Videopelitekoälyn toteutus Goal-Oriented Action Planning -menetelmällä

HAMK
HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Ammattikorkeakoulututkinnon opinnäytetyö

Tietojenkäsittelyn koulutus
kevät, 2023

Sami Lappalainen

HAMK
HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Tietojenkäsittelyn koulutus Tiivistelmä

Tekijä Sami Lappalainen Vuosi 2023

Työn nimi Videopelitekoälyn toteutus Goal-Oriented Action Planning -menetelmällä

Ohjaaja Tommi Lahti

TIIVISTELMÄ

Videopelit luottavat usein tekoälyjärjestelmiin mukaansatempaavien ja haastavien pelikokemusten luomiseksi. Tämä opinnäytetyö tutkii videopelitekoälyn toteutusta Goal-Oriented Action Planning (GOAP) -tekniikalla. GOAP on laajalti käytetty menetelmä, jonka avulla agentit voivat dynaamisesti suunnitella ja toteuttaa toimia tietyn tavoitteen saavuttamiseksi.

Opinnäytetyö antaa yleiskatsauksen suosituimmista videopelitekoälytekniikoista, jonka jälkeen se tutkii tarkemmin GOAP-menetelmää, mukaan lukien sen suunnittelu- ja toteutusvaiheet.

Tuloksena syntyi kaksi simulaatiota, joissa tekoälyn avulla toimivat agentit suorittavat toimintosarjoja, pohjautuen GOAP-menetelmään. Simulaatioiden tarkoituksena oli tutkia, kuinka GOAP soveltuu erilaisiin peliprojekteihin ja mitä hyötyjä tai haasteita sen käytössä on.

Johtopäätöksenä voitiin todeta, että GOAP soveltuu parhaiten peleihin, joissa halutaan luoda monimutkaista käytöstä tekoälyn avulla. Yksinkertaisissa projekteissa se puolestaan voi helposti lisätä työnmäärää ilman, että se tarjoaa huomattavaa hyötyä pelinkehittäjälle. Tällaisissa tapauksissa on siis suotavampaa käyttää muita tekoälyvaihtoehtoja.

Avainsanat Tekoäly, Videopelit, GOAP

Sivut 25 sivua ja liiteitä 1 sivu

HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Degree Programme in Business Information Technology Abstract

Author Sami Lappalainen Year 2023

Subject Implementing Video Game AI using Goal-Oriented Action Planning

Supervisor Tommi Lahti

ABSTRACT

Video games often rely on artificial intelligence to create immersive and challenging gaming experiences. This thesis examines the implementation of artificial intelligence in a video game environment using the Goal-Oriented Action Planning (GOAP) technique. GOAP is a widely used method that allows agents to dynamically plan and execute actions to achieve a given goal.

This thesis provides an overview of the most popular AI techniques used in video games, and then explores the GOAP method in more detail, including its design and implementation phases.

As a result, two simulations were created, in which agents, acting with the help of artificial intelligence, perform sequences of actions based on the GOAP method. The purpose of the simulations was to investigate how suitable GOAP is for different game development projects, and what benefits or challenges arise from its use.

As a conclusion, it could be stated that GOAP is best suited for games where you want to create complex behaviour with the use of artificial intelligence. In simple projects, however, it can easily increase the amount of work without providing a significant benefit to the game developer. In such cases, it is therefore preferable to use other options in the implementation of artificial intelligence.

Keywords Artificial intelligence, Video games, GOAP

Pages 25 pages and appendices 1 page

Sanasto

Unity	Pelimoottori
NPC	Non-Playable Character, eli ei-pelaajahahmo, tekoälyn kontrolloima pelihahmo
Tekoäly	Tietokoneiden kyky tehdä älykkäinä pidettäviä toimintoja
C#	Ohjelmointikieli
GOAP	Goal-Oriented Action Planning, tekoälymenetelmä
FSM	Finite state machine, eli äärellinen automaatti, laskennallinen malli
Käytöspuu	Behaviour Tree, mallinnuskieli ja tekoälytekniikka
A*	A-star, tai A-tähti, polunetsintäalgoritmi
Agentti	Hahmot, jotka toimivat tekoälyn avulla
Polunetsintä	Mahdollisimman hyvän reitin löytäminen pisteestä toiseen, yleensä algoritmin avulla

Sisällys

Kuvat, ohjelmakoodit ja taulukot

1 Johdanto

Videopelit ovat kehittyneet paljon vuosien saatossa niin graafisesti, kuin myös pelimekaanikoiltaan. Yksi alue, joka on myös kasvanut merkittävästi, on tekoäly peleissä. Kyky luoda uskottavia ei-pelaajahahmoja (NPC), jotka voivat ajatella ja toimia itsenäisesti, voi mullistaa pelikokemuksen. Yksi tekoälytekniikka, joka on ollut pelinkehittäjien suosiossa, on Goal-Oriented Action Planning (GOAP), eli tavoitteellinen toimintasuunnittelu.

GOAP on tekniikka, jonka avulla ei-pelaajahahmot voivat tehdä päätöksiä ja toimia nykyiseen tilanteisiinsa ja tavoitteisiinsa perustuen. Siten tekoälyhahmot voivat reagoida ympäröivään maailmaan realistisella ja uskottavalla tavalla.

Tässä opinnäytetyössä tutkin GOAP-menetelmän käyttöä suositussa pelikehitysmoottorissa Unityssa. Tutkin, kuinka GOAP voidaan ottaa käyttöön Unityssa ja miten sen avulla voidaan luoda kiinnostavampia ja interaktiivisempia tekoälyhahmoja. Tutkin myös tämän menetelmän mahdollisia rajoituksia ja haasteita. Tämän opinnäytetyön loppuun mennessä lukijan pitäisi ymmärtää paremmin mitä on tekoäly videopeleissä ja kuinka tavoitteellista toimintasuunnittelua voidaan käyttää uskottavan tekoälyn kehittämiseen.

Alla on listattu tämän opinnäytetyön tutkimuskysymykset.

- Mitä on tekoäly videopeleissä?
- Miten Goal-Oriented Action Planning toimii?
- Miten GOAP sovelletaan omaan peliprojektiin?
- Mitkä ovat GOAP:n käytön hyödyt ja haasteet?

2 Tekoäly videopeleissä

Tekoäly videopeleissä viittaa tietokonealgoritmien ja laskennallisten mallien käyttöön älykkäiden, sekä interaktiivisten ei-pelaajahahmojen luomisessa. Tekoälyn avulla voidaan parantaa pelaajakokemusta esimerkiksi luomalla realistisia ja haastavia vastustajahahmoja, jotka voivat toimia ja tehdä päätöksiä perustuen strategiseen ajatteluun. (Millington & Funge, 2009)

Toinen tärkeä tekoälyn käyttötapa videopeleissä on tarjota dynaamisia haasteita ja lisätä pelien uudelleenpelattavuutta. Tekoälyalgoritmeilla voidaan luoda uutta pelisisältöä, kuten tasoja, vihollisia ja esteitä, jotka tarjoavat uusia kokemuksia aina, kun pelaaja pelaa peliä. Tämä pitää pelaajat uppoutuneena pelimaailmaan kauemmin. (Millington & Funge, 2009)

Jo vuonna 1979 julkaistu Pac-Man, sisälsi yksinkertaisia tekoälytekniikoita sen sisältämien haamuhahmojen käyttäytymisen toteuttamisessa. Siitä lähtien tekoälyn käyttö videopeleissä on muuttunut yhä monipuolisemmiksi, ja eri peligenret käyttävät erilaisia tekoälytekniikoita erilaisten tavoitteiden saavuttamiseksi. Esimerkiksi ajo- tai urheilupeleissä saatetaan haluta laskea nopein tapa kiertää kilparata, kun taas roolipeleissä halutaan hallita monimutkaisia vuorovaikutuksia eri hahmojen väillä. Käytetyt tekoälytekniikat vaihtelevat laajalti edelleen käytetystä tilapohjaisesta tekoälystä, monimutkaisempiin tekniikoihin, kuten neuroverkkoihin. (Millington & Funge, 2009)

2.1 Uskottava tekoäly

Videopelitekoälyn pääasiallinen tehtävä on vaikuttaa uskottavalta pelaajalle, tehden pelistä entistä mukaansatempaavamman. Tämän tehtävän suorittamista varten ei siis välttämättä tarvita monimutkaista tekoälyä, vaan sen sijaan tarkoitus on luoda illuusio älykkyydestä yhdistämällä oikeanlaiset käyttäytymiset oikeanlaisten algoritmien kanssa. (Millington & Funge, 2009)

Hyvä esimerkki tästä on aiemmin mainittu Pac-Man, jossa tekoäly ohjaa neljää haamuhahmoa, jotka liikkuvat pelaajaa kohti eri tavoilla. Normaalitilassaan ne liikkuvat suorassa linjassa risteykseen asti ja valitsevat sitten seuraavan reittinsä, joko kohti pelaajaa tai satunnaiseen suuntaan. Jokaisella haamulla on eri todennäköisyys valita näiden vaihtoehtojen väillä, mikä luo jokaiselle haamulle ainutlaatuisen käyttäytymisen. Niiden puolisatunnaisten liikkeidensä yhdistelmä luo haastavan ja kiinnostavan pelikokemuksen, joka on saanut monet uskomaan, että haamut työskentelevät yhdessä asettaakseen ansoja ja väijytyksiä pelaajaa vastaan. Tekoälyn yksinkertaisuudesta huolimatta se on jättänyt pelaajiin pysyvän vaikutuksen. (Millington & Funge, 2009)

Huonosti toteutettu tekoäly voi puolestaan pilata pelikokemuksen. Esimerkiksi huonosti implementoitu pelihahmojen polunetsintä tai liian älykkäät viholliset voivat tehdä pelikokemuksesta turhauttavan. Tilanteet, joissa pelaaja löytää mahdollisuuden manipuloida tekoälyä, voi myös rikkoa halutun illuusion pelihahmojen älykkyydestä. (Millington & Funge, 2009)

Yksi haasteista uskottavuuden luonnissa on löytää tasapaino monimutkaisen ja yksinkertaisen toteutuksen väiltä. Liian monimutkainen tekoäly johtaa helposti virheisiin, joka saa hahmot näyttämään tyhmiltä. Liian yksinkertainen toteutus taas voi olla liian ennalta ennustettava pelaajille. Avain uskottavan tekoälyn luomiseen on käyttää yksinkertaisia tekniikoita älykkäillä tavoilla luodakseen illuusion monimutkaisesta älykkyydestä. (Millington & Funge, 2009)

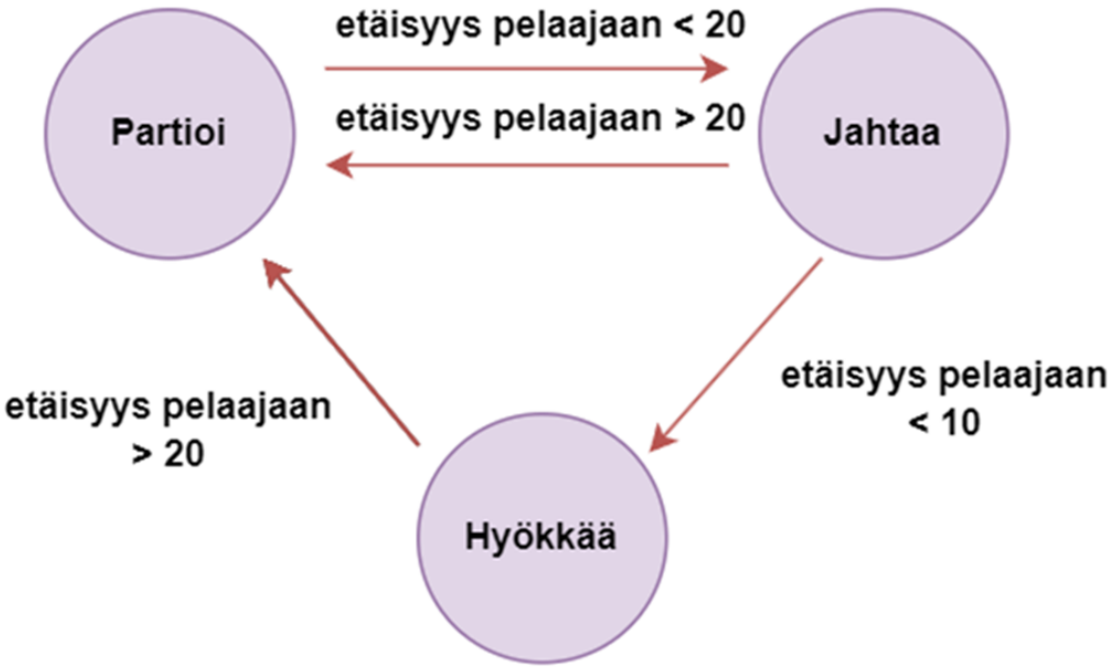
2.2 Muut yleiset menetelmät

Videopelit käyttävät tarpeiden mukaan tekoälyn luonnissa monia erilaisia menetelmiä, kuten sääntöpohjaisia järjestelmiä, äärellisiä tilakoneita, päätöspuita, käytöspuita, neuroverkkoja, vahvistusoppimista ja evoluutioalgoritmeja. Tässä työssä mainitaan näistä kolme; tilakoneet, käytöspuut, sekä GOAP-menetelmän.

2.2.1 Äärellinen automaatti (FSM)

Äärellinen automaatti, tai Finite State Machine (FSM), on laskennallinen malli, jonka avulla voidaan määrittää olion tai järjestelmän tila. Tila puolestaan määrää kuinka olio tai järjestelmä käyttäytyy. Esimerkiksi tekoälyhahmo, joka on tarpeeksi lähellä pelaajahahmoa, saattaa lähteä jahtaamaan sitä, kuten on havainnollistettu kuvassa 1. (iHeartGameDev, 2021)

Kuva 1. Esimerkki äärellisestä automaatista



Tilamalli on tapa järjestää ja toteuttaa nämä eri tilat koodin sisällä, mikä mahdollistaa halutun käyttäytymisen liittämisen kuhunkin tilaan, vaikuttamatta muihin tiloihin. Tämä malli voidaan toteuttaa käyttämällä äärellistä automaattia, jonka avulla voidaan määrittää haluttu määrä tiloja. Äärellinen automaatti voi olla yhdessä tilassa kerrallaan ja siirtyy tilasta toiseen määriteltyjen ehtojen mukaan. Äärellisen automaatin luomiseksi on siis määriteltävä jokainen haluttu tila, niiden väliset siirtymät ja alkutila. (iHeartGameDev, 2021)

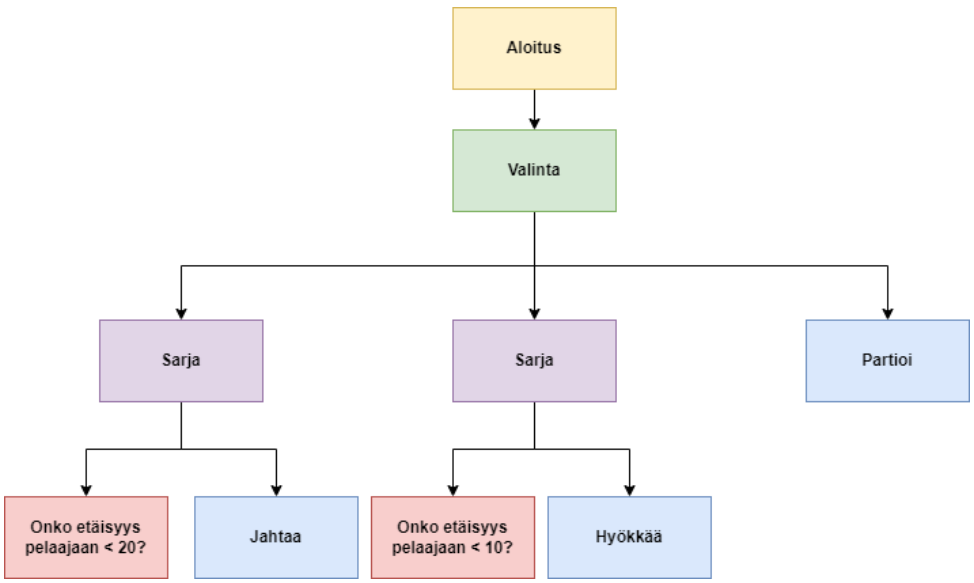
2.2.2 Käytöspuut

Käytöspuu on eräänlainen matemaattinen malli, jota käytetään videopeleissä tekoälyhahmojen käyttäytymisen määrittämiseen. Se koostuu sarjasta solmuja, jotka alkavat yläosassa olevasta juurisolmusta yhteen tai useampaan solmuun, joissa tietty toiminta suoritetaan. Käytöspuun logiikka kulkee puun ylhäältä alas ennalta määritettyyn suuntaan. (Simpson, 2014)

Käytöspuussa olevilla solmuilla on neljä päätyyppiä: juurisolmut (root), lehtisolmut (leaf), yhdistelmäsolmut (composite) ja koristesolmut (decorator). Solmu, joka on linkitetty tämän alapuolella olevaan solmuun, kutsutaan alemman solmun vanhemmaksi. Samoin kaikkia solmuja, joihin se on linkitetty sen alapuolella, kutsutaan sen lapsiksi. (Simpson, 2014)

Juurisolmu on puun aloituspiste. Sillä ei ole vanhempia ja muut solmut ovat sen lapsia. Lehtisolmut sisältävät todellisen tekoälykäyttäytymisen, kuten juoksemisen tiettyyn paikkaan tai vihollisen ampumisen. Yhdistelmäsolmuja on kahdenlaisia ja ne ohjaavat puun kulkua. Valintasolmut (selector) päättävät, mikä lapsisolmu suoritetaan pelimaailman logiikan perusteella, ja sarjasolmut (sequence) jonka avulla voidaan suorittaa useita lapsisolmuja peräkkäin. Koristesolmut puolestaan muokkaavat lapsisolmun käyttäytymistä, esimerkiksi toistamalla toiminnan tietyn määrän kertoja. Puun rakennetta kuvataan kuvassa 2. (Simpson, 2014)

Kuva 2. Esimerkki käytöspuusta



Käytöspuut ovat erittäin suosittu valinta videopelien kehityksessä. Jakamalla tekoälykäyttäytymisen yksittäisiin solmuihin, ne tarjoavat modulaarisen lähestymistavan tekoälylogiikan hallintaan ja muokkaamiseen. Tämä mahdollistaa myös solmujen uudelleenkäytön eri hahmojen tai tilanteiden välillä, mikä vähentää kirjoitettavan koodin määrää. Puurakenteen ansiosta kehittäjien on myös helpompi visualisoida ja ymmärtää tekoälyn käyttäytymisen kulkua. Lisäksi puuta voidaan helposti laajentaa tai muuttaa vastaamaan uusia käyttäytymismalleja tai muutoksia pelimaailmassa. (Isla, 2005)

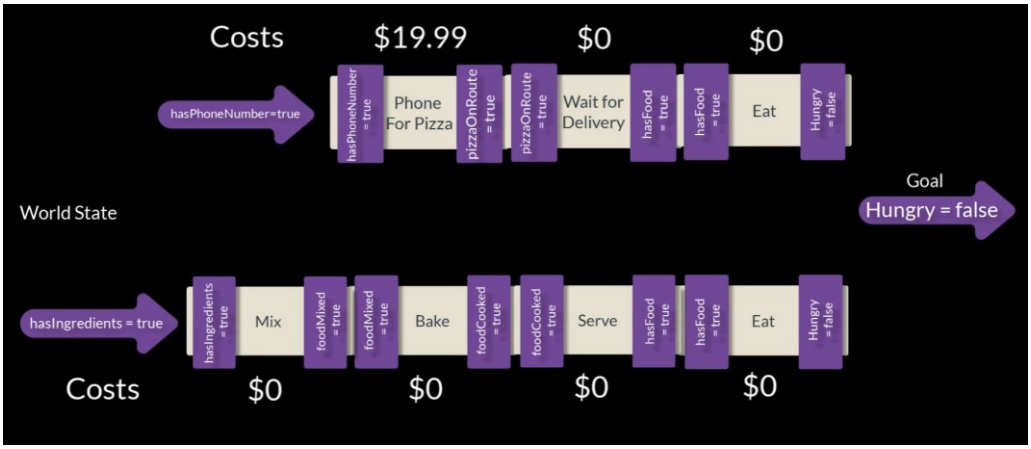
3 GOAP

Goal-Oriented Action Planning (GOAP), tai tavoitteellinen toimintasuunnittelu, on erityisesti itsenäisten hahmojen reaaliaikaista käyttäytymistä ohjaamaan suunniteltu suunnitteluarkkitehtuuri. Sen loi alun perin Jeff Orkin työskennellessään Monolith Productionssissa ja sitä käytettiin ensimmäistä kertaa F.E.A.R. -pelissä vuonna 2005. Tämän jälkeen menetelmää on käytetty monissa muissa suosituissa videopeleissä kuten F.E.A.R 2, Fallout 3 ja Deus Ex: Human Revolution. Menetelmää on käytetty myös muihin tarkoituksiin kuin pelikehitykseen, esimerkiksi robotiikassa ja sotilasteknologiassa. (Orkin, *Goal-Oriented Action Planning (GOAP), n.d.*; Doris & Silvia, 2007)

Tavoitteellisen toimintasuunnittelun avulla itsenäiset agentit (kuten pelihahmot) suunnittelevat ja toteuttavat sarjan toimintoja tietyn tavoitteen saavuttamiseksi. Toimintosarjan valintaan vaikuttaa agentin ja ympäristön nykyinen tila, mikä tarkoittaa, että kaksi agenttia voivat toimia eri tavoilla saman tavoitteen saavuttamiseksi. (Chaduhari, 2017)

Esimerkiksi agentilla voi olla nälkä, jolloin tämän tavoite on syödä pizza. Agentti voi saavuttaa tämän tavoitteen useilla tavoilla, kuten tilaamalla pizzan tai valmistamalla sen itse. Agentin mahdollisuus valmistaa pizza riippuu siitä, onko tällä raaka-aineita. Tilaaminen taas riippuu siitä, onko agentin mahdollista käyttää puhelinta. GOAP valitsee tehokkaimman toimintosarjan agentin nykyisten resurssien ja ympäristön perusteella, kuten kuvassa 3. (de Byl, 2021)

Kuva 3. Esimerkki tavoitteellisen toimintasuunnittelun laatimasta suunnitelmasta (de Byl, 2021)



GOAP tekee koodista modulaarisemman ja helpommin ylläpidettävän, mahdollistaen helpon muokkaamisen ja päivittämisen. Tämä menetelmä on erityisen hyödyllinen peleissä, joissa halutaan toteuttaa monimutkaista agenttien käytöstä, koska se mahdollistaa toimintojen spontaanin lisäämisen tai poistamisen vaikuttamatta yleiseen toimivuuteen. Agentilla tarvitsee vain olla määritetty luettelo toiminnoista, jotka GOAP-suunnittelija käsittelee automaattisesti. (Chaduhari, 2017)

3.1 Maailmantilat

Maailmantila viittaa pelimaailman ja agentin ympäristön nykyiseen tilaan. Se voi sisältää tietoja, kuten esineiden sijainnin, agentin terveyden tilan, sekä muita ympäristötekijöitä, jotka voivat vaikuttaa agentin kykyyn saavuttaa tavoitteensa. Tila voi muuttua dynaamisesti agentin ja muiden pelimaailman olioiden toimintojen seurauksena. (Orkin, *Applying Goal-Oriented Action Planning to Games,* 2003)

Maailmantilojen avulla määritetään toimintojen edellytykset ja vaikutukset, sekä tarkistetaan, onko haluttu tavoite saavutettu. Ne auttavat myös suunnittelijaa ymmärtämään nykytilannetta ja tekemään päätöksiä siitä, mitkä toimet ovat mahdollisia ja olennaisia. (Chaduhari, 2017)

3.2 Tavoitteet

Tavoitteellisessa toimintasuunnittelussa tavoite on tietty tila tai päämäärä, jonka agentti saavuttaa. Agentilla voi olla useita tavoitteita samanaikaisesti, mutta kullakin hetkellä vain yksi tavoite on aktiivinen ja se ohjaa agentin käyttäytymistä. Tavoite pystyy laskemaan, kuinka merkittävä se on ja määrittämään milloin se on saavutettu. (Chaduhari, 2017)

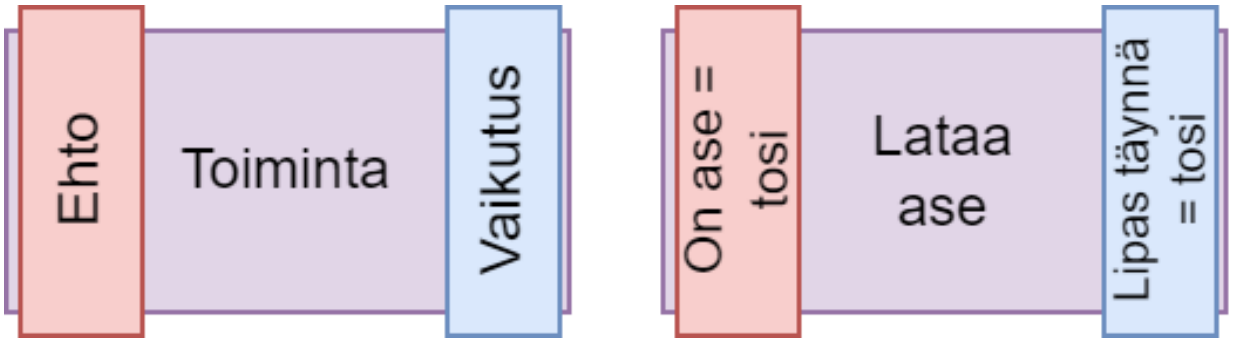
GOAP ei sisällä ennalta määriteltyä toimintasuunnitelmaa, vaan tavoitteet määrittelevät yksinkertaisesti ehdot, jotka on täytettävä niiden saavuttamiseksi. Näiden ehtojen saavuttamiseksi suoritettavat toimenpiteet määritetään reaaliajassa. (Orkin, *Applying Goal-Oriented Action Planning to Games,* 2003)

3.3 Toiminnat

Toiminnalla viitataan yksittäiseen vaiheeseen, jonka agentti suorittaa saavuttaakseen tavoitteen. Nämä toiminnot voivat olla yksinkertaisia, kuten tiettyyn sijaintiin liikkuminen, tai monimutkaisempia, kuten hyökkääminen haluttuun kohteeseen. Toiminnan kesto voi vaihdella, jotkut toiminnot valmistuvat nopeasti (kuten aseen lataaminen uudelleen) ja toiset jatkuvat toistaiseksi (kuten hyökkääminen kohteeseen), kunnes tietty ehto täyttyy. (Orkin, *Applying Goal-Oriented Action Planning to Games,* 2003)

Jokaisella toiminnolla on tietyt ennakkoehdot ja vaikutukset, jotka määräävät milloin se voidaan suorittaa ja mikä vaikutus sillä on pelimaailmaan. Näiden edellytysten ja vaikutusten avulla ketjutetaan toimintoja yhteen kelvolliseksi toimintosarjaksi. Esimerkiksi kohteen hyökkäämisen edellytyksenä voi olla, että hahmon ase on ladattu ja aseen uudelleenlatauksen vaikutus on, että aseessa on täysi lipas. Aseen lataus -esimerkin voi nähdä kuvassa 4. (Orkin, *Applying Goal-Oriented Action Planning to Games,* 2003)

Kuva 4. Esimerkki tavoitteellisen toimintasuunnittelun toiminnoista

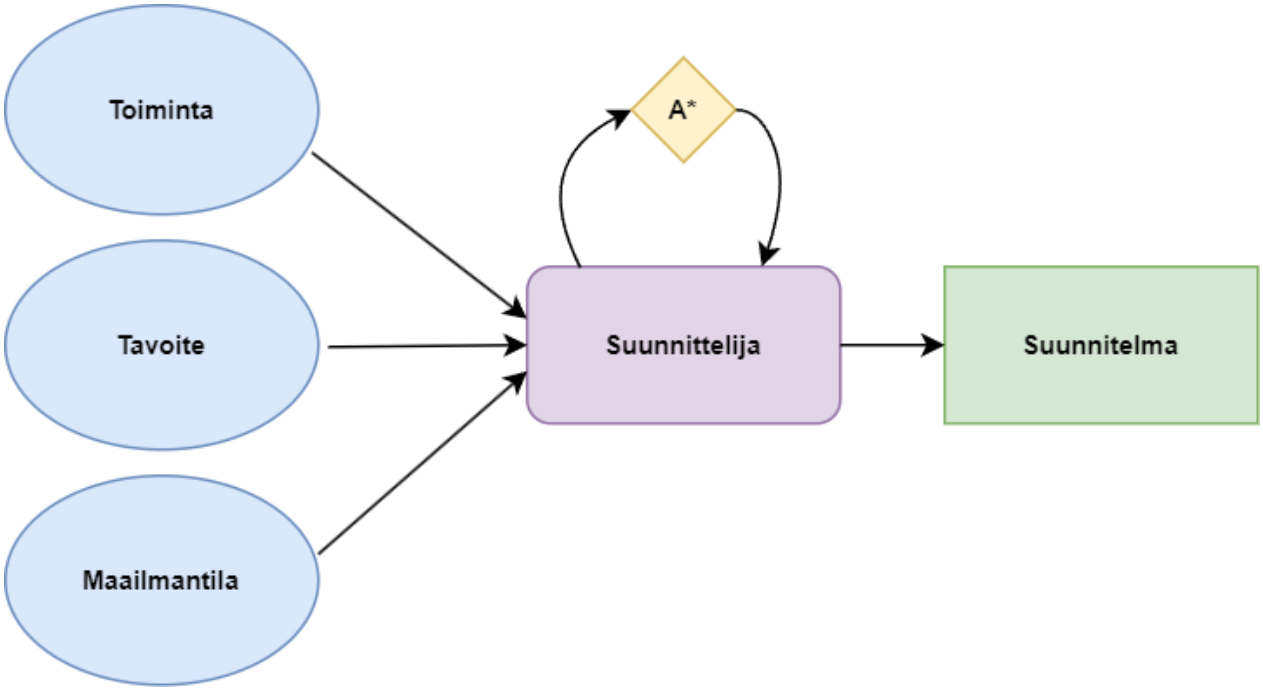


GOAP-järjestelmä yksinkertaistaa huomattavasti tarvittavaa äärellistä automaattia erottamalla tilasiirtymälogiikan itse tiloista. Toiminnot määrittelevät milloin siirtyä tilaan ja siitä pois, ja mitä tapahtuu pelimaailmalle siirtymän seurauksena, jolloin taustalla oleva FSM voi olla paljon yksinkertaisempi. (Orkin, *Applying Goal-Oriented Action Planning to Games,* 2003)

3.4 GOAP-suunnittelija

Tavoitteellisessa toimintasuunnittelussa agentti luo suunnitelman reaaliajassa antamalla tavoitteen järjestelmälle, jota kutsutaan suunnittelijaksi. Suunnittelijan toimintaa on havainnollistettu kuvassa 5. Suunnittelija laatii suunnitelman etsimällä käytettävissä olevia toimintoja löytääkseen sarjan, joka vie hahmon nykyisestä tilastaan haluttuun tavoitetilaan. (Orkin, *Applying Goal-Oriented Action Planning to Games,* 2003)

Kuva 5. Kuvaus siitä miten tavoitteellinen toimintasuunnittelu toimii



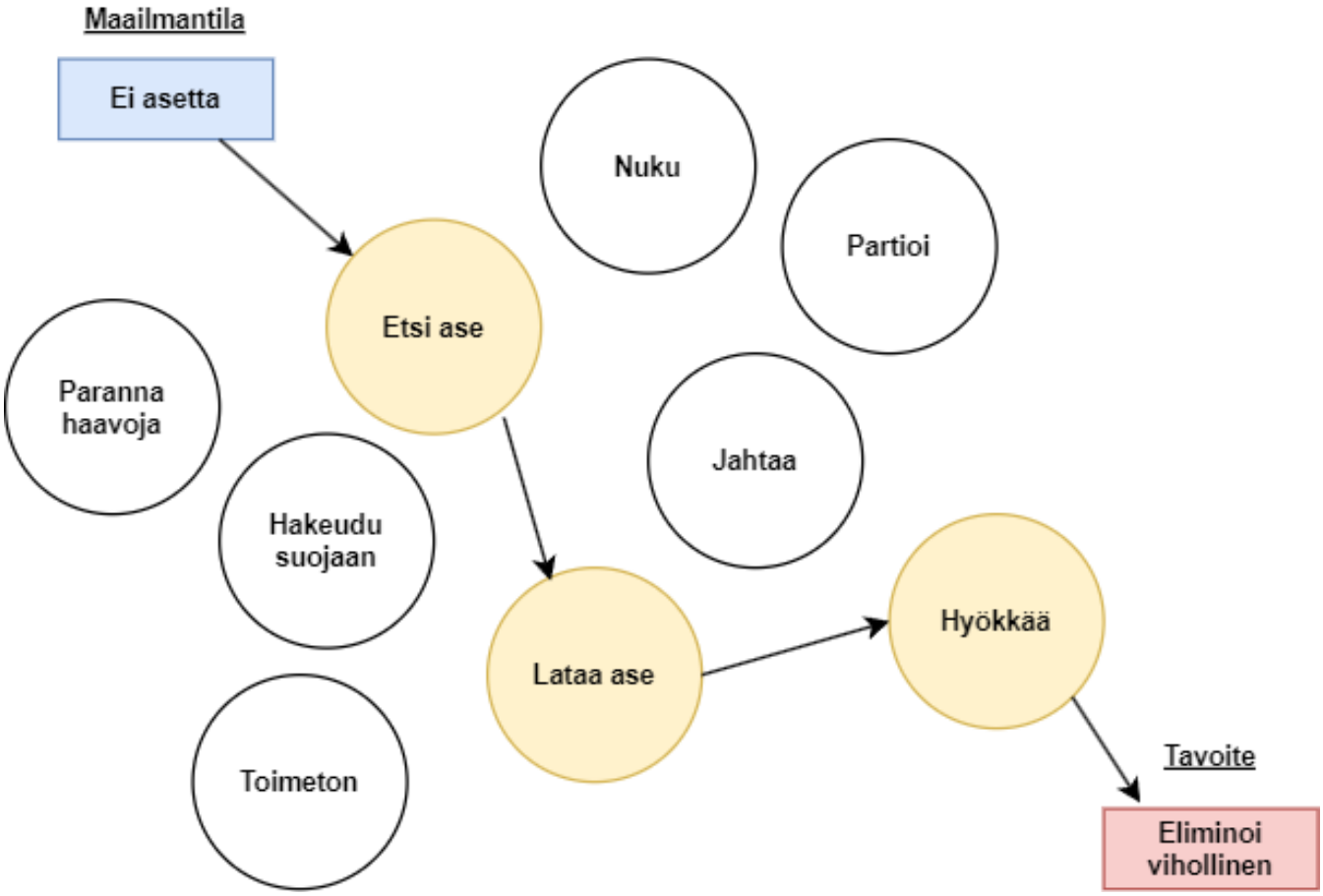
Jos suunnittelija onnistuu, se palauttaa agentille suunnitelman, jota se seuraa ja joka ohjaa sen käyttäytymistä. Agentti noudattaa tätä suunnitelmaa, kunnes se mitätöidään, tavoite on saavutettu tai muusta tavoitteesta tulee merkityksellisempi. Jos jokin muu tavoite aktivoituu tai nykyinen suunnitelma jostain syystä mitätöityy, hahmo keskeyttää nykyisen suunnitelman ja luo uuden. Päteviä suunnitelmia voi olla useita ja suunnittelijan tarvitsee löytää vain yksi niistä. (Orkin, *Applying Goal-Oriented Action Planning to Games,* 2003)

Polunetsinnän tapaan, suunnittelijan hakualgoritmille voidaan antaa tietoja ohjaamaan hakua. Esimerkiksi antamalla toiminnoille kustannukset, voidaan löytää halvin toimintosarja. (Orkin, *Applying Goal-Oriented Action Planning to Games,* 2003)

3.4.1 Suunnitelma

Suunnitelma on toimintosarja, joka vie agentin nykyisestä tilasta toiseen, jossa tavoite saavutetaan. Nämä toimet voidaan määrittää reaaliajassa vallitsevan tilanteen ja agentin tavoitteiden perusteella, kuten havainnollistettu kuvassa 6. (Orkin, *Applying Goal-Oriented Action Planning to Games,* 2003)

Kuva 6. Esimerkki tavoitteellisesta toimintasuunnittelusta



3.4.2 A*

A* (A tähti, englanniksi A-star) on suosittu algoritmi, jota useimmiten käytetään polunhaussa ja kuvaajan läpikäymisessä (graph traversal). Se on niin sanottu paras ensin -hakualgoritmi, joka löytää lyhimmän polun aloitussolmusta tavoitesolmuun. Tämä tapahtuu yhdistämällä solmun saavuttamisen kustannukset, arvioon tavoitteen saavuttamisesta kyseisestä solmusta. A*-algoritmia käytetään laajasti navigoinnissa ja pelien tekoälyssä. (Pound, 2017)

A*-algoritmia käytetään GOAP-menetelmän suunnitteluprosessissa. Suunnittelija käyttää algoritmia arvioidakseen solmut kustannusfunktion perusteella ja priorisoi ne solmut, joilla on alhaisemmat yhteiskustannukset. Täten se löytää optimaalisen toimintasuunnitelman tekoälyhahmolle. (Orkin, *Applying Goal-Oriented Action Planning to Games,* 2003)

4 Kehitysprojektin kuvaus

Kehitysprojektina luotiin kaksi simulaatiota, jossa itsenäiset agentit suorittavat toimintoja GOAP-tekniikan avulla. Projektit kehitettiin Unity-pelimoottoria ja C# -ohjelmointikieltä käyttäen. Ohjelmointiympäristönä toimi Microsoftin kehittämä Visual Studio. Projektin etenemistä seurattiin ja taltioitiin mukaillen SCRUM-menetelmiä ja pitäen yllä päiväkirjaa.

4.1 Unity yleisesti

Unity on Unity Technologiesin kehittämä ja alun perin vuonna 2005 julkaisema pelimoottori. Sitä käytetään videopelien kehittämiseen PC:lle, konsoleille, mobiililaitteille ja verkkosivustoille. Videopelikehityksen lisäksi sitä käytetään muun muassa elokuva- ja autoteollisuudessa.

Unity sisältää visuaalisen editorin, fysiikkamoottorin, sekä tuen 2D- ja 3D-grafiikalle, ja sen ohjelmointikielenä toimii C#. Unity tarjoaa myös käyttäjilleen Asset Store -palvelun, josta kehittäjät voivat ladata ilmaisia ja maksullisia resursseja käytettäväksi peleissään. Unity on laajalti käytetty pelikehitysteollisuudessa, ja se tunnetaan helppokäyttöisyydestään ja joustavuudestaan.

4.2 Scrum

Scrum on ketterässä kehityksessä käytetty projektinhallinnan viitekehys. Se korostaa yhteistyötä, sopeutumiskykyä ja toimivan tuotteen toimittamista vähitellen "sprintteinä"

tunnettujen kehityssyklien kautta, joiden aikana työstetään kehitysjonossa olevia tehtäviä. Scrumtiimi, johon kuuluu Scrum Master, tuotteen omistaja ja kehitystiimi, työskentelee yhdessä tuotteen suunnittelussa, rakentamisessa ja jatkuvassa parantamisessa. Sprinttien lisäksi Scrumin käytäntöihin kuuluvat sprinttien suunnittelupalaverit, sprinttikatselmus, retrospektiivi, sekä lyhyet, enintään 15 minuuttia kestävät, päivittäiset palaverit. (ScrumAlliance, 2020)

Kehitysprojektiin kehitysjonoa, eli backlogia, ylläpidettiin käyttäen GitHub Projects-palvelua. Työn aikana kirjasin myös muistiinpanoja ja päiväkirjaa, joiden avulla mukailin Scrum-käytänteitä soveltumaan itsenäiseen työskentelyyn.

5 Projektin toteutus

Työssä luotiin kaksi kooltaan ja mekaaniselta vaativuudeltaan erisuuruista projektia. Tarkoituksena oli testata GOAP:in soveltuvuutta erilaajuisiin projekteihin.

Ensimmäinen projekti oli yksinkertainen ravintolasimulaatio ja toinen oli hieman monimutkaisempi sotasimulaatio. Toinen projekti ei valmistunut tämän työn aikana, mutta eteni tarpeeksi pitkälle, jotta siitä saatiin tehtyä havaintoja tätä työtä varten.

Unityn asennuksen jälkeen ensimmäinen tehtävä oli suunnitella halutut toiminnot simulaatioissa oleville agenteille. Tämän jälkeen luotiin haluttu peliympäristö, sekä koodipohja GOAP-toiminnallisuuksille.

5.1 Peliympäristön luonti

Peliympäristö luotiin käyttäen niin sanottua greyboxing -tekniikkaa. Tämä tarkoittaa sitä, että peliympäristö rakennettiin käyttäen yksinkertaisia 3D-objekteja, kuten kuutioita ja kapseleita (jackw-gamedesign, 2016). Tämä saavutettiin täysin käyttäen Unityn sisältämiä muotoja ja toiminnallisuuksia.

5.1.1 NavMesh

Navmesh, lyhenne sanoista "navigation mesh", on tietorakenne, jota käytetään videopelien kehityksessä ja robotiikassa kuvaamaan alueita peli- tai simulaatioympäristössä, joissa on mahdollista liikkua. Se on kokoelma toisiinsa liittyviä polygoneja tai kolmioita, jotka muodostavat 3D-verkon, joka edustaa ympäristön navigoitavia pintoja, kuten lattioita, seiniä ja esteitä.

Navmesh-verkkoa käyttämällä agentit voivat liikkua ympäristössä törmäämättä seiniin tai muihin esteisiin. Navmesh tarjoaa tavan laskea optimaalinen reitti paikasta toiseen ottaen huomioon ympäristön geometrian ja tiellä mahdollisesti olevat esteet.

Navmesh on hyödyllinen työkalu realististen, dynaamisten ympäristöjen luomiseen, joissa hahmot ja agentit voivat liikkua luonnollisella ja intuitiivisella tavalla. (gamedev.net, 2018)

Unity tarjoaa oman sisäänrakennetun NavMesh toiminnallisuuden, jota käytettiin projekteissa hahmojen liikkumisen mahdollistamiseen. (Unity, 2021)

5.1.2 TextMeshPro

TextMeshPro (TMP) on Unityn oma tekstinrenderöintityökalu. Se tarjoaa runsaasti työkaluja tekstin ulkonäön ja käyttäytymisen mukauttamiseksi, sekä huomattavasti paremman suorituskyvyn kuin oletustekstikomponentit. Projektissa tätä käytettiin tiedon visualisointiin dynaamisesti tekstin avulla. (Unity, 2021)

5.2 GOAP Toteutus

GOAP-toiminallisuuksia saavuttamiseksi tarvittiin ensin C#-luokat jokaiselle toimintasuunnittelun osalle. Näitä luokkia käyttäen ja niistä periyttäen oli mahdollista luoda halutut agentit ja niiden halutut toiminnat.

Työssä käytetty versio GOAP:ista perustui Penny de Bylin luomaan pohjaan, jota muokaten rakennettiin simulaatioiden toiminnallisuudet. Työssä ei käydä yksityiskohtaisesti läpi itse koodia, vaan enemmänkin sen toiminnallisuutta.

5.2.1 World ja WorldStates

World-luokka, tai maailmaluokka, on keskeinen komponentti, joka hallitsee ja seuraa kaikkia toimintasuunnittelussa tarvittavia olennaisia muuttujia. Se on vastuussa näiden muuttujien päivittämisestä pelin nykyisen tilan ja muiden ulkoisten tekijöiden perusteella. Maailmanluokka sisältää myös WorldStates-luokan, joka kuvaa pelimaailman nykytilaa.

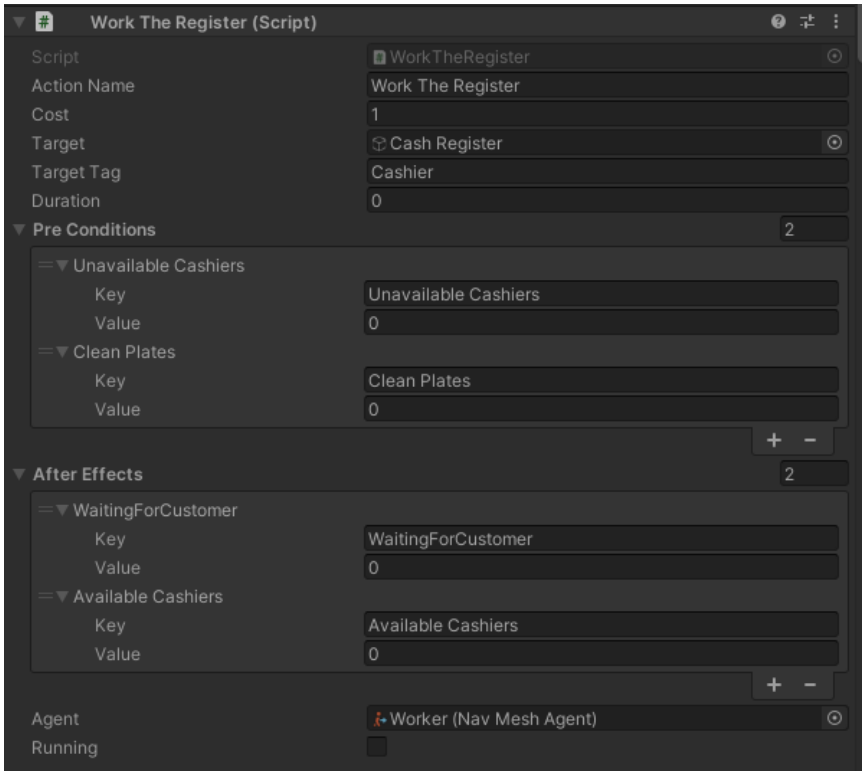
WorldStates-luokka on kokoelma tila-arvo-pareja, jotka edustavat pelimaailman tämänhetkistä tilaa. Näitä tila-arvo-pareja voidaan muokata pelin edetessä ja kun pelaaja tekee erilaisia toimintoja. Seuraamalla pelimaailman tilaa WorldStates-luokka mahdollistaa tekoälyn toimintojen suunnittelun ja suorittamisen siten, että ne ovat yhdenmukaisia pelimaailman nykytilan kanssa.

5.2.2 Agent, Action ja Inventory

Projektissa olevien agenttien tavoitteet ja toiminnot saadaan käyttäen Agent- ja Action-luokkia.

Action-luokan avulla voidaan luoda haluttuja toimintoja, jotka voidaan sen jälkeen määrittää halutuille agenteille. Näillä toiminnoilla on nimi, hinta, kohdepeliobjekti, jonka luokse saavuttua haluttu toiminto suoritetaan, toiminnon suorittamiseen kuluva aika, sekä kokoelmat kaikista toiminnon ehdoista ja vaikutuksista pelimaailmaan. Kuvaus työntekijä agentit ”Work The Register”-toiminnasta kuvassa 7.

Kuva 7. Työntekijähahmon ”Work The Register”-toiminta ravintolasimulaatiossa



Agent-luokasta periyttämällä voidaan määrittää tekoälyhahmon halutut tavoitteet, niiden painoarvon, sekä halutaanko tavoite suorittaa toistuvasti pelin aikana. Luokka myös visualisoi listan agentin tämänhetkisistä toiminnoista, ja suorittaa nämä perustuen suunnittelijan luomaan suunnitelmaan.

Kun agentti luodaan pelimaailmaan, se tarkistaa siihen liitetyt toiminnot ja listaa ne taulukkoon myöhempää käyttöä varten. Agentti tarkistaa jatkuvasti nykyisen toimintatilansa alkaen tutkimalla, onko seuraava toiminto määritetty. Jos toiminto on jo määritetty, agentti jatkaa toiminnon suorittamista.

Tapauksessa, jossa toimintoa ei ole määritetty, agentti tutkii sille määritettyjen toimintojen sarjan. Jos toimintoja on, agentti poimii seuraavan toiminnon jonosta. Jos toimintojonoa ei ole ollenkaan, agentti luo itselleen uuden suunnittelijakomponentin. Tämä suunnittelija luo toimintasuunnitelman, joka johtaa agentin nykyisen tavoitteen saavuttamiseen. Kun suunnitelma on luotu, agentti suorittaa suunnitelmassa määritellyn toimintojen sarjan.

Agentti päivittää myös Worldstate-tilansa heijastamaan muutoksia, joita pelimaailmaan on tehty suorittamalla toimintoja. Näitä pelimaailmantiloja on kahdenlaisia; maailmantilat, jotka vastaavat koko pelimaailmaan vaikuttavia muutoksia, ja hahmon itseensä liittyvä tila. Agentti voi luoda uuden suunnitelman, jos nykyinen suunnitelma ei ole enää voimassa pelimaailman muutosten tai muiden odottamattomien tapahtumien vuoksi.

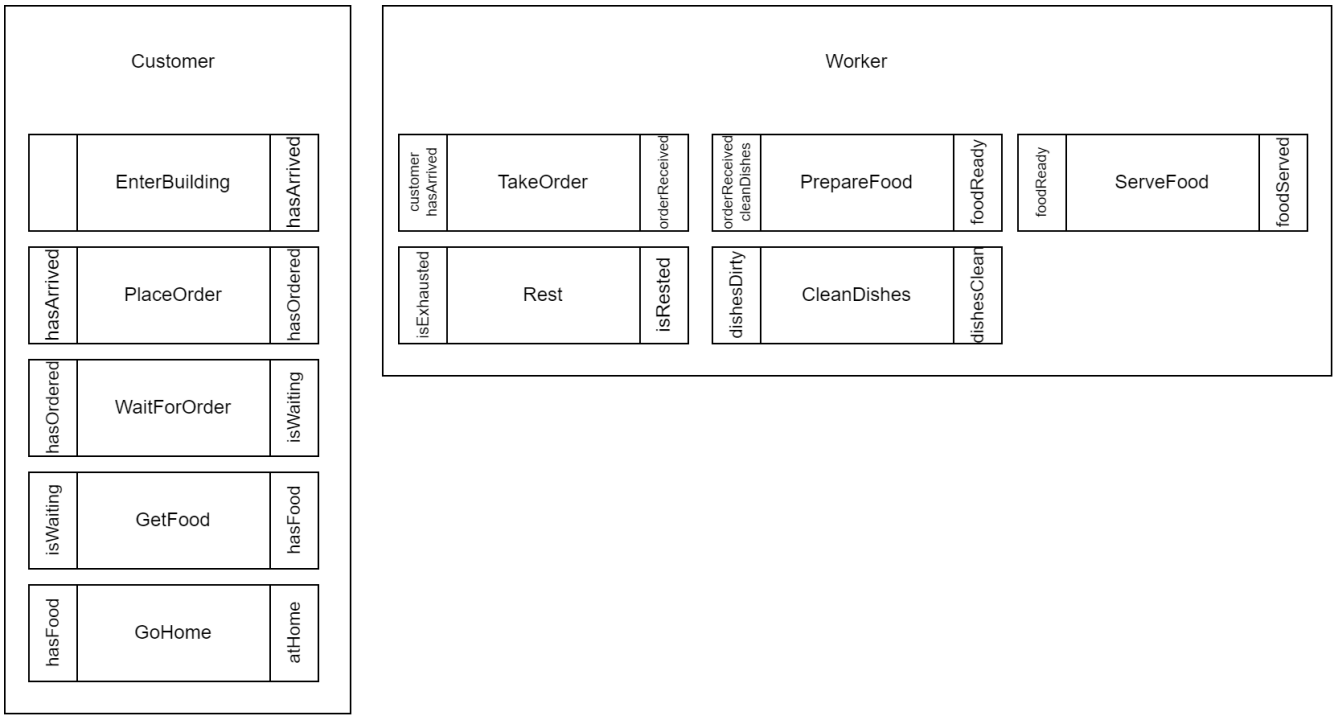
5.2.3 Planner

Agentti kutsuu Planner -, eli suunnittelijaluokkaa ja välittää sille nykyisen tilansa ja halutun tavoitetilan. Planner -luokka käyttää sitten A*-hakualgoritmia luodakseen toimintasuunnitelman, jonka agentti voi suorittaa tavoitteen saavuttamiseksi.
Luotu suunnitelma palautetaan Agent -luokkaan, joka suorittaa toimenpiteet peräkkäin, kunnes tavoite on saavutettu.
Jos suunnitelma ei saavuta tavoitetta, Agent -luokka voi kutsua Planner C# -luokan uudelleen luodakseen uuden suunnitelman.

5.3 Ravintolasimulaatio

Ensimmäisen projektin tarkoituksena oli luoda yksinkertainen ravintolasimulaatio, jonka avulla testattiin tavoitteellisen toimintasuunnittelun sopivuutta ja toiminnallisuutta pienimuotoisissa projekteissa. Simulaatio sisältää kahdenlaisia agentteja: asiakas ja työntekijä. Agenttien suunnitellut toiminnot on kuvattu kuvassa 8.

Kuva 8. Halutut toiminnallisuudet ravintolasimulaation agenteille



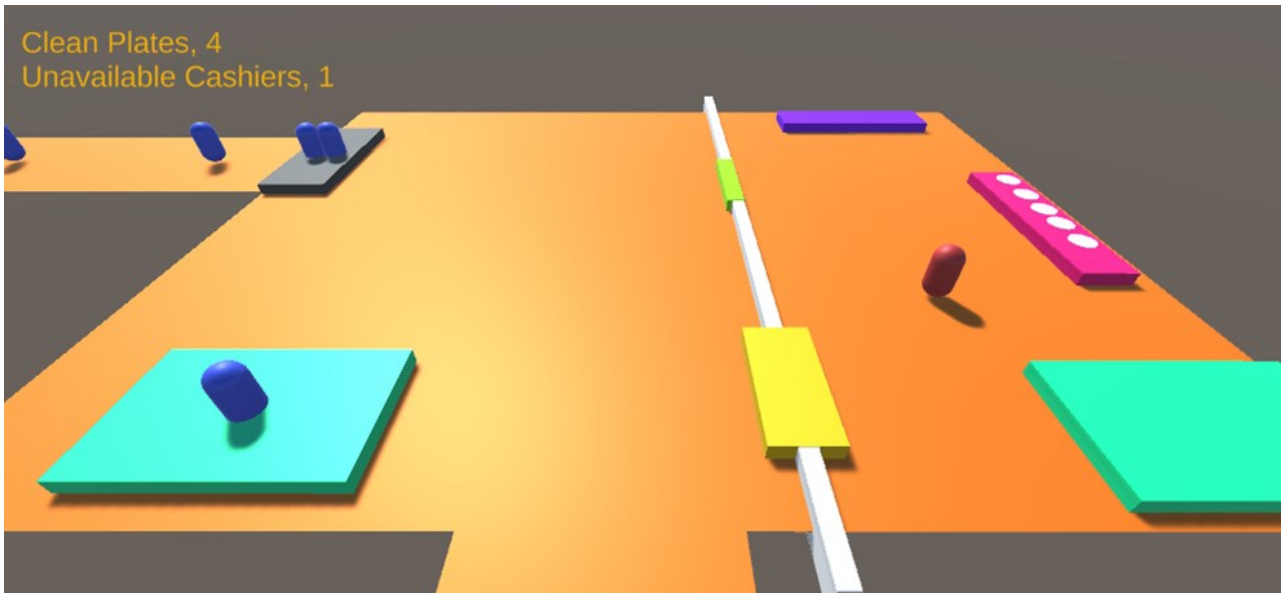
Ravintolassa työskentelevät hahmot omaavat neljä tavoitetilaa, jotka ovat ”odota tilausta”, ”tarjoile ruoka”, ”puhdista astiat”, sekä ”lepää”. Ravintolaan tulevilla asiakkailla puolestaan on yksi tavoite, ”mene kotiin”. Mekaanisesti hahmot ovat kuitenkin yksinkertaisia. Ne kykenevät ainoastaan liikkumaan ja odottamaan.

Asiakas agentin haluttiin pystyvän suorittamaan yksinkertainen sarja toimintoja, jotka alkavat sen saapumisesta rakennukseen ja loppuu kun se lähtee kotiin.
Työntekijä agentilla on kolme toimintojen sarjaa. Ensimmäinen on palvella asiakasta, toinen on pestä likaiset astiat kun tietty määrä asiakkaita on palveltu ja kolmas on mennä lepäämään.

Asiakas ja työntekijä agentit ovat riippuvaisia toisistaan, sillä asiakas ei voi edetä tilaamaan ruokaa jos paikalla ei ole vapaana olevaa työntekijää. Asiakas ei voi myöskään lähteä kotiin ennen kuin työntekijä on suorittanut tehtäväsarjansa. Työntekijä puolestaan ei voi suorittaa sarjaansa jos astioista ei ole puhtaana.

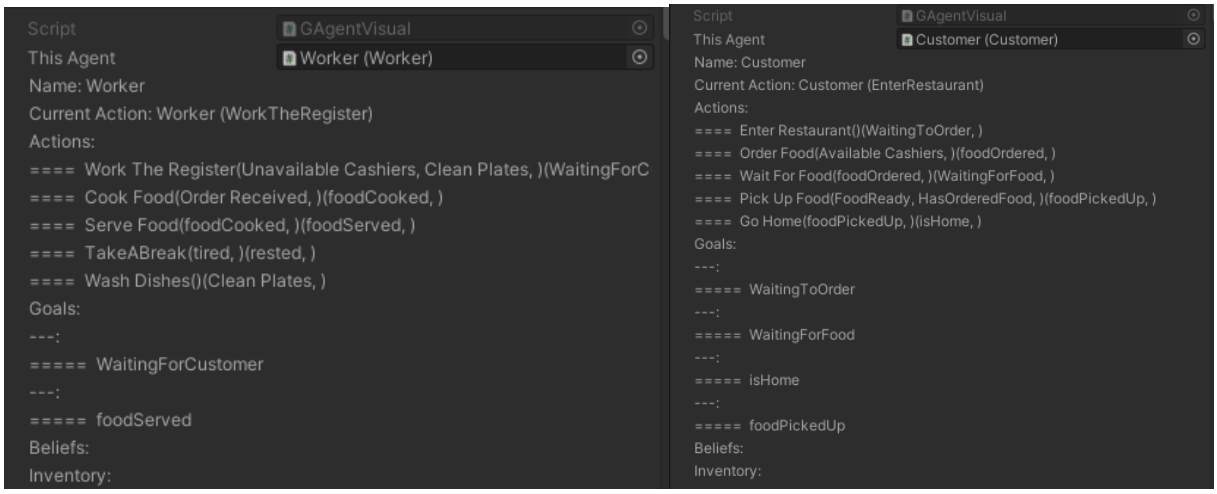
Tuloksena syntyneessä simulaatiossa asiakasagentti saapuu ravintolaan ja odottaa ovella, kunnes työntekijä agentti on vapaana. Tämän jälkeen asiakasagentti liikkuu kassalla, tilaa ruoan, ja siirtyy odotusalueelle odottamaan tilauksen valmistumista. Kuvaus simulaatiosta kuvassa 9.

Kuva 9. Kuvaus ravintolasimulaatiosta



Kun työntekijäagentti on suorittanut tilauksen valmistamisen, saapuu asiakas hakemaan sen noutotiskiltä, jonka jälkeen se lähtee kotiin. Agenttien toiminnat visualisoituna Unityn inspector-ikkunassa kuvassa 10.

Kuva 10. Agenttien toimintasarjat ja tavoitteet



Työntekijäagentti odottaa kassalla jos sillä ei ole muuta tehtävää. Tilauksen saatuaan asiakkaalta se siirtyy valmistusasemalle, jonka jälkeen se vie valmiin tilauksen noutotiskille. Kun tilauksia on valmistettu viisi kappaletta, täytyy työntekijäagentin pestä tiskit tiskialtaalla, ennen kuin se voi ottaa uuden tilauksen.

5.4 Projekti 2

Toisessa projektissa kaksi joukkuetta hahmoja käyvät sotaa määritellyllä pelialueella. Simulaation tarkoituksena oli testata toimintasuunnittelun toimivuutta tilanteessa, jossa hahmoilla on enemmän vaihtoehtoja toiminnoilla, sekä enemmän mekaanisia toimintoja. Tämä simulaatioprojekti ei valmistunut halutulla tavalla opinnäytetyön loppuun mennessä, mutta se kehittyi kuitenkin tarpeeksi, jotta siitä saatiin tehtyä havaintoja menetelmän soveltuvuudesta.

Simulaatiossa hahmot lähtevät liikkeelle ennalta määritellystä lähtökohdasta, ja etenevät satunnaisesti valittuja pisteitä kohti pelialueella. Kun hahmo törmää vihollisjoukkueen hahmoon, se tähtää ja ampuu sitä. Hahmot voivat myös haavoittua, jolloin toinen samassa joukkueessa oleva hahmo yrittää elvyttää sen. Hahmoilla on vain tietty määrä ammuksia, joiden loputtua on niitä haettava lisää, joko ammuslaatikosta, tai poimimalla eliminoidun vihollisen ase. Hahmot voivat myös tarvittaessa hakeutua suojaan.

Simulaatio perustui samaan GOAP-pohjaan, jota käytettiin ensimmäisessä projektissa. Huomattavat erot simulaatioiden väillä olivat erilaisten toimintojen määrä, toistensa kanssa toiminnassa olevien hahmojen samanaikainen määrä, tieto, jota hahmojen täytyi pitää muistissa itsenäisesti, sekä mekaaninen monimutkaisuus.

6 Yhteenveto

Työn lopputuloksena voidaan todeta, että GOAP on erittäin mielenkiintoinen aihealue ja sillä on selviä käyttötarkoituksia suuremmissa pelikokonaisuuksissa. En kuitenkaan usko, että se oli juuri paras tekoälymenetelmä simulaatioissa luomiini pienempiin pelikokonaisuuksiin.

Kun toimintasuunnittelun perustoiminnallisuudet on saatu toimiviksi, on kyseessä erittäin modulaarinen systeemi. Toimintoja on todella helppo lisätä, poistaa, muokata ja tarkkailla reaaliajassa, varsinkin jos tätä varten luodaan jonkinlainen työkalu, joka visuaalisesti näyttää hahmojen toimintaketjut.

Toisaalta, se että toimintasuunnittelun perustoiminnallisuudet saadaan valmiiksi, on itsessään jo erittäin työlästä. Jos koodia lähdetään rakentamaan itse alusta alkaen, täytyy työtä tehdä henkilö, joka on erittäin osaava ohjelmoija. Jos taas halutaan käyttää valmiina löytyviä GOAP-toteutuksia, ovat ne usein joko toiminnoiltaan puutteellisia, koodi on vanhentunutta tai muuten vaikeasti ymmärrettävää kokemattomalle ohjelmoijalle, joko siitä syystä, että dokumentaatio on huonoa, tai yksinkertaisesti koska koodin määrä on niin suurta.

Mahdollisten vikojen korjaus on myös vaikeaa. Koska hahmojen toiminta perustuu toimivaan ketjuun suoritettavia toimintoja, jos yksi näistä toiminnoista ei toimi halutulla tavalla, ei hahmo lähde suorittamaan ketjua ollenkaan. Jos ketjussa siis on yksikin ongelmakohta, ei hahmo tee yhtään mitään. Muissa tekoälytoteutuksissa olisi oletettavasti helpompaa löytää vikakohta, sillä hahmo luultavasti suorittaisi toiminnot vikaan asti, ja pysähtyisi siihen.

Toimintasuunnittelun ketterä kehitys on myös tavoin haastavaa. Koska työtä täytyy tehdä niin paljon, jotta perustoiminnallisuudet saadaan valmiiksi, pitää suunnitella tarkkaan jo etukäteen mitä toiminnoilta vaaditaan. Kun itse projektia lähdetään kehittämään, nousee kuitenkin varmasti tarpeita uusille ominaisuuksille ja näitä voi olla erittäin vaikea lähteä lisäämään niin suureen ohjelmistokokonaisuuteen jälkikäteen.

Kun luodaan varsinkin ensimmäisen projektin kaltaisia yksinkertaisia kokonaisuuksia, GOAP vaikuttaa lopulta tuottavan enemmän työtä kuin olisi tarve. Se määrä tietoa jota oli

pidettävä yllä maailmantilassa, hahmojen sisäisessä tilassa, sekä tavaranhallinta systeemissä, teki projektista paljon monimutkaisemman kuin mitä sen olisi tarvinnut olla. Jos projektissa olisi käytetty esimerkiksi äärellistä tilakonetta, tai edes käytöspuuta, olisi kaikki toiminnallisuus saatu käyttöön huomattavasti pienemmässä ajassa ja pienemmällä vaivalla. Tässä tapauksessa koodin modulaarisuudesta koituva hyöty ei mielestäni ollut suurempaa kuin se työmäärä joka menetelmästä koitui.

GOAP ei siis vaikuta olevan oikea ratkaisu projekteihin, joissa hahmoilla ei ole suurta määrää toimintoja, kuten tässä työssä tehty ensimmäinen simulaatio. Toinen simulaatio puolestaan vaikutti olevan tähän menetelmään sopivampi. Suoritustehosta ei koitunut ongelmia, mutta en myöskään olettanut, että näin pienissä projekteissa tällaisia ongelmia olisi ilmennyt.

Lopulta saatiin projektit, joiden avulla pystyttiin testaamaan tehtävän suunnittelijaa halutulla tavalla. GOAP-selvästi soveltuu parhaiten monimutkaisempiin kokonaisuuksiin, ja jatkossa olisikin tavoitteena päästä työstämään pelikokonaisuutta, joka vaatii monimutkaisemman tekoälyn GOAP:in tavoin.

Lähteet

Orkin, J. (n.d., haettu 2023). *Goal-Oriented Action Planning (GOAP).* *https://alumni.media.mit.edu/~jorkin/goap.html*

Doris, K & Silvia, D. (2007). *Improved Missile Route Planning and Targeting using Game-Based Computational Intelligence.* 2007 IEEE Symposium on Computational Intelligence in Security and Defense Applications. https://ieeexplore.ieee.org/document/4219083

Chaduhari, V. (2017). *Goal Oriented Action Planning.* *https://medium.com/@vedantchaudhari/goal-oriented-action-planning-34035ed40d0b*

Orkin, J. (2003). *Applying Goal-Oriented Action Planning to Games.* https://alumni.media.mit.edu/~jorkin/GOAP_draft_AIWisdom2_2003.pdf

iHeartGameDev. (2021). *How to Program in Unity: State Machines Explained.* https://www.youtube.com/watch?v=Vt8aZDPzRjI

Millington, I & Funge, J. (2009). *Artificial Intelligence for Games: Edition 2*. CRC Press.

Simpson, C. (2014). *Behavior trees for AI: How they work.* Game Developer. https://www.gamedeveloper.com/programming/behavior-trees-for-ai-how-they-work

Isla, D. (2005). *GDC 2005 Proceeding: Handling Complexity in the Halo 2 AI*. https://www.gamedeveloper.com/programming/gdc-2005-proceeding-handling-complexity-in-the-i-halo-2-i-ai

de Byl, P. (2021). *Goal Driven Behaviour.* Unity Learn https://learn.unity.com/project/goal-driven-behaviour?uv=2019.4

Pound, Mike. (2017). *A* (A Star) Search Algorithm – Computerphile.* Computerphile. https://www.youtube.com/watch?v=ySN5Wnu88nE

ScrumAlliance. (2020). *Scrum Overview.*
https://www.scrumalliance.org/about-scrum

Unity. (2021). *TextMeshPro.*
https://docs.unity3d.com/Manual/com.unity.textmeshpro.html

Unity. (2021). *Building a NavMesh.*
https://docs.unity3d.com/Manual/nav-BuildingNavMesh.html

gamedev.net. (2018). *Navigation Meshes and Pathfinding.*
https://www.gamedev.net/tutorials/programming/artificial-intelligence/navigation-meshes-and-pathfinding-r4880/

jackw-gamedesign. (2016). *What is Greyboxing?*
https://jackw-gamedesign.tumblr.com/post/139960850160/what-is-greyboxing

Linkki tässä työssä mainittuun ravintola simulaatioon. 2023. Sami Lappalainen
https://samitl.itch.io/goap-v1

Liite 1: Aineistonhallintasuunnitelma

Kehitysprojektin aikana pidetään päiväkirjaa, johon kerätään havaintoja projektin etenemisestä. Tämä tieto analysoidaan opinnäytetyötä varten. Päiväkirjaa säilytetään tekijän tietokoneen C-asemalla. Päiväkirjan lisäksi työn etenemistä ylläpidetään käyttäen GitHubin Projects-palvelua. Päiväkirjaa säilytetään C-asemalla ainakin vuoden verran opinnäytetyön valmistumisesta.

Kehitysprojektin seurauksena syntyneitä tiedostoja säilytetään tekijän tietokoneen C-asemalla. Tiedostoja säilytetään C-asemalla ainakin vuoden verran opinnäytetyön valmistumisesta.

Projektin aikana valmistunut simulaatio on saatavilla itch.io palvelusta seuraavasta osoitteesta: https://samitl.itch.io/goap-v1
Simulaation on tarkoitus olla saatavilla mainitussa osoitteessa pysyvästi.