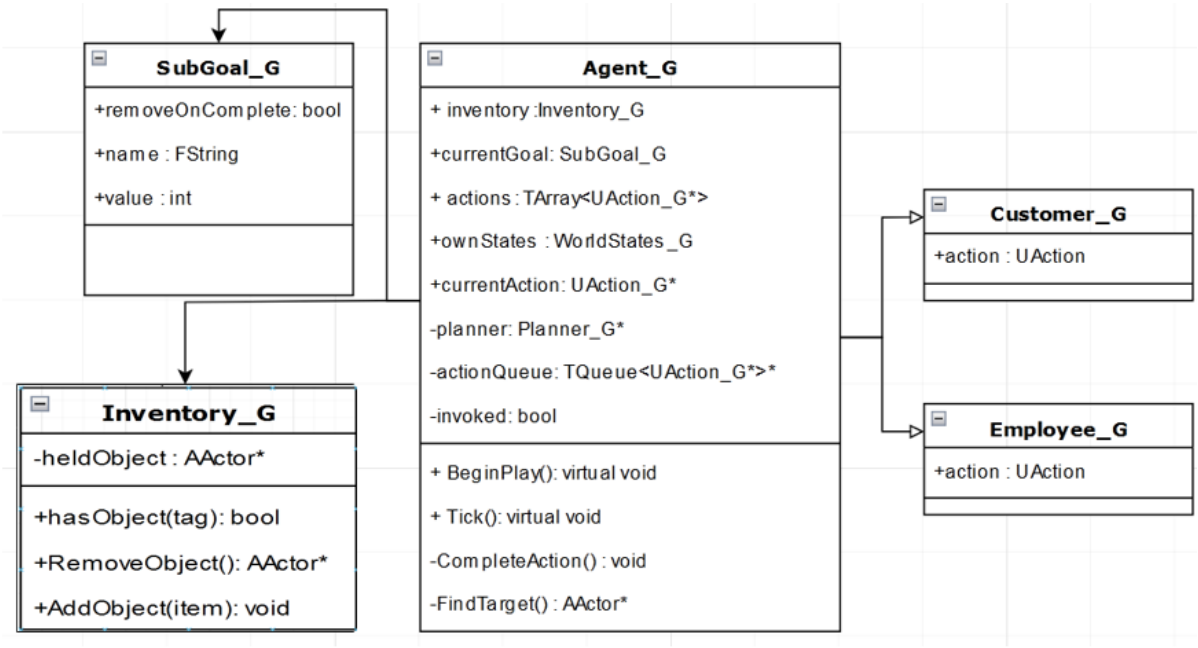
7.2 Agent suunnittelun aloittajana

Toiminnansuunnittelu lähtee liikkeelle pelissä olevasta tekoälyhahmosta, eli Agentista*. Agentin olomuodolle tai ominaisuuksille itsestään ei ole minkäänlaisia rajoituksia, mutta ainoa asia, jonka

hahmo tarvitsee, on tälle hahmolle käyvät toimet. Agent luokasta peritään molemmat pelin hahmot Employee ja Customer, tällöin tarkemmat määritykset ja käytössä olevat toiminnot voidaan määrätä tämän alaluokan mukaan. Nämä toiminnot, eli Actionit lisätään hahmoon erillisinä komponentteina, tässä tapauksessa apuna käyttäen Unreal Enginen omaa ”ActorComponent” järjestelmää.

Agent hahmolla on suunnittelun pohjana myös tavoite, lista omia ”tiloja” ja tavaraluettelo, jonka avulla agentti voi määrittää omistussuhteitaan, kuten esimerkiksi mikä kassa on käytössä, ja ketä asiakkaista palvellaan.

Kun Agent-hahmo luodaan pelimaailmaan, hahmo tarkistaa itseensä liitetyt actionit ja listaa ne toimintotaulukkoon myöhempää käyttöä varten.



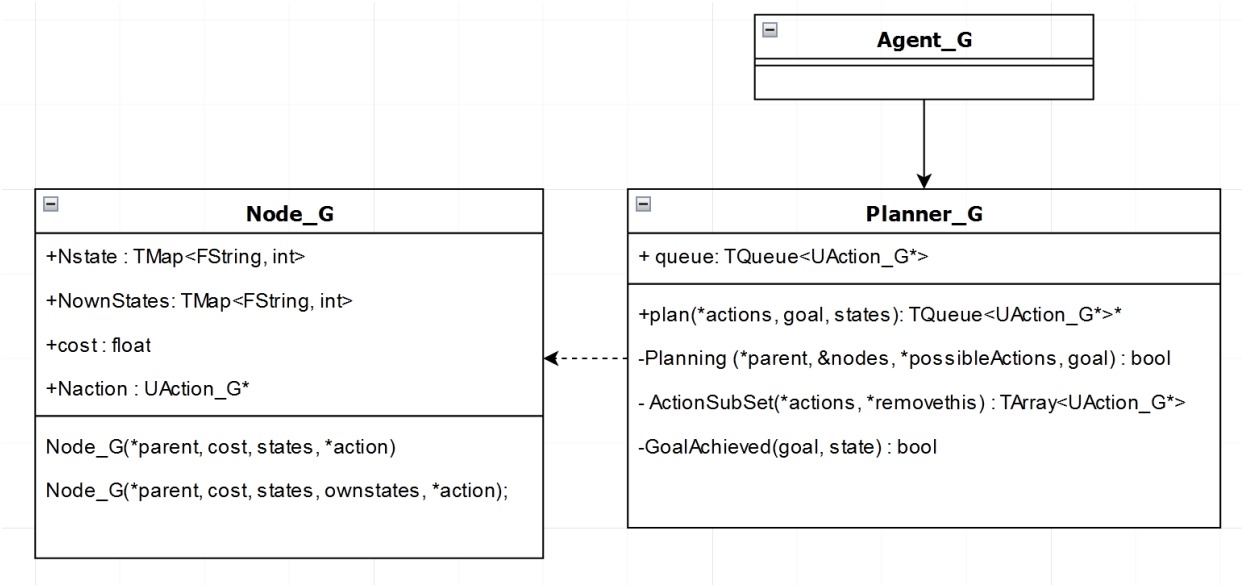
Kuva 10. Agent luokan luokkakaavio

Agentin ollessa olemassa pelimaailmassa, agentti jatkuvasti tarkistaa nykyistä toimintotilaansa. Tämä alkaa ensimmäisenä tarkastamalla, onko seuraavaa toimea määrätty. Jos toimi on jo määrätty, niin hahmo jatkaa toimen suorittamista. Huomioitavaa on, että määrättyä toimea ei aloitustilanteessa koskaan ole.

Jos toimea ei ole määrätty, on agentin tutkittava sille määrättyä toimintajonoa, mikäli tällainen on olemassa, agentti poimii jonosta seuraavan toimen. Mikäli toimintajonoa ei ole ollenkaan olemassa, agentti luo itselleen uuden suunnittelijakomponentin, eli plannerin.

7.3 Suunnittelija

Agentin luotua itselleen suunnittelijakomponentin, kutsuu se heti suunnittelijalta toimintajonoa. Tähän toimintajonon muodostamiseen suunnittelija tarvitsee agentin omistamien actioneiden muodostaman toimintataulukon, agentin oman tilanteen, ja agentin tavoitteen.



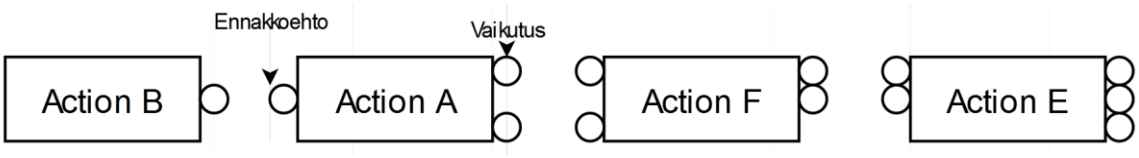
Kuva 11. Suunnittelija ja Node luokkakaaviot

Tämän toimintajonon muodostamiseksi suunnittelijan, eli plannerin täytyy tarkistaa toimintataulukon toimet tälle agentille mahdollisiksi. Suunnittelija luo tämän jälkeen tyhjän solmutaulukon, sekä luo suunnitelman aloitussolmun, eli noodin, antaen tälle aloitussolmulle tämänhetkisen maailmantilan, aiemmin tuodun agentin oman tilanteen sekä agentin määrittelemän tavoitetilan.

Aloitusnoodista suunnittelija aloittaa rakentamaan mahdollista polkua agentin tavoitteeseen, käyttämällä tarkoitukseen muokattua A* algoritmia. Tämä tapahtuu vertaamalla nykyistä maailmantilaa *jokaiseen* agentin actioniin. Mikäli toimen ennakkoehdot ovat toteutuneet, eli action on mahdollinen suorittaa, luodaan *jokaisen* actionin vaikutuksesta uusi teoreettinen ”tämänhetkinen” maailmantila, joka kirjaa actionin aiheuttamat maailmantila muutokset vain itseensä.

Tätä teoreettista maailmantilaa verrataan agentin tavoitemaailmantilaan, jos tämä maailmantilat vastaavat toisiaan, niin voidaan todeta, että edeltävät toimet ovat aiheuttaneet halutun lopputuloksen.

Haluttuun maailmantilaan ei kuitenkaan usein päästä vain yhdellä toimella, joten tällaisessa tilanteessa *jokaisesta* actionista luodaan uusi mahdollisten toimien toimintotaulukko, josta poistetaan jo suoritettu toimi. *Jokaisesta* actionista luodaan tällöin uusi noodi ja teoreettiset toimet ovat voineet toteuttaa uusien toimien ennakkoehtoja. Nämä noodit kantavat mukanaan yhteyden edelliseen noodin, sekä kaikkien toimien yhteishintaa ja omaa teoreettista maailmantilannetta.



Kuva 12. Noodien muodostamat mahdolliset toimijonot

Tämän jälkeen jokaisesta solmusta jatketaan samalla tavalla; käyttämällä uusia toimintotaulukkoja, teoreettisia maailmantiloja, alkuperäistä agentin tavoitetta uutena pohjana, ja laskien samalla toimien yhteishintaa. Toteutus jatkuu niin kauan, että kaikki mahdolliset toimet on käyty läpi. Jos tavoite täyttyy, sen täyttäneet noodit tallennetaan.

Kuva 13. Suunnitelmia punaisella, ja tehtävien toteutumisia sinisellä

Kun kaikki mahdolliset tavoitetilat on löydetty, palautaan lopputulos takaisin toimijonon suunnitteluun, ja aloitetaan näiden noodien vertailu keskenään. Jokainen noodi on kerännyt vaadittujen actioneiden hintojen yhteissumman, joten näistä matalin arvo on paras vaihtoehto.

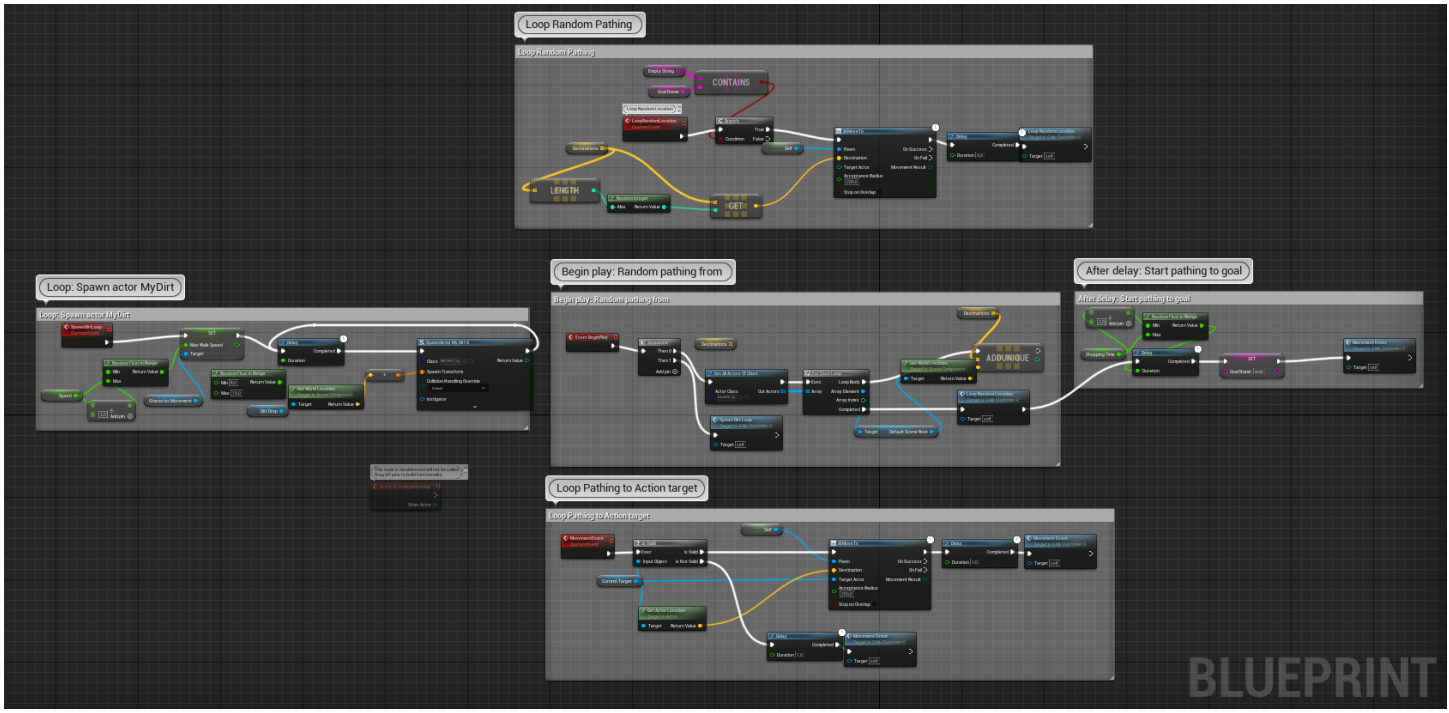
Tämän jälkeen halvin noodi laittaa sisältämänsä toimen toimijonoon ja korvaa itsensä edellisellä noodilla, toistaen tätä niin kauan kuin noodeja on. Muodostaen näin toimijonon, joka saavuttaa lopulta agentin tavoitetilan.

Kun toimintajono on valmis, palautuu se agentille, jonka jälkeen agentti aloittaa toimintajonon ensimmäisen toimen suorittamisen.

7.4 Actionin suoritus

Actionin suoritusta ei voida aloittaa, ennen kuin sen ennakkoehdot ovat tarkastettu. Jokaisella actionilla on kahdentyyppisiä ennakkoehtoja; ehdot agentille ja ehdot maailmantilalle. Agentti on täyttänyt ehdot suorittamiselle suunnitteluvaiheessa, mutta maailmantilalle määritetyt ehdot voivat estää suunnitelman toteuttamisen, jolloin agentti määrittelee uuden suunnitelman. Maailmantila ehdot varmistavat, että toimen vaatimat ulkopuoliset asiat ovat olemassa.

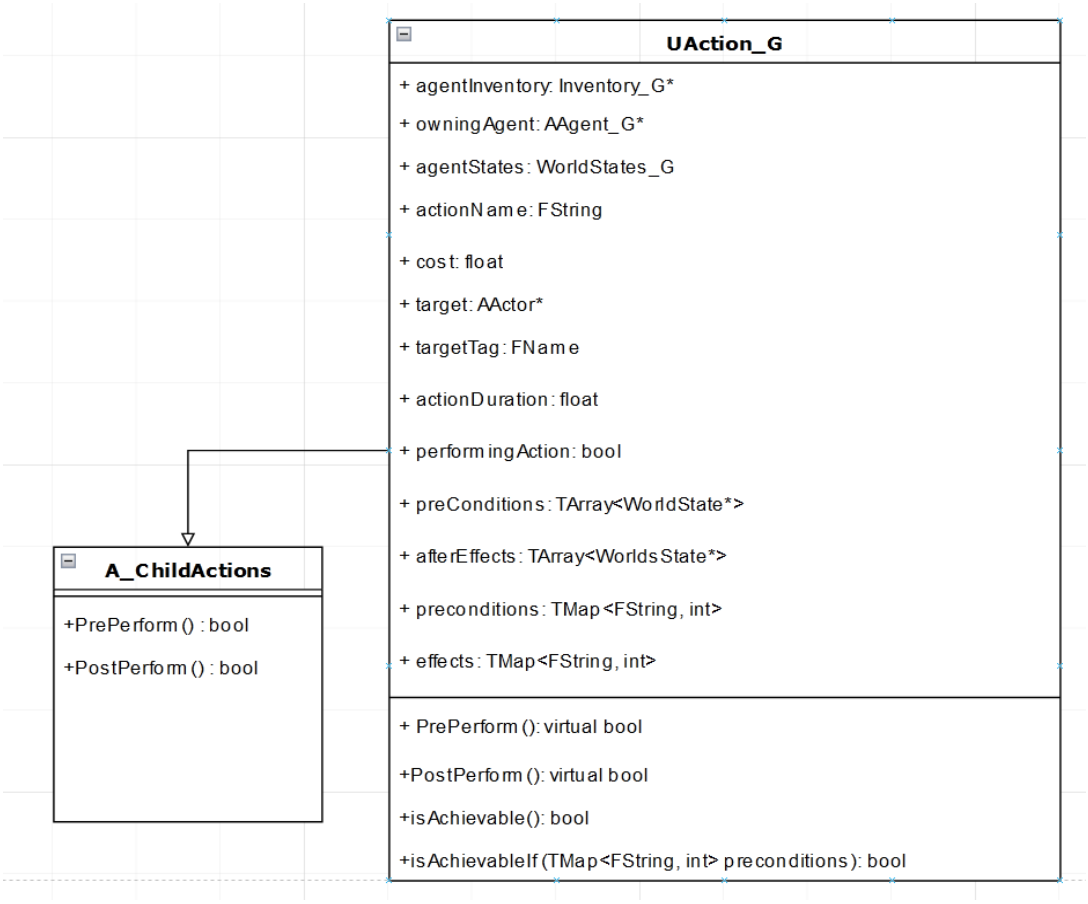
Jos actionin suorittaminen on mahdollista, on agentin ensin löydettävä actionin kohde maailmasta, ja tähän apuna on käytössä Unreal Enginen tag-merkintä. Pelimoottori etsii merkittyä esinettä pelimaailmasta, ja asettaa sen agentin kohteeksi. Tämä kohde paljastetaan hahmon blueprintille, josta yksinkertainen liikkumislogiikka ohjaa agentin kohteen luo käyttäen pelimoottorin omaa polunetsintää.



Kuva 14. Asiakasluokan blueprint, joka ohjaa polunetsintää

Actionionin suorittaminen projektissa on yksinkertaistettu; hahmo odottaa kohteen luona actionin määrittelemän ajan, jonka jälkeen action on suoritettu ja sen määritetyt jälkivaikutukset tapahtuvat. Kuten ennakkoehdoissa, näitä jälkivaikutuksia on kahdenlaisia; vaikutukset agenttiin ja vaikutukset maailmantilaan.

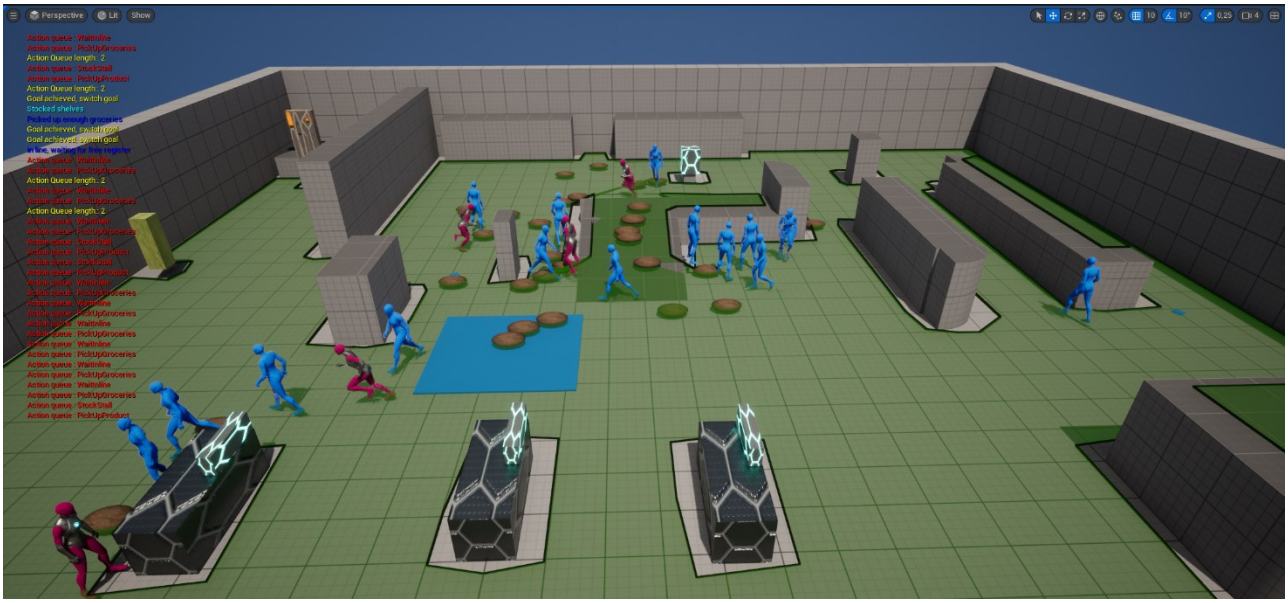
Agenttiin kohdistuvat vaikutukset täyttävät usein ennakkoehdot seuraavalle toimelle. Maailmantilaan kohdistuvat vaikutukset taas päivittävät pelimaailmantila muuttujia tulevia suunnitelmia varten.



Kuva 15. Action luokka

7.5 Actioneiden eri tyypit

Projektissa actioneiden ennakkoehtojen toteutumiselle muodostui useampia erilaisia toteutumistapoja. Sillä osa actioneista ennakkoehdoista vaati maailmantilojen tai muiden agenttien osallistumisen, jolloin pelkkä omien toimien vaikutusten huomioonottaminen ei riittänyt. Tässä kappaleessa käyn läpi erilaiset ennakkoehtojen toteutustavat.



Kuva 16. AI agentteja eri toimissa

Yksinkertaisin kaikista toimista oli siivous, joka vaati agentilta omistuksesta siivousvälineet, ja agentin kerättyä siivousvälineet, agentti pystyy siivoamaan maassa olevia likoja jatkuvasti, ilman muita vaatimuksia tälle toimelle. Suunnittelija ottaa huomioon, ettei tavaraluettelossa olevat siivousvälineet katoa, ellei agentti vaihda toiseen toimeen, joka korvaisi välineet.

Toimena hyllyjentäyttö on myös yksinkertainen, mutta eroaa siivouksesta, koska agentti joutuu hakemaan lisää tuotteita. Tämän vuoksi ennakkoehtoon ei tule muutosta, vaikka toimi toteutettaisiin uudestaan.

Kassatyöskentely oli edeltäviin toimiin verrattuna huomattavasti monimutkaisempi, ja tämän vuoksi käytti myös hyväkseen useampia projektissa olevia resursseja. Kassatyöskentely vaati maailmantilojen seuraamista, sillä myyjän täytyi löytää itselleen tyhjä kassa, ja kassalla ollessaan maailmantilojen avulla löydetään myös jonottavat asiakkaat.

Hahmojen täytyi myös toimia yhdessä, jotta asiakaspalvelu saatiin suoritettua, tämän apuna käytettiin tavaraluetteloa. Kun myyjähahmo havaitsee, että maailmantila jonottavista asiakkaista päivittyy, valitsee agentti seuraavan näistä asiakkaista, ja siirtää tämän ”omistukseen” tavaraluetteloon oman kassansa. Tätä käytetään asiakkaan maksutoimen ennakkoehtona, joten tämän jälkeen asiakas valitsee kohteeksensa kassan, ja tapahtuman suorituksen myötä poistuu kokonaan. Tämä tapahtuma toteuttaa molempien osapuolien tavoitteen.

7.6 Tavoitteiden päivittyminen

Hahmot suorittavat omia tavoitteitaan erittäin nopeasti, ja tämän vuoksi projektissa tavoitteet päivittyvät aina kun edellinen tavoite tulee suoritetuksi. Kun hahmo suorittaa oman tavoitteensa, tutustuu hahmo eri maailmantilan muuttujiin, ja jos yksi tai useampi muuttujista on kriittisessä tilassa, vaihtaa hahmo oman tavoitteen sellaiseksi, joka edistää tämän muuttujan tilannetta. Muussa tapauksessa hahmon tavoite pysyy samana, ja suunnittelija luo uuden suunnitelman.

Kuva 17. Kriittinen siivoustilanne

8 Yhteenveto ja teorian toteutus kokemuksena

Projektin käytännöntoteutus oli hyvin monipuolinen kokemus, ja törmäsin itselleni täysiin uusin ongelmiin ja löysin näiden myötä ratkaisujen lisäksi myös uusia parempia toimintatapoja ja ratkaisuja.

Projektin lopputuloksesta itselläni on hieman ristiriitaiset tunteet. Jos alkuperäistä suunnitelmaa vertaa projektin lopputulokseen, on helposti huomattavissa, että osa toiminnoista jäi toteuttamatta. Tähän päällimmäisenä syynä olivat virheet alkuperäisissä suunnitelmissa.

Suurimpana pettymyksenä oli tekniikan rajoittuvuus maalitilan määrittämisessä. Toteutuksessa oli mahdotonta määrittää maalitilaksi alun perin suunniteltua myymälän tuoton maksimointia, sillä jokaisen agentin olisi täytynyt jatkuvasti seurata kaikkia näitä muuttujia ja niiden vaikutuksia, eikä tämä ollut teknisesti järkevä toteuttaa. Toteutukseen olisi mielestäni soveltunut paremmin toinen AI tekniikka, jossa agenttien yksilöllisten suunnittelijoiden lisäksi olisi ollut kokonaisvaltainen suunnittelija, ja tämä toteutus olisi tällöin mennyt opinnäytetyön aiheen ulkopuolelle.

Projektin rakentamisessa ongelmia tuli ylivoimaisesti eniten Unreal enginestä, ja osaan näistä en löytänyt selkeää syytä tai ratkaisua. Yksittäisiä pieniä ongelmia oli kymmenittäin, ja usein oikea ratkaisu oli yksinkertaisesti tehdä asia eri tavalla, vain koska Unreal ei tukenutkaan tiettyä toimintatapaa kyseisessä tilanteessa tai kontekstissa.

Tavoitteellisen tehtävänsuunnittelun rakentamisessa oli itsessään paljon haasteita. Ensimmäisenä haasteena oli se, ettei tekniikka pystynyt helposti testaamaan, ennen kuin se on vähintään osittain valmis. Tällaiseen sokkona työskentelyyn en ollut varautunut, sillä virheiden ratkominen niiden muodostuessa, on olennainen osa oppimista. Vasta kun logiikka alkoi olla suoritettavalla tasolla, muodostuikin ongelmia koodista, joka oli tuotettu jo alkuvaiheessa, ja näiden ongelmien havaitsemissa kesti normaalia pidempi aika. Toisena erittäin huomattavana haasteena oli suunnittelijan ongelmien ratkaisu, kun ongelmat kohdistuivat suunnitelmien rakentamiseen, sillä kaikkien mahdollisten toimintajonojen seasta oli joskus työlästä löytää yksittäinen ongelmakohta.

Lopputuloksena projektista tuli kuitenkin täysin toimiva tavoitteellinen tehtäväsuunnittelija, jonka kaikki teoria ja käyttötavat toimivat oikein, eikä ongelmia esimerkiksi suorituskykyyn liittyen ollut havaittavissa. Tästä voin olla tyytyväinen, ja tämän vuoksi projektista jäikin ristiriitaiset tunteet. Sillä vaikka tavoitteellinen tehtäväsuunnittelija toimii ongelmitta, on projektin asetelma tek-

niikalle liian yksinkertainen. Oppikokemuksena tästä on otettava se, että tavoitteellisen tehtäväsuunnittelijan käyttötarkoitukseen on löydettävä tarve erittäin monimutkaiselle tekoälylle, sillä vaikka tavoitteita, tilanteita ja tehtäviä oli erilaisia, niin heikkoudeksi osoittautui se, ettei näiden tavoitteiden suoritustavoissa ollut ollenkaan variaatiota. Parhaan suunnitelman löytämisen tekniikka toimii, mutta jos paras suunnitelma on aina samanlainen, on projekti tällöin huono esittelemään tekniikkaa kokonaisuudessaan, joka jäi harmittamaan. On otettava kuitenkin myös huomioon, että vielä monimutkaisen projektin rakentaminen tekniikan ympärille olisi vaatinut huomattavasti aikaa.

Opinnäytetyön jälkeen toiveena olisi tehdä tavoitteellisesta tehtäväsuunnittelijasta mahdollisesti kokonaan itsenäinen Unreal enginen plugin, minkä jälkeen olisi mahdollista nähdä tekniikka käytössä monimutkaisemmassa ympäristössä.

Lähteet

Buckland, M. (2005). *Programming game AI by example (TKÄS-koulutus, 2018)*. Wordware Pub.

Epic Games. (2022). Artificial Intelligence in Unreal Engine. Unreal Engine. Viitattu 8.9.2022 Saatavilla https://docs.unrealengine.com/5.0/en-US/artificial-intelligence-in-unreal-engine/

Epic Games. (2022). Adding Components to an Actor. Unreal Engine. Viitattu 18.11.2022 Saatavilla https://docs.unrealengine.com/5.0/en-US/adding-components-to-an-actor-in-unreal-engine/

Epic Games. (2022). Unreal Engine 5 is now available! Viitattu 20.9.2022 Saatavilla https://www.unrealengine.com/en-US/blog/unreal-engine-5-is-now-available

Happy Coding (n.d). Pathfinding. Saatavilla 02.11.2022 https://happycoding.io/tutorials/libgdx/pathfinding

IGN. (2009). Epic Games Announces Unreal Development Kit, Powered by Unreal Engine 3. Viitattu 20.9.2022 Saatavilla https://www.ign.com/articles/2009/11/05/epic-games-announces-unreal-development-kit-powered-by-unreal-engine-3

Long, E. (2007). Enhanced NPC behaviour using goal-oriented action planning. *Master's Thesis, School of Computing and Advanced Technologies, University of Abertay Dundee, Dundee, UK*. Viitattu 8.9.2022 Saatavilla https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.131.8964&rep=rep1&type=pdf

Orkin, J. (2004). Symbolic representation of game world state: Toward real-time planning in games. In *Proceedings of the AAAI Workshop on Challenges in Game Artificial Intelligence* (Vol. 5, pp. 26-30). Viitattu 8.9.2022 Saatavilla https://www.aaai.org/Papers/Workshops/2004/WS-04-04/WS04-04-006.pdf

Orkin, J. (2006). Three States and a Plan: The A.I of F.E.A.R. Viitattu 8.9.2022 Saatavilla http://alumni.media.mit.edu/~jorkin/gdc2006_orkin_jeff_fear.pdf

Rabin, S. (2014). *Game AI pro: Collected wisdom of game AI professionals (TKÄS-koulutus, 2018)*. CRC Press, Taylor & Francis Group

Rasmussen, J. (2016). Are Behavior Trees a Thing of the Past? Gamedeveloper Blog. Viitattu 25.10.2022 Saatavilla https://www.gamedeveloper.com/programming/are-behavior-trees-a-thing-of-the-past-