Janne Sillanpää

Pelitekoälyn päätöksenteko

Tradenomi

Tietojenkäsittely

Syksy 2020

KAMK • University of Applied Sciences

Tiivistelmä

Tekijä: Sillanpää Janne

Työn nimi: Pelitekoälyn päätöksenteko

Tutkintonimike: Tradenomi, tietojenkäsittely

Asiasanat: tekoäly, pelitekoäly, päätöksenteko

Opinnäytetyössä tutkittiin pelitekoälyä ja sen päätöksentekoa. Tarkoituksena oli antaa selkeä kuva erilaisista päätöksentekotekniikoista sekä selostaa, miten näitä tekniikoita on sovellettu erilaisissa peleissä. Selitykset tapahtuivat teoriatasolla.

Työ aloitettiin ensiksi selittämällä mitä eroa on akateemisella tekoälyllä ja pelitekoälyllä. Tässä työssä keskityttiin pelitekoälyyn ja akateeminen jätettiin kokonaan pois. Tämän lisäksi selitettiin mikä on tekoälyn tarkoitus peleissä ja mitä päätöksenteko tarkoittaa tässä yhteydessä. Tärkeimmät asiat pelitekoälyn toteutukselle olivat hauskuus ja niin sanottu illuusio älykkyydestä.

Tekniikat, jotka käytiin läpi tämän työn aikana, olivat yksinkertainen päätöksenteko, äärelliset tilakoneet, käytöspuut, tavoitteellinen tehtäväsuunnittelija, hierarkkiset tehtäväverkostot, ohjaajajärjestelmä, Monte Carlo -menetelmä, hyötypohjainen päätöksenteko ja viimeiseksi neuroverkot.

Yksinkertainen päätöksenteko on helpoin tapa toteuttaa päätöksenteko. Tilakoneet ovat seuraava askel eteenpäin yksinkertaisesta päätöksenteosta ja ovat ensimmäinen oikea tekniikka, jota monet aloittelevat ohjelmoijat käyttävät. Se koostuu erilaisista tiloista ja niiden vaihtelusta. Käytöspuut ovat suosittu tekniikka nykyajan peleissä niiden joustavuuden ja helpon toteutuksen takia. Tavoitteellinen tehtäväsuunnittelija eroaa tilakoneista ja käytöspuista siinä, että sen sijaan, että se valitsisi suoraan tietyistä tiloista tai käytöksistä, se valitseekin useamman toimen, jotka siten johtavat tiettyyn lopputulokseen. Hierarkkiset tehtäväverkostot on hieman muokattu versio tavoitteellisesta tehtäväsuunnittelijasta, joka luotiin ratkaisemaan suurimpia ongelmia tehtäväsuunnittelijassa.

Ohjaajajärjestelmä hallitsee pelien kulkua suoraan, ja sitä hyödynnetään yhdessä muiden tekniikoiden kanssa. Monte Carlo -menetelmä on toimiva ratkaisu erilaisissa strategiapeleissä, joissa tekoälylle voidaan antaa paljon aikaa miettimiseen. Hyötypohjainen päätöksenteko on sekoitus käytöspuusta ja tehtäväsuunnittelijasta. Sen avulla voidaan luoda tekoäly, jonka toiminta on monimutkaisempaa kuin käytöspuu, mutta kehittäjällä on enemmän hallintaa sen toiminnasta kuin tehtäväsuunnittelijassa. Neuroverkot ovat koneoppimistekniikka, jota harvoin käytetään peleissä. Se eroaa täysin edellä mainituista tekniikoista toiminnaltaan siinä, että se on täysin autonomisesti päättävä tekniikka.

Nämä tekniikat valittiin niiden mielenkiintoisuuden ja tärkeyden perusteella. Tavoite oli antaa ensiksi lyhyehkö kuvaus eri tekniikoista ja sen jälkeen yksi peliesimerkki kustakin tekniikasta. Nämä pelit ovat tärkeitä merkkipaaluja pelitekoälylle. Selityksissä käytiin tekniikoiden hyviä sekä huonoja puolia läpi. Työn aikana todettiin, että kaikista tekniikoista löytyi molempia, koska täydellisiä tekniikoita ei ole vielä keksitty. Peliesimerkeissä annettiin kuva siitä, miksi kehittäjät valitsivat juuri tämän tekniikan ja kuinka se hyödytti heidän peliään.

Työn loppupuolella pohdittiin eri tekniikoita ja annettiin niistä tulevaisuuden ennustuksia. Kaikki käydyt tekniikat tulevat olemaan tärkeitä myös tulevaisuudessa, mutta varsinkin neuroverkkojen hyödyntäminen peleissä mahdollisesti lisääntyy. Nämä ennustukset perustuivat kokemuksiin sekä työn aikana käytyihin materiaaleihin.

KAMK • University of Applied Sciences

Abstract

Author(s): Sillanpää Janne

Title of the Publication: Decision Making in Game Artificial Intelligence

Degree Title: Bachelor of Business Administration, Information Technologies

Keywords: artificial intelligence, decision making

The subject of this Bachelor's thesis was Decision Making in Game Artificial Intelligence. The purpose was to go through different decision-making techniques and explain what they are and how have they been used in different games. These explanations were more on the theoretical side and did not delve into actual implementation.

The thesis began by explaining the difference between academic AI and game AI. In addition to this, it was explained what decision-making means in this context. The most important things for Game AI are fun and the so called "Illusion of intelligence".

The techniques that were explained in this thesis were: simple decision making, finite state machine, behavior trees, goal-oriented action planning, hierarchical task networks, director system, Monte Carlo -method, utility AI and neural networks.

Simple decision making is the easiest one to implement. State machines are the next step forward in simple decision making. They are often the first proper technique many novice programmers use. They consist of different states and their changes. Behavior trees are a popular technique in modern games, because of their flexibility and ease of implementation. Goal-oriented action planning differs from state machines and behavior trees in that, instead of choosing directly from certain states or actions, it chooses a certain number of different actions, that lead to a certain outcome. Hierarchical task networks is a slightly modified version of goal-oriented action planning. It was created to solve the biggest problems of goal-oriented action planning.

The director system directly controls the flow of the game and it is often utilized in conjunction with other techniques. The Monte Carlo method is a solution used in various strategy games where artificial intelligence can be given a lot of time to think. Utility based decision making is a mixture of behavioral trees and planners like goal-oriented action planning. It can be used to create an artificial intelligence that is more complex than a behavior tree, but the developer retains more control over it, than in a planner. Neural networks are a machine learning technique rarely used in games. It differs completely from the above-mentioned techniques in that, it is a completely autonomous decision-making technique.

These techniques were chosen by their importance and interest. The goal was to first give a short description of the different techniques and then a single game example for every technique. These games are important milestones for game AI. Explanations went through both good and bad sides to every technique. During this thesis it was found out that there is no "perfect" technique. The game examples gave a picture of why developers chose a specific technique and how it benefited their game.

Lastly different techniques were discussed, and predictions were given about their future. These predictions were based on experiences and sources that were used to create this thesis.

Sisällys

1 Johdanto 1

2 Pelitekoäly 2

2.1 Tekoälyn tarkoitus peleissä 2

2.2 Päätöksenteko tekoälyssä 3

3 Päätöksentekotekniikat 4

3.1 Yksinkertainen päätöksenteko 5

3.2 Äärelliset tilakoneet 6

3.3 Käytöspuu 8

3.4 Tavoitteellinen tehtäväsuunnittelija 13

3.5 Hierarkkiset tehtäväverkostot 14

3.6 Ohjaajajärjestelmä 17

3.7 Monte Carlo -menetelmä 19

3.8 Hyötypohjainen päätöksenteko 21

3.9 Neuroverkot 23

4 Pohdinta 25

5 Yhteenveto 27

Lähteet 29

1 Johdanto

Tämän työn tarkoitus on selvittää, mitä päätöksenteko pelitekoälyssä tarkoittaa ja sen jälkeen käydä läpi erilaisia päätöksentekotekniikoita. Aloitan työn käymällä läpi, mitä sana ”pelitekoäly” tarkoittaa tässä yhteydessä, jonka jälkeen siirryn pääasiaan eli pelitekoälyn päätöksentekoon. Tarkoitus on selostaa eri tekniikoiden hyviä ja huonoja puolia sekä tehdä siitä päätelmiä eli missä yhteydessä on parasta käyttää kutakin tekniikkaa. Kun käyn tekniikoita läpi, kerron niistä vain teoreettisesti, enkä mene niiden käytännölliseen toteutukseen. En siis anna kokonaisia koodi-toteutuksia.

Tämän saavuttaakseni otan esimerkkejä eri peleistä, joissa näitä tekniikoita on käytetty. Sen jälkeen analysoin, miksi juuri tätä tekniikkaa on käytetty tässä tapauksessa. Tekniikat, jotka käyn läpi, ovat suosittuja pelitekoälyn toteutuksessa ja niitä on sovellettu useammassa eri pelissä. Tässä tekstissä kuitenkin otan yleensä vain yhden peliesimerkin kustakin tekniikasta. Nämä pelit ovat tärkeitä merkkipaaluja pelitekoälylle.

Tarkoitukseni on antaa tämän tekstin aikana selkeä kuva erilaisista päätöksentekotekniikoista ja sen avulla auttaa kehittäjiä päättämään, mikä olisi paras päätöksentekotekniikka heidän peliprojektiinsa.

2 Pelitekoäly

Akateeminen tekoäly ja pelitekoäly eroavat toisistaan huomattavasti. Akateemisen tekoälyn yleinen tehtävä on optimoida sen käyttäytyminen mahdollisimman älykkääksi. Tämä ei yleisesti ottaen kuitenkaan toimisi pelitekoälyn kehittämisessä. Peleissä emme halua kaikista älykkäintä tekoälyä, vaan sen sijaan haluamme illuusion älykkyydestä.[1.] Tästä lähtien, kun puhun tekoälystä, tarkoitan pelitekoälyä enkä akateemista tekoälyä.

2.1 Tekoälyn tarkoitus peleissä

Yleisesti ottaen on vaikea sanoa, mikä on tekoälyn tarkoitus peleissä, koska kaikki pelit ovat erilaisia. Voidaan kuitenkin sanoa, että suurimmassa osassa peleistä pelitekoälyn päätarkoitus on tehdä pelaamisesta mielekästä. Tekoälyn suunnittelu täytyy tehdä tämä asia mielessä. Tekoälyn tarkoitus on myös tukea pelin niin sanottua ”tarkoitettua kokemusta”. Kehittäjän vastuulla on se, että pystyy luomaan uskottavan illuusion niin, että pelaaja käyttäytyy tekoälyä kohtaan niin kuin se olisi oikea. [1.]

Epäonnistunut tekoälytoteutus on sellainen, että jossakin vaiheessa tapahtuu jotakin, joka muistuttaa pelaajaa siitä, että tekoälyhahmo on vain tietokoneohjelma. Tämä ei kuitenkaan tarkoita sitä, että täytyisi luoda hyvin monimutkainen algoritmi. Ihmisen mieli toimii niin, että jos jokin näyttäisi siltä, että se voisi tapahtua tässä tilanteessa, niin yleensä me hyväksymme sen oikeana käyttäytymisenä. Jos tekoälyn toiminta näyttää järkevältä, niin pelaaja usein mielessään luo selityksen sille, miksi tällainen toiminta tapahtui. Tämä selitys on usein paljon monimutkaisempi, kuin se mikä tekoälyn toteutus on. [1.]

Tekoälyä ei kannata rakentaa liian älykkääksi, koska tämä voi johtaa tekoälypäätöksiin, jotka ovat järkeviä tietokoneelle, mutta eivät välttämättä ihmiselle. Tärkein asia, mitä täytyy välttää tekoälyä tehdessä, on keinotekoinen tyhmyys. Esimerkiksi ihmiset usein vaihtavat mieltään kesken kaiken, mutta jos tekoäly tekee tällaista, se voi näyttää tyhmältä pelaajan silmissä. Suosittu tapa ilmaista, mitä tietyt tekoälyhahmot ajattelevat milläkin hetkellä, on antaa pelaajalle äänivihjeitä. Esimerkiksi hahmo voi huutaa: ”Minuun osui!” Tämän ainoa tavoite on saada pelaajalle ymmärrys siitä, mitä tietokone ajattelee, eikä hahmojen välinen kommunikointi. [1.]

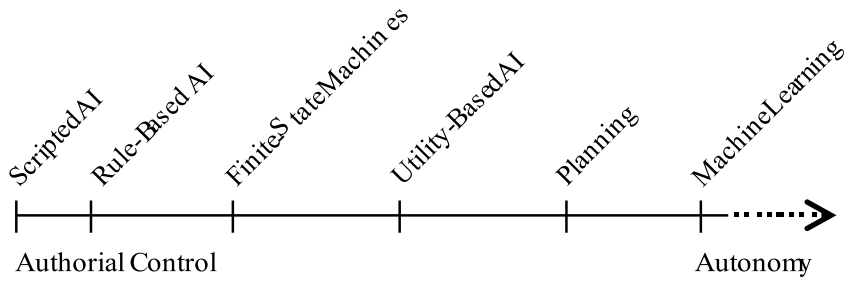
2.2 Päätöksenteko tekoälyssä

Päätöksenteko tekoälyssä tarkoittaa sitä, miten erilaiset tietokoneohjatut hahmot päättävät siitä, mitä ne tekevät seuraavaksi. Tällaisia toimia voisivat olla esimerkiksi hyökätä, kävellä, juosta, seurata tai muuta sellaista. Päätöksenteon toteuttamiseen on luotu erilaisia tekniikoita. Näiden tekniikoiden tarkoitus on selvittää, mitä tietokoneohjattujen hahmojen kuuluu tehdä milläkin hetkellä. Päätöksentekotekniikoiden toteutukset vaihtelevat yksinkertaisista monimutkaisiin ratkaisuihin. [2.]

Vaikka päätöksentekotekniikat eroavat toisistaan, on niillä myös jotakin yhteistä. Kaikkien päätöksentekotekniikoiden perusideana on se, että ne tarvitsevat jonkinlaista tietoa maailmasta, jonka avulla ne voivat tehdä päätöksiä. Tällaista tietoa voisi olla esimerkiksi, että pelaaja on lähellä tekoälyhahmoa. Päätöksentekotekniikka siten ottaisi tämän tiedon ja käyttäisi sitä sen seuraavan päätöksen tekemiseen.

3 Päätöksentekotekniikat

Päätöksentekotekniikoita on monenlaisia, ja on usein kiistelty, mikä näistä tekniikoista sopisi parhaiten tekoälyn toteuttamiseen. Vastausta tähän kysymykseen ei ole vieläkään löydetty. Tämä tietenkin johtuu siitä, että kaikki pelit ovat erilaisia ja ne tarvitsevat uniikkeja tekoälyratkaisuja. Tämä ei kuitenkaan tarkoita sitä, että ei olisi olemassa hyviksi todettuja tekoälytekniikoita peleihin käytettäväksi. Kuvassa 1 nähdään erilaisia päätöksentekotekniikoita listattuna ja niiden tyyppi.



Kuva 1. Päätöksenteontyyppi määritetystä täysin autonomiseen [1]

Kaikki päätöksentekotekniikat voidaan luokitella janalle, jossa yhdessä päässä on täysin määritelty tekoäly (Authorial Control) ja toisessa päässä täysin autonomisesti päättävä tekoäly (Autonomy) [1]. Täysin autonomisesta tekoälystä esimerkkinä on koneoppiminen (Machine Learning). Koneoppimisesta tulen kertomaan myöhemmin, mutta keskityn lähinnä tekniikoihin, jotka suurimmaksi osaksi toimivat ihmisten suunnittelemalla tavalla.

Täysin autonomista tekoälyä harvoin käytetään pelitekoälyn luomisessa, koska on tärkeää pystyä hallitsemaan pelaajalle haluttua kokemusta peleissä. Kehittäjinä haluamme, että pelaajat kokevat meidän pelimme halutulla tavalla. Tässä tapauksessa kaikkien päätöksien jättäminen tietokoneen haltuun ei välttämättä tuota niitä tuloksia, joita haluamme. [1.] Seuraavaksi käyn läpi useimmiten käytettyjä tekniikoita peleissä ja niiden hyviä ja huonoja puolia.

3.1 Yksinkertainen päätöksenteko

Yksinkertaista päätöksentekoa voidaan kutsua myös ”skriptatuksi tekoälyksi”. Tämä tekniikka on erittäin vanha, ja voidaan jopa sanoa, että se oli ensimmäinen pelitekoälyn päätöksenteon muoto. 1990-luvulla suurin osa peleistä oli ohjelmoitu tällä tekniikalla. Syy tähän on se, että se on erittäin helppo toteuttaa ja sen lisäksi se ei ole vaativa suoritukseltaan tietokoneelle. Tämä tekniikka on yleensä käytössä, jos projektin koko on pieni ja ohjelmoijia on vain yksi tai kaksi. Yksinkertainen päätöksenteko kuitenkin nopeasti hylättiin, kun tiimien koko alkoi kasvamaan ja tarvittiin paljon monimutkaisempaa päätöksentekoa. [2.]

Voidaan myös sanoa, että yksinkertainen päätöksenteko ei varsinaisesti edes ole tekniikka. Halusin kuitenkin aloittaa sillä, koska se on yleisin muoto, miten tekoäly toteutetaan esimerkiksi kouluprojekteissa. Syy tähän tietenkin on se, että yksinkertainen päätöksenteko on erittäin helppo toteuttaa. Usein kouluprojektit eivät sisällä asioita, joihin tarvitsisi monimutkaisia tekoälytekniikoita, jonka takia tämä tekniikka sopii siihen käyttöön.

Yksinkertaisella päätöksenteolla tarkoitan tässä tapauksessa ”kovakoodattuja” ehtoja. Tämä tarkoittaa sitä, että kehittäjä kirjoittaa suoraan koodiin: jos näin tapahtuu, tee tämä. Näitä ehtoja on usein vain muutama, koska jos ehtoja tulee useita, alkavat yksinkertaisen päätöksenteon huonot puolet tulemaan esille. Yksinkertaista päätöksentekoa tulisi käyttää vain, niin kuin sen nimi kertoo, hyvin yksinkertaisen tekoälyn tekemisessä. Jos tällä tavalla rupeaisi monimutkaista tekoälyä toteuttamaan, niin koodin luettavuus alkaisi kärsimään ja sen muokkaaminen jälkeenpäin olisi hyvin kömpelöä. Yksinkertainen päätöksenteko sopisi aikaisemmassa kuvassa täysin määriteltyyn tekoälyyn.

Tässä on koodiesimerkki siitä, miten yksinkertainen päätöksenteko toimisi:

```
If this happens
  Do this
else if this happens
  Do this
```

Tämä päätöksenteko on niin yksinkertainen, että kuka tahansa pystyisi sen luomaan, vaikka ei olisikaan erikoistunut tekoälyyn. Tästä peliesimerkkinä voisi olla moni kouluprojekteista, joita monet ovat tehneet. Yksi tällainen voisi olla esimerkiksi ensimmäinen kouluprojektini.

Kouluprojekti oli hyvin yksinkertainen, koska se toteutettiin ensimmäisellä lukukaudella. Tässä vaiheessa en osannut edes ohjelmoida kovin hyvin. Siinä pelissä tekoäly käytti yksinkertaista päätöksentekoa niin, että viholliset matkustavat kohti pelaajaa niin kauan, kunnes he olivat aivan pelaajan vieressä. Tällöin heidän ensimmäinen ehtonsa ei enää toteutunut, joten he vaihtoivat seuraavaan toimintoon. Seuraavassa toiminnossa he löivät pelaajaa niin kauan, kuin pelaaja pysyi heidän lähellänsä. Jos pelaajaa ei ole enää vihollisen lähellä, niin tekoäly vaihtaa takaisin matkustamiseen pelaajaa kohti. Tekniikka, josta seuraavaksi kerron, on myös yksinkertainen, mutta paljon parempi kuin tämä, joten se on suositellumpi tapa toteuttaa ei niin monimutkainen päätöksenteko.

3.2 Äärelliset tilakoneet

Pelien äärelliset tilakoneet (Finite State Machine) ovat erittäin suosittu päätöksentekotekniikka peleissä. Tämä on usein seuraava askel yksinkertaisen päätöksenteon jälkeen, jos haluaa tehdä tekoälylle monimutkaisempaa käyttäytymistä. Tilakoneet muodostuvat erilaisista ”tiloista”. Jokaiselle tilalle on määritelty käyttäytyminen eli miten hahmo käyttäytyy, kun se on tietyssä tilassa. Hahmo jatkaa samaa käyttäytymistä niin kauan, kunnes se siirtyy uuteen tilaan. Nämä tilat on yhdistetty toisiinsa ”siirtymillä”, joiden avulla voidaan siirtyä toisesta tilasta toiseen. Näillä siirtymillä on ehtoja, jotka pitää täyttyä, jotta tämä siirtymä voi tapahtua. Pelin koodissa tarkkaillaan tämänhetkisen tilan eri siirtymiä, ja jos jonkin siirtymän kaikki ehdot toteutuvat, siirrymme siihen tilaan. Tämä tarkoittaa sitä, että yhdestä tilasta voi siirtyä vain niihin tiloihin, jotka on yhdistetty ”siirtymällä” nykyiseen tilaan. Siirtymät ovat yksisuuntaisia, eli esimerkiksi uudesta tilasta ei välttämättä voi siirtyä edelliseen tilaan, jos sitä ei ole yhdistetty siirtymällä. [2.]

Kaikki tilakoneet toimivat tällä perusidealla, mutta niiden toteutus voi erota huomattavasti. Suosituin tapa toteuttaa tilakone aloittelijoilla on ”kovakoodattu äärellinen tilakone”. Tämä usein johtuu siitä, että se on toteutus, joka opetetaan ensimmäisenä tilakoneista puhuttaessa. [2.]

Alla oleva kaavio on esimerkki tästä.

```
tämänhetkinenTila
if tämänhetkinenTila == TOIMETON
        if näenPelaajan(): tämänhetkinenTila = JUOKSEE
        if not näenPelaajan(): tämänhetkinenTila = NUKU
else if tämänhetkinenTila == JUOKSEE
        if not näenPelaajan(): tämänhetkinenTila = TOIMETON
else if tämänhetkinenTila == NUKU
        if näenPelaajan(): tämänhetkinenTila = JUOKSEE
```

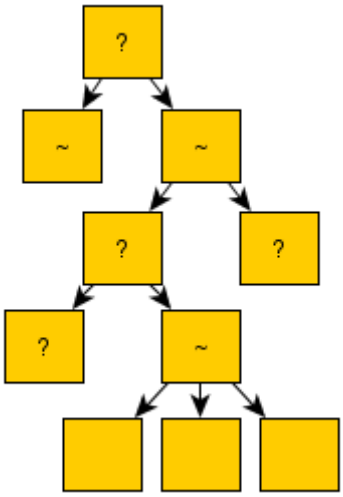
Tässä toteutuksessa kaikki säännöt ja toiminnot on suoraan kirjoitettu koodiin. Tällainen toteutus on yksinkertainen ja siten helppo toteuttaa. Tästä kuitenkin seuraa ongelmia esimerkiksi, jos ehtojen tai tilojen lukumäärä kasvaa suureksi. Tämä johtaa epäselvään ja vaikeasti hallittavaan koodiin. Yksi ratkaisu tähän ongelmaan on toteuttaa tilakone rajapintojen avulla sen sijaan, että koodaisi kaikki ehdot ja toiminnot yhteen paikkaan. [2.] Perusidealtaan se toimii samalla lailla kuin normaali tilakone, mutta on selkeämpi ja siten helpompi ylläpitää.

Äärelliset tilakoneet ovat erittäin hyvä vaihtoehto, jos peli tarvitsee vain yksinkertaista tekoälyä. Se on helppo toteuttaa ja ymmärtää. Esimerkiksi vuonna 1980 tullut peli Pac-Man toimii tilakoneiden avulla. Se on hyvin yksinkertainen peli, joka käyttää kolmea eri tilaa sen tilakoneessa. Nämä tilat ovat: jahtaa (chase), hajaannu (scatter) ja pelästynyt (frightened). Jahtaa tilassa hahmot jahtaavat pelaajaa tietyllä tavalla, joka niille on määritelty. Tämä on hahmojen perustila. Hajaannu tilassa hahmot lopettavat jahtaamisen muutamaksi sekunniksi, ja sen sijaan liikkuvat kohti heille määrättyä aluetta. Tämän tilan jälkeen hahmot siirtyvät takaisin jahtaa tilaan. Pelästynyt tila tapahtuu silloin, kun pelaaja on syönyt yhden erikoispisteen, jonka avulla hän pystyy syömään vihollishahmoja. Tässä tilassa viholliset liikkuvat satunnaisesti eri suuntiin, kunnes pelaajan kyky syödä heidät katoaa. [3.]

Valitettavasti pelkästään tilakoneiden käyttö tekoälyn päätöksenteon toteuttamiseen ei sovi suurimpaan osaan peleistä. Syynä tähän on se, että tilojen määrän kasvaessa, vaikka kuinka hyvin sen toteutuksen tekisi, alkaa tilakoneen ylläpitäminen vaikeutua huomattavasti. Suurissa peleissä, joissa on hahmoja, jotka tarvitsevat monimutkaista päätöksentekoa samanaikaisesti, olisi pelkän äärellisen tilakoneen käyttö hyvin kömpelöä. [2.]

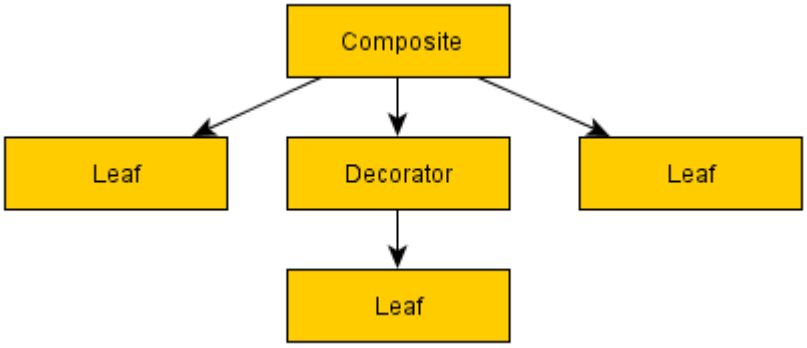
3.3 Käytöspuu

Jos tilakoneet ovat suosittu tekniikka yksinkertaisen tekoälyn tekemisessä, niin käytöspuut (Behavior trees) ovat niiden vastike hieman monimutkaisemman tekoälyn tekemisessä. Voidaan sanoa, että käytöspuu on yhdistelmä aikaisemmin tulleista tekoälytekniikoista, kuten tilakoneista. Niiden vahvuus on se, että ne ovat yksinkertaisia ymmärtää ja tehdä. [2.] Käytöspuut koostuvat niin sanotuista ”soluista” (node), jotka ovat hieman samanlaisia kuin tilakoneiden tilat. Nämä solut siten muodostavat hierarkkisen ”puun”. [4.] Tämän puun rakenteen voi nähdä kuvassa 2.



Kuva 2. Hierarkkinen puu [4]

Hierarkkisen puun perusidea on se, että se aloittaa ylimmäisestä solusta ja jatkaa matkaa alaspäin, kunnes se saavuttaa viimeisen solun puussa. Puuta luetaan vasemmalta oikealle, ylhäältä alaspäin. Jokaisella solulla on yleensä kolme eri tilaa: onnistunut (success), epäonnistunut (failure) ja suorittumassa (running). Tämä hierarkkinen puu koostuu erilaisista soluista, joita kutsutaan lehti- (leaf), yhdistelmä- (composite) ja koristelija- (decorator) soluiksi. Nämä solut näkyvät kuvassa 3.

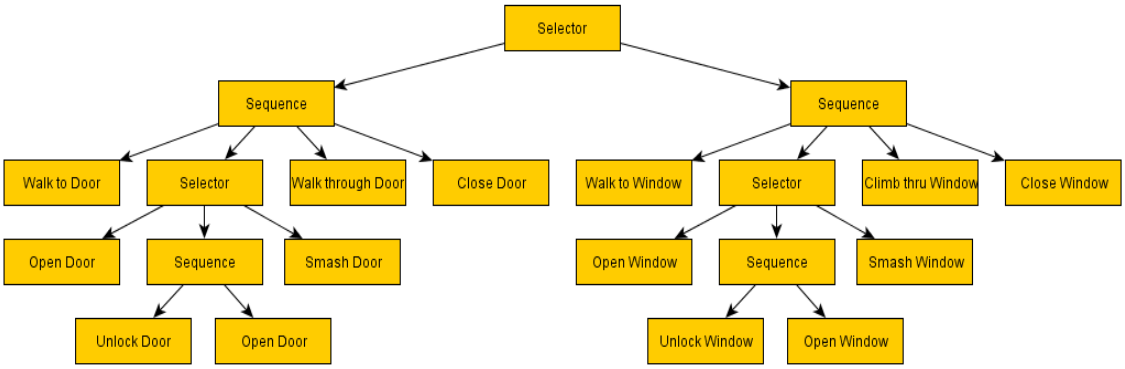


Kuva 3. Erityyppiset solut [4]

Yhdistelmäsolu voi koostua yhdestä tai useammasta lapsisolusta. Esimerkiksi kuvassa 3 yhdistelmäsolulla on kolme lapsisolua. Yhdistelmäsolu käsittelee sen lapsisolut jossakin tietyssä järjestyksessä, joka on asetettu. Sen jälkeen, kun se on käynyt kaikki lapsisolunsa läpi, yhdistelmäsolu palauttaa sen hallitsevalle eli vanhempisolulle (parent node) tilansa, eli onnistunut tai epäonnistunut. Yhdistelmäsolut voidaan vielä jakaa sarja- (sequence) ja valitsija- (selector) soluihin. Sarjasolu käy järjestyksessä sen lapsensa läpi palauttaen onnistuneen tilan sen vanhemmalle vain, jos sen kaikki lapset onnistuvat. Valitsijasolu sen sijaan tarvitsee vain yhden onnistuneen lapsen, jotta se palauttaa onnistuneen tilan sen vanhemmalle. [4.]

Toisin kuin yhdistelmäsolulla, koristelijasolulla voi olla vain yksi lapsi. Koristelijasolun tarkoitus on muokata lapsensa käyttäytymistä. Esimerkiksi koristelijasolun lapsi voisi palauttaa sille tilan: ”onnistunut”, mutta sen sijaan, että koristelijasolu palauttaisi tämän tilan omalle vanhemmalle, niin kuin yhdistelmäsolu tekisi tässä tapauksessa, se voisikin kääntää tilan toisinpäin ja palauttaa ”epäonnistunut” -tilan. [4.]

Lehtisolut ovat kaikista tärkein solutyyppi käytöspuissa. Niillä ei voi olla lapsisoluja. Lehtisolut ovat niitä soluja, jotka oikeasti tekevät erilaisia toimintoja. Tällaisia toimintoja voisi olla esimerkiksi: kävele tiettyyn paikkaan, avaa ovi tai ammu jotakin kohdetta. [4.] Yhdistelmä- ja koristelijasoluja voisi siis kutsua ehdoiksi ja lehtisolut ovat itse toimintoja, jotka toteutuvat, kun tietyt ehdot joko täyttyvät tai eivät täyty. Seuraavaksi kuva käytöspuusta, joka on täytetty erilaisilla soluilla.

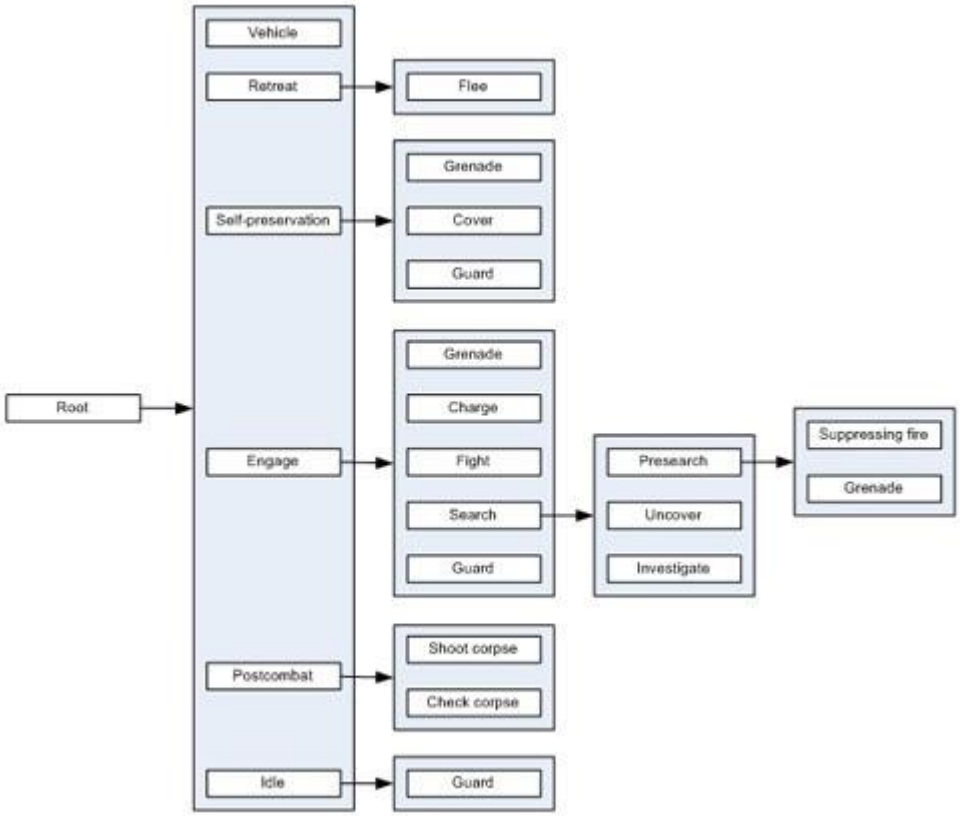


Kuva 4. Käytöspuu täytetty eri soluilla [4]

Kuvassa 4 voi nähdä, kuinka helposti käytöspuilla voi luoda monimutkaisen tekoälyn hahmolle. Puurakenteen ansiosta se on myös helppolukuinen. Kuvassa oleva käytöspuu alkaa ylimmäisestä valitsijasolusta ja sen jälkeen jatkaa vasemmalle sarjasoluun. Tämä tarkoittaa sitä, että kaikki pitää toteutua, jotta tämä sarja onnistuisi. Sarja alkaa ensimmäisestä vasemmalla olevasta solusta eli ”Walk to Door” -lehtisolusta. Jos tämä onnistuu, niin sen jälkeen siirrytään ”Walk to Door” -solun oikealla puolella olevaan valitsijasoluun. Valitsijasolu tarvitsee vain yhden sen lapsen onnistuvan, joten ensimmäiseksi se yrittää ”Open Door” -solua. Jos tämä ei onnistu, niin se yrittää sarjasolua, joka koostuu lehtisoluista: ”Unlock Door” ja ”Open Door”. Jos tämä sarja ei onnistu, niin käytöspuu yrittää vielä viimeiseksi ”Smash Door” -solua. Jos tämäkään ei onnistu, niin koko vasemmanpuolinen haara käytöspuusta epäonnistuu ja joudutaan siirtymään oikeanpuoleiseen haaraan. Tämä on käytöspuun perusrakenne.

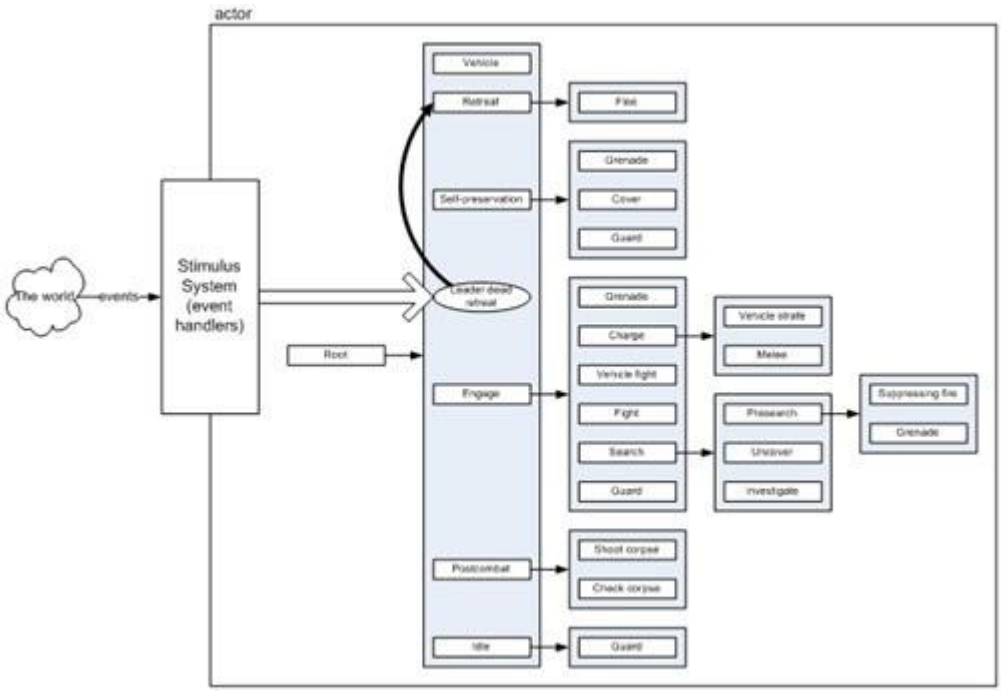
Käytöspuilla on myös huonoja puolia, jonka takia se ei välttämättä sovellu kaikkiin peleihin, esimerkiksi jos hahmon pitäisi keskeyttää sen nykyinen tekeminen ja sen sijaan aloittaa jonkin muun toiminnon tekeminen. Tämmöisen käytöksen luominen käytöspuilla ei ole mahdotonta, mutta se voi tehdä niiden käytöstä kömpelöä. [2.]

Kun puhutaan peleistä, jotka käyttävät käytöspuita, ei voi olla mainitsematta peliä, joka teki käytöspuista yhden suosituimmista tekniikoista pelitekoälyssä. Bungien vuonna 2004 julkaisema peli Halo 2 tunnetaan käytöspuiden pioneerina pelitekoälyssä. Halo 2:sta luodessa Bungie tarvitsi tekoälytekniikkaa, joka pystyisi luomaan samanaikaisesti kymmenille hahmoille monimutkaista käyttäytymistä, jota ei pystyisi saavuttamaan esimerkiksi tavallisella tilakoneella. Tämän ratkaisemiseksi he päätyivät kehittämään ratkaisun, joka nykyään tunnetaan nimellä käytöspuu. Heidän käytöspuunsa ottavat esimerkkiä hierarkkisista äärellisistä tilakoneista. [5.] Seuraavassa kuvassa on Halo 2:n käytöspuu.



Kuva 5. Käytöspuu Halo 2:sta [5]

Halo 2:n käytöspuut toimivat suurin piirtein aiemmin esitetyllä tavalla, mutta niissä on muutamia eroja normaaleihin käytöspuihin. Kuvasta 5 voi nähdä, että heti alkuun Halo 2:n käytöspuussa on valitsijasoluja. Niitä ovat esimerkiksi ajoneuvossa oleminen (Vehicle), perääntyminen (Retreat), hyökkäys (Engage), taistelunjälkeinentila (Postcombat) ja normaalitila (Idle). Esimerkiksi jos hahmo on ajoneuvossa, niin tämä käytöspuu valitsee ajoneuvosolun, joka menee täysin uuteen lapsikäytöspuuhun, jossa on ajoneuvolle tarkoitettu käyttäytyminen. Tällainen rajaaminen yhä tarkempaan käyttäytymiseen helpottaa käytöspuun suunnittelua ja ymmärtämistä. Seuraavassa kuvassa on Halo 2:n niin sanottu virikejärjestelmä.



Kuva 6. Virikejärjestelmä Halo 2:ssa [5]

Tällaisella rajaamisella on kuitenkin ongelmia. Mitä jos haluamme, että jokin tietty käyttäytyminen tapahtuu juuri tietyssä tilanteessa, esimerkiksi normaalissa tilanteessa vihollishahmo ei yritä nousta ajoneuvoon, jos pelaaja on lähellä. Entäpä jos pelaaja onkin ajoneuvossa? Tässä tilanteessa haluamme, että vihollinen meneekin ajoneuvoon. Tietenkin voi testata pelaajan tilaa ja tarkemmin sitä, onko hän jo ajoneuvossa heti käytöspuun alussa. Ongelma tässä on se, että tällaisten ehtojen lisääminen itse puuhun tekisi siitä aivan liian suuren. Näitä ehtoja pitäisi testata koko ajan, vaikka se ei olisikaan tarpeellista. [5.]

Tämän ongelman ratkaisemiseksi Bungie käytti ”virikejärjestelmää”. Kuten kuvassa 6 voi nähdä, virikkeet lisätään tiettyyn kohtaan puussa, jos jokin ”tapahtuma” tapahtuu. Esimerkiksi jos vihollisten johtaja kuolee, se lisää ”johtaja on kuollut” -virikkeen puuhun, mikä mahdollistaa puun menemisen pakenemiskäyttäytymiseen, vaikka se ei olisi normaalisti mahdollista tässä tilanteessa. Tämä virike poistuu puusta tietyn ajan kuluessa, jonka jälkeen puu palautuu normaaliin tilanteeseen. Syy siihen, että emme vain pakota hahmoja tähän käyttäytymiseen, vaan sen sijaan lisäämme sen puuhun ja annamme hahmon itse päättää mitä se tekee, on se, että tietyssä tilanteessa emme halua, että hahmo pakenee, vaikka heidän johtajansa kuolisi. [5.] Tämän järjestelmän ansiosta hahmoille on helppo luoda monimutkaista käyttäytymistä ilman, että puun rakenne muuttuu lukukelvottomaksi.

3.4 Tavoitteellinen tehtäväsuunnittelija

Suunnittelijat (planners) poikkeavat edellä mainituista tekniikoista siinä, että suunnittelijat eivät reagoi tapahtumiin niin kuin tilakoneet tai käytöspuut, vaan sen sijaan ne valitsevat tietyt toimet, jotka johtavat tiettyyn lopputulokseen [6]. Suunnittelijoita on jo kauan käytetty akateemisessa tekoälyssä, mutta niitä aloitettiin soveltamaan peleissä vasta ensimmäistä kertaa tavoitteellisen tehtäväsuunnittelijan (goal oriented action planning) avulla [7].

Tavoitteellinen tehtäväsuunnittelija muodostuu tehtävistä (actions) ja niiden edellytyksistä (preconditions) ja viimeisenä tavoitteesta (goal). Tavoitteellisen tehtäväsuunnittelijan tekoälyhahmolla on tavoitteita, joita se yrittää saavuttaa tietyin keinoin. Tämä saavutetaan siten, että suunnittelija luo sarjan tehtävistä, joita tarvitaan tavoitteen täyttämiseksi. Jokaisella sarjassa olevalla tehtävällä voi olla omia edellytyksiä, joita se tarvitsee toteutuakseen. [7.]

Otetaan esimerkiksi hahmo, jolla on tavoite hyökätä pelaajan kimppuun. Tällöin suunnittelija alkaisi muodostamaan suunnitelmaa siitä, miten tämä tavoite parhaiten tavoitettaisiin. Tehtävällä ”hyökätä ” on edellytys ”omista ase”. Jos hahmolla ei ole asetta, sen sijaan, että hahmo hylkäisi tämän tavoitteen kokonaan, se luokin uuden suunnitelman ja lisää tehtävän ”hanki ase” siihen. Tässä tehtävässä on taas edellytys, että hahmo on lähellä poimittavaa asetta, joten suunnittelija lisää tehtävän ”liiku aseen luokse”. Jos kaikki edellytykset ovat nyt täyttyneet, voi suunnittelija toteuttaa tämän suunnitelman. [7.]

Tämä tarkoittaa sitä, että suunnittelija yrittää aina ensimmäiseksi kaikkein optimaalisinta suunnitelmaa, mutta sitä ei tietenkään pysty aina toteuttamaan. Suunnittelija siinä tapauksessa kokeilee seuraavaksi parhainta suunnitelmaa, ja tämä jatkuu niin kauan, kunnes se löytää sellaisen suunnitelman, jonka se voi toteuttaa. Mikä tekee tavoitteellisesta tehtäväsuunnittelijasta hyvän, on se, että tehtävillä ja tavoitteilla ei ole mitään, joka yhdistäisi ne toisiinsa. Tämä on toisin kuin tilakoneissa ja käytöspuissa, ja tämän takia kehittäjän ei tarvitse tietää, koska tai miten hahmo tekee jotakin, vaan sen sijaan se osaa päättää itse asioista. [7.]

Peliesimerkkinä tavoitteellisesta tehtäväsuunnittelijasta ei voi ottaa mitään muuta kuin sen alkuperäisen käyttäjän: F.E.A.R.: First Encounter Assault Recon. F.E.A.R. on vuonna 2005 Monolith Productionin tekemä peli ja siten ensimmäinen peli, joka käytti nykyään tunnettua tavoitteellista tehtäväsuunnittelijaa. F.E.A.R.:in tavoite oli luoda älykäs tekoäly, mutta tämä olisi erittäin työläs tehdä pelkällä äärellisellä tilakoneella, joten he joutuivat miettimään uutta ratkaisua. Tämä johti heidät suunnittelijoihin ja siihen, miten niitä voisi käyttää pelitekoälyssä. [8.]

Voidaan sanoa, että F.E.A.R.:issa on tilakone, jossa on kaksi tilaa: ”Mene” (Go) ja ”Animoi” (Animate). Kaikki toiminnot toimivat animaatioiden avulla. Esimerkiksi ”Mene” tarkoittaa, että mene johonkin paikkaan ja tee animaatio, kun taas ”Animoi” tarkoittaa, että tee animaatio nyt tässä paikassa. Suunnittelija siten päättää, missä tilassa kunakin hetkenä ollaan. [8.] Seuraavaksi kuva F.E.A.R.:in tehtävistä.

Kuva 7. F.E.A.R.:in eri hahmotyyppien mahdollisista tehtävistä [8]

Kuvassa 7 voi nähdä, millaisia erilaisia tehtäviä on annettu valittavaksi F.E.A.R.:in tekoälyhahmoille. Esimerkiksi ”Rotta” -hahmolla on neljä eri tehtävää, mistä se voi suunnitella oman suunnitelmansa toteuttaakseen tavoitteensa, kun taas ”Sotilas”- tai ”Palkkamurhaaja” -hahmoilla on paljon enemmän valinnan varaa.

3.5 Hierarkkiset tehtäväverkostot

Hierarkkinen tehtäväverkosto (hierarchical task networks), kuten tavoitteellinen tehtäväsuunnittelija, on suunnittelija. Mikä tekee hierarkkisesta tehtäväverkostosta erilaisen, on se, että sen ”tehtävät” voidaan jakaa kahteen eri tyyppiin. Nämä tyypit ovat primitiiviset tehtävät (primitive tasks) ja yhdistetehtävät (compound tasks). [1.]

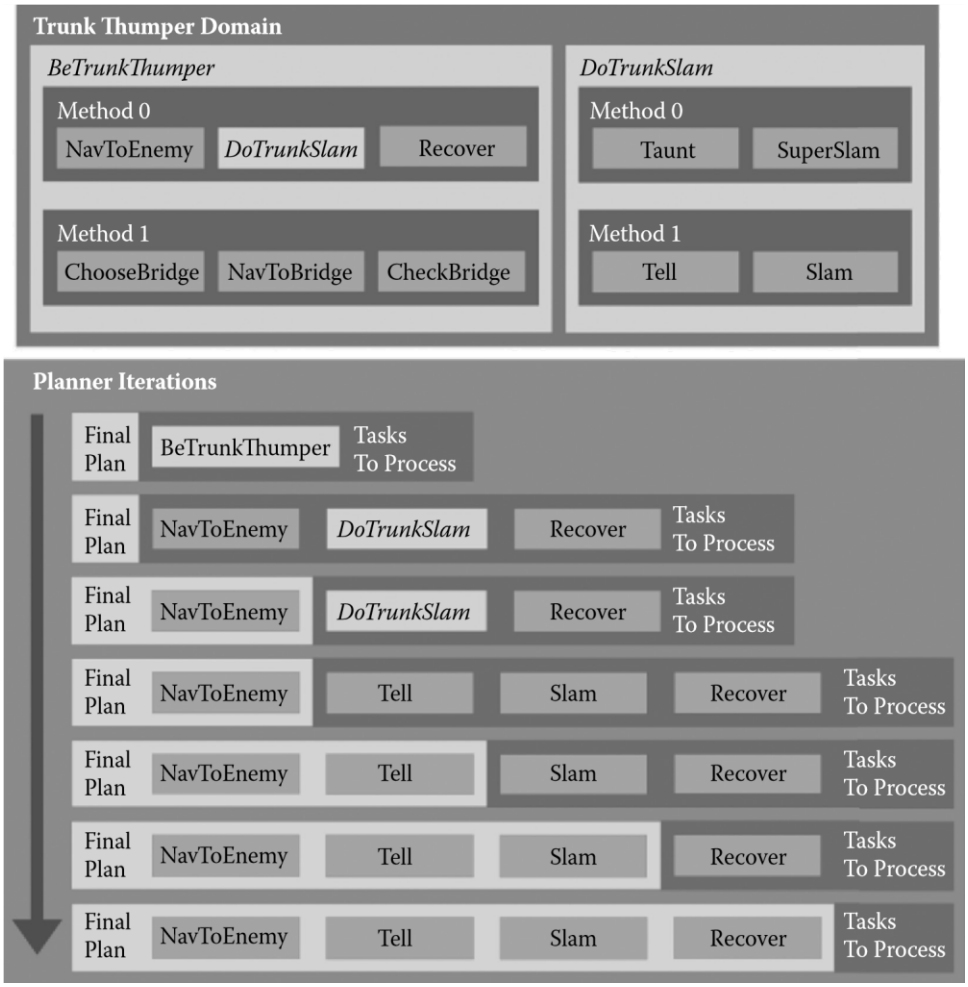
Primitiiviset tehtävät ovat verrattavissa tavoitteellisen tehtäväsuunnittelijan tehtäviin, kun taas yhdiste tehtävät ovat tehtäviä, jotka muodostuvat useammasta pienemmästä tehtävästä ja ne tehtävät voivat vielä muodostua pienemmistä tehtävistä ja niin edelleen. Tästä syystä tätä tekniikka kutsutaan hierarkkiseksi. Yhdistetehtävät yksinään eivät tee mitään, vaan kaikki toiminnallisuus on primitiiveissä tehtävissä. [1.]

Esimerkiksi primitiivinen tehtävä voisi olla ”kävele tähän paikkaan”, kun taas yhdistetehtävä voisi olla ”hyökkää pelaajan kimppuun”. Tässä tapauksessa primitiivisen tehtävän voisi toteuttaa vain yhdellä tavalla eli kävellä, mutta yhdistetehtävä on paljon abstraktimpi. Voit toteuttaa sen monella eri tavalla ja voit jopa tarvita ”kävele tähän paikkaan” -primitiivistä tehtävää sen toteuttamiseksi. [1.]

Mikä tekee hierarkkisista tehtäväverkostoista houkuttelevamman vaihtoehdon tavoitteellisen tehtäväsuunnittelijan sijaan, on sen parempi suorituskyky. Hierarkkiset tehtäväverkostot käyttävät syvyyssuuntaista hakua suunnitelmia tehdessään toisin, kuin tavoitteellinen tehtäväsuunnittelija. Hierarkkisen rakenteen takia syvyyssuuntaisen haun ei tarvitse mennä läpi kaikkia tehtäviä, vaan vain ne, jotka kuuluvat siihen yhdistetehtävään. Tämä nopeuttaa suunnitelmien tekoa huomattavasti. [1.]

Hierarkkisia tehtäväverkostoja käytettiin ensimmäisen kerran vuonna 2012 Transformers: Fall of Cybertron -pelissä. High Moon, Fall of Cybertronin kehittäjä, käytti edellisessä pelissään Transformers: War for Cybertron tavoitteellista tehtäväsuunnittelijaa. He kuitenkin huomasivat ongelmia tätä peliä kehittäessään ja päättivät luoda sen pohjalta uuden järjestelmän, joka toimisi paremmin heille. [9.]

Suurimmista ongelmista yksi oli se, että heillä oli paljon erilaisia hahmoja, jotka tarvitsivat hieman erilaista käyttäytymistä. Tavoitteellisessa tehtäväsuunnittelijassa ei voi muuttaa tehtävien suoritushintoja, jonka perusteella suunnittelija valitsee suunnitelmia erilaisiksi eri hahmoille. Tämän lisäksi tavoitteellinen tehtäväsuunnittelija antaa kaiken vallan itse hahmolle päättää siitä mitä se tekee. Tämä luo konfliktin sen kanssa, että kehittäjät haluavat hallita tekoälynsä käyttäytymistä. [9.] Seuraavaksi esimerkkikuva tehtäväverkoston mahdollisesta rakenteesta.



Kuva 8. Hierarkkisen tehtäväverkoston suunnitelmarakenne [1]

Näiden ongelmien ratkaisemiseksi he kehittävät hierarkkisen tehtäväverkoston. Tässä ratkaisussa on kehittäjän vastuulla luoda yhdistetehtäviä, jotka ovat kokonaisia suunniteltuja käyttäytymisiä sen sijaan, että tekoäly yrittäisi toteuttaa jotakin tavoitetta. [9.] Kuvasta 8 voi nähdä, kuinka tämä rakenne eroaa huomattavasti tavoitteellisesta tehtäväsuunnittelijasta. Esimerkiksi voi olla käyttäytyminen ”BeTrunkThumper”, joka koostuu kahdesta eri tavasta toteuttaa se. Nämä toteutustavat koostuvat taas pienemmistä tehtävistä, joilla voi olla itsellään useampia toteutustapoja, esimerkiksi ”DoTrunkSlam”.

3.6 Ohjaajajärjestelmä

Ohjaajajärjestelmä on hieman erilainen kaikista edellä mainituista tekniikoista. Se ei välttämättä edes ole oikeastaan ”tekniikka”. Ohjaajajärjestelmän idea on se, että ohjataan pelin kulkua tekoälyllä sen sijaan, että ohjattaisiin yksittäistä tekoälyhahmoa. Ohjaajajärjestelmän nimi tulee Valven julkaisemasta pelistä nimeltä Left 4 Dead, jossa tätä tekniikkaa käytettiin. [10.]

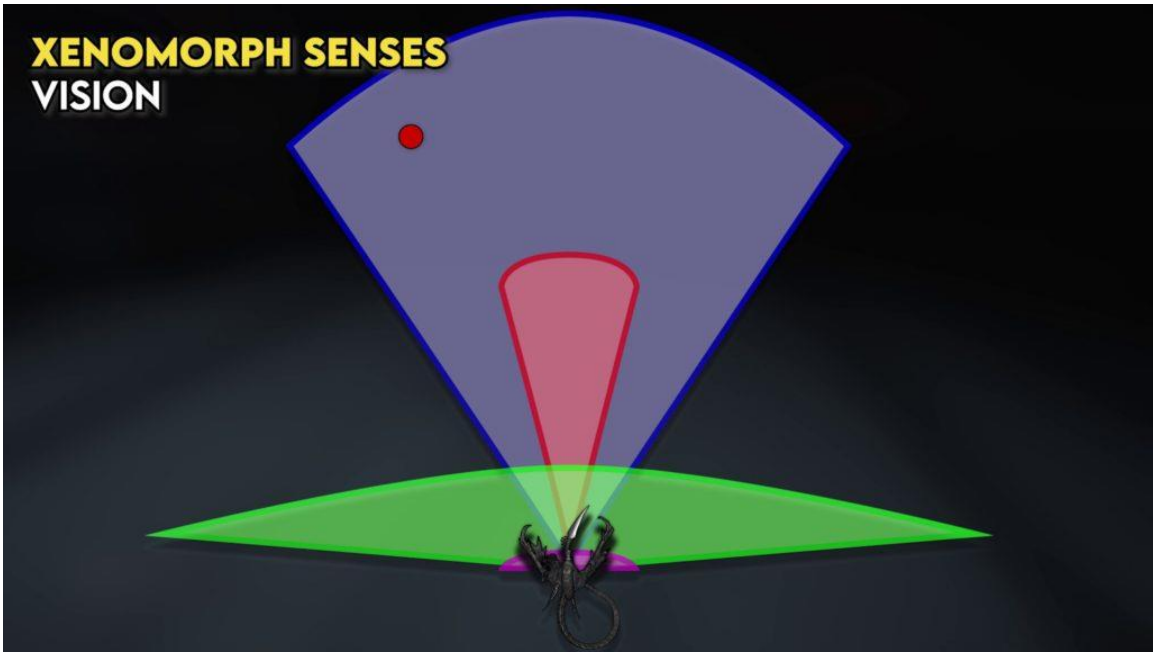
Ohjaajajärjestelmä tekee päätöksiä sen perusteella, mitä pelaaja tekee sillä hetkellä. Ajatuksena on, että vaikeustaso riippuu pelaajan aktiivisuudesta. Kehittäjät haluavat, että peli on tarpeeksi vaikea, jotta se on mielenkiintoinen, mutta ei liian vaikea, etteivät pelaajat lopeta kesken. Ohjaajajärjestelmän tehtävä on tarkkailla, kuinka hyvin pelaajat pelaavat ja siten ohjata pelin vaikeutta. Pelaajien taitoa tarkkaillaan käyttämällä erilaista statistiikkaa, esimerkiksi tappoja, vihollisiin vahingon tekemisen määrää, ampumatarkkuutta ja mahdollisesti paljon muita. Left 4 Dead -peli myös tarkkailee pelaajien ”stressitasoa”. Tämä yrittää simuloida pelaajien tämänhetkistä intensiteettiä. Tätä stressitasoa nostaa esimerkiksi lähellä olevat viholliset, jotka ovat hyökkäämässä. Tämän stressitason avulla tekoäly voi tehdä päätöksiä siitä, mihinkä pelaajaan hyökätään seuraavaksi, tai jos stressitasot ovat liian korkealla, niin kannattaa pitää hengähdystauko pelaajille. [10.]

Ohjaajajärjestelmä voidaan jakaa kolmeen eri vaiheeseen. Nämä vaiheet ovat ”Kasvu-”, ”Huippu-” ja ”Rentoutumis-”vaihe. Kasvuvaiheen aikana ohjaajan tarkoitus on kasvattaa pelaajien stressitasoa kasvattamalla pelin vaikeutta vähitellen. Kun pelaajien stressitaso on noussut asetettuun maksimikorkeuteen, päästään huippuvaiheeseen. Tällöin ohjaajan tehtävä on rauhoittaa tilanne. Viimeiseksi on rentoutumisvaihe, jolloin pelaajille annetaan lyhyt aika valmistautua seuraavaan taisteluun. Nämä kolme vaihetta muodostavat ohjaajajärjestelmän ytimen. [10.]

Tällainen järjestelmä tekee pelistä dynaamisen ja varmistaa sen, että kaksi pelikertaa ovat aina erilaiset. Eli tällaisilla peleillä on mahdollisesti paljon uudelleen pelaamisen mahdollisuutta. Tietenkään tällainen järjestelmä ei varmastikaan toimisi kaikenlaisissa peleissä, varsinkaan jos haluaa hyvin säännöllisen tekoälyn.

Vaikka aikaisemmin mainitsin Left 4 Dead pelin tästä tekniikasta, en kuitenkaan ota sitä peliesimerkiksi, vaan sen sijaan kunnian saa Creative Assemblyn vuonna 2014 julkaisema kauhupeli Alien: Isolation. Alien Isolation, kuten Left 4 Dead, käyttää samankaltaista ohjaajajärjestelmää. Pelin idea on selviytyä yksin avaruusasemalla, jossa Xenomorph-otus asustaa. Tämä otus on melkein voittamaton, jolle pelaaja ei voi suoraan tehdä mitään. [11.]

Mikä tekee tästä mielenkiintoisen, on se, että tässä pelissä vihollinen pystyy tappamaan sinut yhdestä iskusta. Normaalisti tällainen tekoäly olisi erittäin turhauttava ja liian vaikea, mutta Creative Assembly pystyi luomaan tyydyttävän käyttäytymisen otukselle ohjaajajärjestelmän ansiosta. Miten se toimii, on niin, että on kaksi erilaista tekoälyä. On niin sanottu ohjaajatekoäly ja otuksen tekoäly. Ohjaajatekoäly tietää koko ajan, missä pelaaja on ja mitä hän tekee. Otuksen tekoäly sen sijaan yrittää löytää pelaajan sen tämänhetkisen tiedon avulla. Ohjaajatekoäly antaa vinkkejä otuksen tekoälylle siitä, missä pelaaja on. Otuksen tekoäly ei kuitenkaan ikinä huijaa. Se ei tiedä, missä pelaaja tarkalleen on, jos se ei ole itse havainnut pelaajaa. Tämä antaa pelaajalle mahdollisuuden yllättää otus erilaisilla tempuilla. [11.] Kuvassa 9 voi nähdä otuksen näköaistialueet.



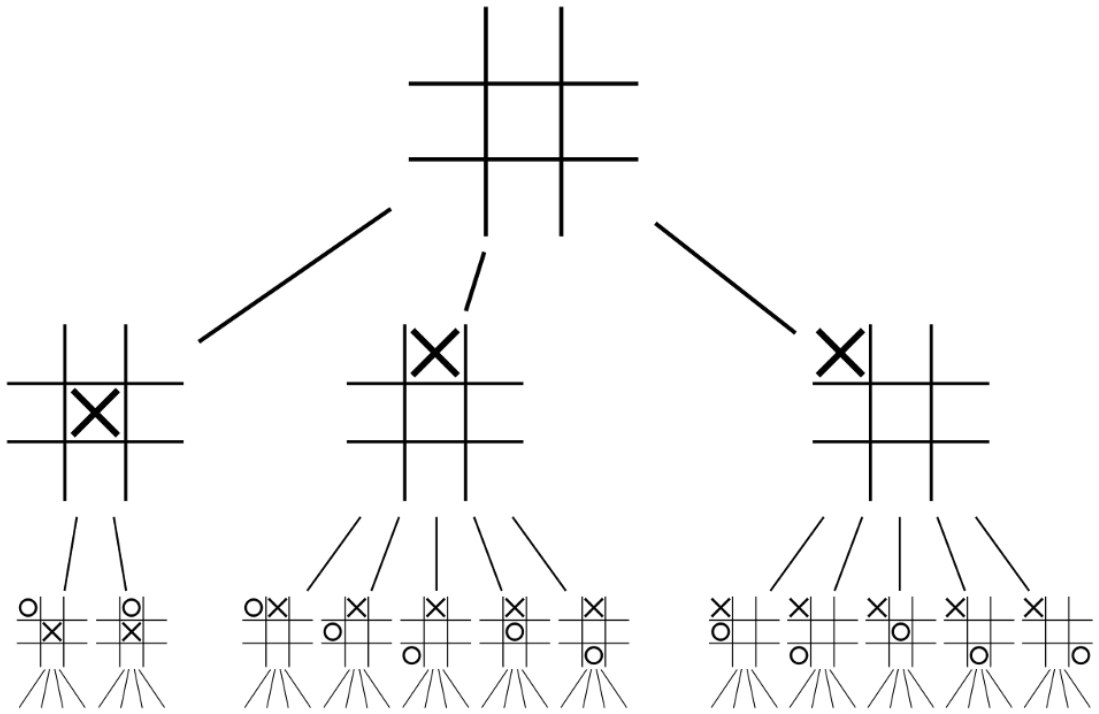
Kuva 9. Otuksen näköaistit [12]

Jos pelaaja kulkee näille alueille, niin tekoäly huomaa hänet. Se ei kuitenkaan tapahdu heti, vaan se riippuu ajasta, jonka pelaaja on alueella. Tämä aika taas riippuu siitä, millä alueella hän on. Nämä eri alueet ovat kuvassa näkyvät vihreä, sininen, violetti, ja punainen alue.[13] Otuksen oma tekoäly käyttää käytöspuita sen päätöksentekoon. Käytöspuut tässä pelissä toimivat samalla lailla kuin aikaisemmin mainittu Halo 2. Alien Isolation käyttää myös stressitasoa, kuten Left 4 Dead, jolla se säätelee pelin vaikeutta. [11.]

3.7 Monte Carlo -menetelmä

Monte Carlo -puuhaku (Monte-Carlo Tree Search) tai MCTS on algoritmi, jonka tavoitteena on löytää optimaalisin teko, mitä voi tällä hetkellä tehdä. Tämä voi kuulostaa samanlaiselta kuin aikaisemmin käydyt tekniikat, mutta se eroaa huomattavasti. Puuhakuja on monia erilaisia, mutta Monte Carlo -menetelmää on sovellettu pelien tekoälyssä toimivasti. Se on myös suhteellisen uusi menetelmä. Peleissä sitä on käytetty vain yhden vuosikymmenen ajan. [13.]

Jotta Monte Carlo -menetelmä voidaan selittää selkeästi, on hyvä ottaa esimerkiksi yksinkertainen ristinollapeli. Alla olevassa kuvassa voi nähdä ristinollapelilautoja ja Monte Carlon puurakenteen.



Kuva 10. Ristinolla Monte Carlo -menetelmällä. [13]

Monte Carlon perusidea on simuloida pelin eri mahdollisia siirtoja ja sitten valita niistä parhain. Yllä olevassa kuvassa nähdään ylimpänä aloituspiste. Sen jälkeen algoritmi aloittaa siirtonsa. Ensimmäinen vaihtoehto on laittaa risti keskelle lautaa. Tämän jälkeen algoritmi ottaa huomioon vastustajan vaihtoehdot. Niitä on kaksi, joko vasemmalle ylhäälle tai keskelle ylhäälle, ja koska lauta on symmetrinen, sen ei tarvitse laskea muita vaihtoehtoja laudalla. Tämä prosessi jatkuu, kunnes tekoäly pääsee pelin loppuun, jonka jälkeen se näkee voittajan. Jos tekoäly ei voittanut,

niin se palaa takaisin ja aloittaa seuraavan oksan puusta. Tämä jatkuu niin kauan, kunnes tekoäly voittaa tai aikaraja umpeutuu. Jos aikaraja umpeutuu, niin se valitsee sillä hetkellä sen tiedossa olevan parhaimman siirron. Mitä enemmän aikaa tälle algoritmille annetaan, sitä paremman lopputuloksen se saa. Tietenkin on maksimiaikaraja, jonka jälkeen siirto ei enää paljoa parane. [13.] Eli jos antaa algoritmille yhden minuutin vastaan kaksi minuuttia aikaa, ero niiden välillä ei välttämättä ole niin paljon kuin esimerkiksi viiden ja kahdenkymmenen sekunnin välillä.

Tämä oli hyvin yksinkertainen selitys siitä, miten tämä algoritmi toimii, mutta se on käytännössä monimutkaisempi. Esimerkiksi missä järjestyksessä se menee eri vaihtoehdot läpi, on oma matemaattinen algoritminsa.

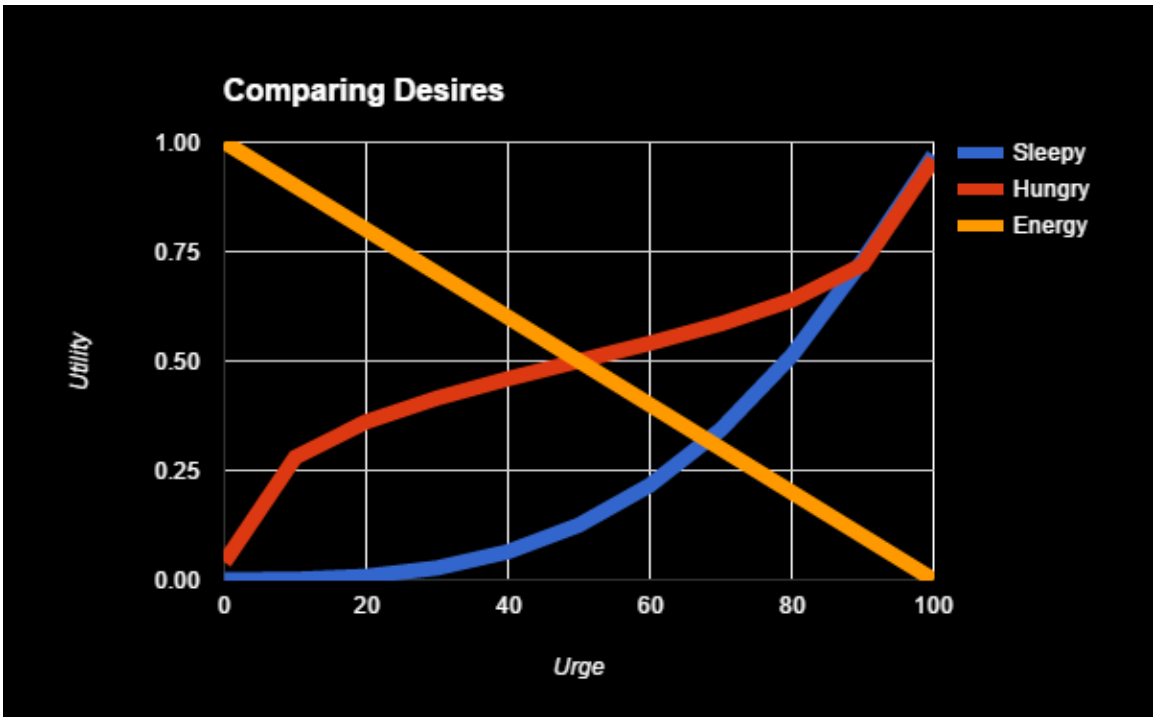
Peliesimerkiksi voidaan ottaa toinen Creative Assemblyn peli nimeltä Total War: Rome 2. Tämä on vuonna 2013 julkaistu strategiapeli ja luultavasti myös kuuluisin esimerkki, joka hyödynsi Monte Carlo menetelmää. Rome 2 on vain yksi peli osana suurta Total War sarjaa, johon oli tullut aikaisemmin jo useita pelejä. Aikaisemmat pelit sarjassa käyttivät tavoitteellista tehtäväsuunnittelijaa niiden tekoälyn päätöksenteossa. Creative Assembly ei kuitenkaan ollut tyytyväinen tähän. He tarvitsivat älykkään tekoälyn, joka pystyi hallitsemaan suurta määrää tietoa nopeasti. Tämä johti heidän päätökseensä käyttää Monte Carlo menetelmää Rome 2 -kampanjatekoälyn toteuttamiseen. Rome 2:n kampanjan tekoäly tarvitsee tehdä monia päätöksiä sen vuoron aikana. Tämän takia yksi vuoro voidaan jakaa kolmeen eri vaiheeseen. ”Ennen liikkumista-”, ”tehtävien allokointi- ” ja ”siirron jälkeinen vaihe”. Ennen liikkumista -vaiheessa tekoäly tunnistaa mahdolliset uhat, hallitsee sen eri resursseja sekä tekee mahdollisia diplomaattisia toimintoja, joko pelaajaa tai eri tekoälyjä kohti. Tehtävien allokointivaiheessa tapahtuu suurin työ Monte Carlo -menetelmälle. Tässä vaiheessa sen täytyy simuloida parhaimmat liikkumiskomennot eri hahmoille, joita tekoäly ohjaa. Tämän jälkeen siirrytään viimeiseen eli siirron jälkeiseen vaiheeseen, jolloin tekoäly käyttää resursseja rakennusten rakentamiseen tai muihin hallinnollisiin tehtäviin. [14.]

Toisin kuin ristinolla, Rome 2 on paljon monimutkaisempi peli, joten simulointi pelin loppuun saakka olisi mahdotonta. Rome 2 simuloikin vain yhden vuoron edestä eri vaihtoehtoja ja sen jälkeen valitsee niistä parhaimman. Monte Carlo menetelmällä on myös ongelmia, kuten pelin julkaisussa kävi ilmi. Monet pelaajat valittivat kampanjan tekoälyn hitaudesta. Usein vuorot kestivät useamman minuutin, josta pelaajat eivät olleet mielissään. Tämä johtui Monte Carlon simuloinnista. Creative Assembly kuitenkin sai korjattua monet ongelmat päivitysten kautta, ja myöhemmin he paransivat tätä menetelmää uusissa peleissään. [14.]

3.8 Hyötypohjainen päätöksenteko

Hyötypohjaista päätöksentekoa (Utility AI) voidaan verrata hieman tehtäväsuunnittelijaan siinä mielessä, että päätökset perustuvat eri toimintojen arvoihin. Esimerkiksi tekoälyn pitäisi valita kahden eri toiminnon välillä, joista toinen maksaa 3 € ja toinen 5 €. Tämän lisäksi ensimmäinen kestää tehdä kaksi päivää ja toinen yhden päivän. Hyötypohjainen päätöksenteko ottaa huomioon nämä ja valitsee parhaimman sille. Toisin kuin tehtäväsuunnittelija, hyötypohjaisessa ei ole minkäänlaisia tehtäviä. [15.] Ei voi sanoa, että ”mene keräämään puita ja sitten myy ne torilla, jonka jälkeen osta niillä rahoilla kalliimpi vaihtoehto kaksi”. Voidaan sanoa, että tämä päätöksentekotekniikka on sekoitus suunnittelijan ja käytöspuun kanssa.

Perusidealtaan hyötypohjaisen päätöksenteon mahdolliset toiminnot perustuvat eri tietoihin, joita tekoälyllä on käytössään. Tällaisia tietoja voisi olla elämäpisteet, raha ja ruokamäärä. Esimerkiksi elämäpisteet ovat hyvin alhaalla ja ruokaa ei ole, mutta rahaa on paljon. Tällöin tekoäly voi valita kalliimman, mutta nopeamman tavan saada ruokaa.



Kuva 11. Hyötypohjaisen päätöksenteon mahdolliset tiedot. [16]

Kuvassa 11 on eri tilat pisteytettynä käyrälle. Käyrät menevät yhdestä nollaan pystyakselilla tarkoittaen, kuinka tyydytetty se tila on. Vaaka-akselilla on taas tarve täyttää se tila, joka menee nollasta sataan. Tilat käyrälle ovat: väsynyt, nälkäinen ja energia. Jos väsynyt on 0 pystyakselilla se tarkoittaa, että pelaaja on täysin virkeä. Käyttäen näitä kolmea eri tietoa tekoäly voisi tehdä erilaisia päätöksiä varmistaakseen, että nälkäinen ja väsynyt tilat pysyvät mahdollisimman lähellä nollaa ja energia pysyy lähellä yhtä.

Peliesimerkkinä tästä tekniikasta on Guerrilla Gamesin vuonna 2009 julkaistu peli Killzone 2. Tämä peli on hyvin positiivisesti vastaanotettu pelitekoälyn kannalta. Guerilla Games päätyi tähän ratkaisuun, koska käytöspuiden kanssa on vaikea seurata satoja eri tilanteita pelissä eikä uusia asioita voida lisätä dynaamisesti pelin aikana. Tämä tekniikka on myös helposti muokattavissa, koska voi vain lisätä asioita, eikä tarvitse huolehtia yhteyksistä eri tilojen tai oksien välillä. Tämä johtaa siihen, että ei haaskata kehittämisaikaa erilaisten virheiden korjaamiseen. Hyötypohjainen päätöksenteko on myös helppo suunnittelijan näkökulmasta. Ohjelmoijan ei tarvitse selittää eri tiloja, ehtoja tai sarjoja. Voidaan vain sanoa, että jos pelaajaa ammutaan, niin hae suojaan. Tämän lauseen pystyy ymmärtämään kuka vain. [16.]

Voidaan siis sanoa, että hyötypohjainen päätöksenteko on hyvä vaihtoehto, jos käytöspuut eivät toimi hyvin. Esimerkiksi jos ne toisivat liikaa monimutkaisuutta toteutukseen. Hyötypohjaisen negatiiviset puolet ovat ne, että se tarvitsee mahdollisesti paljon testausta ja aikaa, jotta sen voi muokata juuri sellaiseksi kuin haluaa. Tämä tekniikka voi myös aiheuttaa epäselvyyttä siitä, miksi jokin tekoälyhahmo tekee jonkin tietyn toiminnon tietyssä tilanteessa. [16.]

3.9 Neuroverkot

Neuroverkot (Neural Networks) eroaa täysin käydyistä edellisistä tekniikoista. Neuroverkot ovat koneoppimistekniikka. Kaikki päätöksentekotekniikat voidaan luokitella janalle, jossa yhdessä päässä on täysin määritelty tekoäly ja toisessa päässä täysin autonomisesti päättävä tekoäly. Neuroverkot kuuluvat täysin autonomisesti päättävään tekoälyyn. Se on hyvin vanha tekniikka, jota on käytetty monia vuosikymmeniä. [1, s. 391.] Koneoppinen on tekniikka, jota harvoin käytetään peleissä. Tämä johtuu siitä, että se menee vastoin pelitekoälyn ideaa, jossa halutaan vain illuusio älykkyydestä, ei oikeaa akateemista tekoälyä. Koneoppimisella on myös hyviä puolia, jonka takia sitä voidaan käyttää tietyissä tilanteissa pelitekoälyä tehdessä. Yksi tällainen syy on se, että se pystyy paljon monimutkaisempaan päätöksentekoon kuin aiemmin esitellyt tekniikat.

Neuroverkkojen idea on luoda tekoäly, jota koulutetaan erilaisilla harjoitteluseteillä. Näitä harjoittelusettejä se käy läpi mahdollisesti tuhansia. Esimerkiksi vuonna 2010 julkaistussa Gas Powered Gamesin kehittämässä Supreme Commander 2 -strategiapelissä he laittoivat neuroverkkotekoälyn pelaamaan itseään vastaan. Tämän pelin aikana neuroverkko hallitsi molempia puolia ja teki päätöksiä keräämällä tietoa omista hahmoista sekä vihollishahmoista ja siten syötti sen tiedon neuroverkkoon. Harjoituksen aikana tekoäly ei kuitenkaan suorittanut sitä, mitä neuroverkko suositteli, vaan sen sijaan teki satunnaisia päätöksiä, joista se arvioi, kuinka hyviä ne päätökset olivat. Kaikki tämä tieto syötettiin neuroverkkoon, ja vähitellen sen päätöksenteko parani. [1, s. 394.] Neuroverkot ovat suhteellisen monimutkainen asia, joten niiden selitys kokonaan vaatisi oman työnsä.

Yksi suuri huono puoli neuroverkoissa on virheiden korjauksen hankaluus. Ei voi suoraan lähteä muuttamaan asioita. Virhe neuroverkossa tarkoittaa sitä, että jotain on vikana syötetyssä tiedossa. Esimerkiksi jos harjoittelun aikana kaikki toimii normaalisti, mutta vika ilmestyy, kun sitä testataan oikeassa tilanteessa, se tarkoittaa sitä, että harjoittelu ei ole kunnolla vastannut oikeaa tilannetta. Tässä tapauksessa se ei ole saanut tarvittua tietoa, jotta se osaisi toimia oikein. Supreme Commander 2 törmäsi tähän ongelmaan, kun harjoittelussa tekoälyn tavoite ei ollut voittaa peliä, vaan sen sijaan taistella taisteluja. Supreme Commanderissa täytyy tuhota tietty rakennus voittaakseen pelin. Kun tämä rakennus tuhoutuu, niin se aiheuttaa räjähdyksen, joka tuhoaa lähellä olevat hahmot. Tämä johti siihen, että oikeassa pelissä neuroverkko ei ikinä yrittänyt voittaa, koska se ei tiennyt voittamisesta mitään. Neuroverkko kuitenkin tiesi sen, että se menettää hahmoja tuhotessaan voittorakennuksen, jonka takia se ei tietenkään halunnut tehdä sitä. [1, s. 396.]

Peliesimerkiksi neuroverkoista voidaan ottaa Blizzardin vuonna 2010 julkaistu StarCraft 2. Tässä tapauksessa neuroverkkoja ei käytetty itse pelinpäätöksenteossa, vaan sen sijaan nykypäivänä StarCraft 2 on suosittu tapa testata neuroverkkotekoälyä. Kuuluisin esimerkki tästä on neuroverkko AlphaStar, joka on ollut paljon uutisissa viime vuosien aikana. StarCraft 2 on yksi monimutkaisimmista peleistä, joten se on mielenkiintoinen haaste tekoälylle. Parhaimmat pelaajat StarCraft 2:ssa ovat ammattilaispelaajia, jotka kilpailevat satojen tuhansien eurojen palkinnoista. AlphaStar on usein laitettu tällaisia pelaajia vastaan, ja se pystyy jopa voittamaan nämä pelaajat, jotka ovat käyttäneet kymmeniä tuhansia tunteja tullakseen parhaiksi StarCraft-pelaajiksi. [17.]

AlphaStar aloitti oppimisensa menemällä läpi ihmisten pelaamia pelejä ja oppimalla niistä erilaisia mikro- ja makrostrategioita. Pelkästään tällä AlphaStar pystyi voittamaan pelissä jo valmista Blizzardin tekemää vaikeinta tekoälyä vastaan sekä suurinta osaa pelaajista. Sen jälkeen AlphaStar laitettiin pelaamaan turnauksen erilaisia koneoppimistekoälyjä vastaan parantaakseen sitä. Tällainen turnaus kesti noin 14 päivää, jolloin se pelasi noin 200 vuoden edestä pelejä. Kun tämän jälkeen AlphaStar haastoi yhden parhaimmista StarCraft-pelaajista, se voitti viisi nolla tämän pelaajan. AlphaStar ei pelkästään tee päätöksiä nopeammin kuin ihminen, se tekee myös parempia päätöksiä. Ammattilaispelaajat ovat jopa sanoneet, että AlphaStar loi sellaisia strategioita, joita he eivät ole edes koskaan ajatelleet ennen. Tämä kertoo siitä, että vaikka StarCraftia on pelattu jo vuosikymmenen ajan, vieläkin uusia tapoja pelata sitä voidaan löytää. [17.]

Kun ottaa huomioon kaiken tämän, voi nähdä helposti, miksi koneoppiminen ja neuroverkot ovat voimakas tekniikka luoda tekoäly. Neuroverkoilla voi luoda hyvin älykkään tekoälyn, joka voittaa parhaimpia pelaajia vastaan. Tämä on kuitenkin myös sen suurin heikkous. Suurimmassa osassa peleissä ei haluta näin älykästä tekoälyä. Neuroverkkoja ei myöskään pysty hallitsemaan suunnittelijan kannalta hyvin. Suunnittelija ei voi tehdä tarkkoja muutoksia tekoälyn käytökseen, vaan kaikki riippuu neuroverkosta itsestään. Toinen merkittävä huono puoli on se, että neuroverkot pitää aina kouluttaa, ja se vie huomattavasti aikaa. Jos tekee pienenkin muutoksen pelin kulkuun tai muuten sen yleiseen rakenteeseen, joutuu luultavasti kouluttamaan neuroverkon taas uudestaan. [1, s. 398.]

4 Pohdinta

Seuraavassa kerrataan esitellyt tekniikat ja kuvataan omia kokemuksia niiden käytöstä sekä niiden toimivuudesta peleissä.

Yksinkertainen päätöksenteko on todennäköisesti ollut monilla käytössä jossakin projektissa. Se ei sovellu välttämättä nykyisten kaupallisten pelien tekemiseen, mutta sillä tulee aina olemaan paikka yksinkertaisia pelejä tehdessä. Siitä seuraava askel on äärelliset tilakoneet, jotka ovat hyvin toimiva ratkaisu kaikenlaisiin peleihin. Ne sopivat isoihin tai pieniin peleihin ja ovat yksi parhaimmista tavoista toteuttaa tekoäly helposti, mutta toimivasti. Tilakoneet ovat pysyneet ennallaan jo vuosia pienillä muutoksilla ja luultavasti pysyvät sellaisina tulevaisuudessakin.

Käytöspuut ovat yleisin ja helpoin tapa toteuttaa tekoäly, varsinkin, jos käyttäjällä on visuaalinen tapa muokata niitä. Monet pelit ovat onnistuneet tekemään erittäin hyviä tekoälyhahmoja käytöspuiden ansiosta, esimerkiksi Halo 2. Joka on yksi lempipeleistäni. Se on myös suurin syy mielenkiintooni tekoälyä kohtaan. Olen aina pitänyt sen tekoälyä yhtenä parhaimmista, joihin olen törmännyt. Vaikka se on suhteellisen yksinkertainen ensisilmäyksellä, sillä on silti yllättävän paljon syvyyttä. Tämä illuusio on varmasti yksi tärkein asia pelitekoälyä luodessa. Käytöspuut tulevat olevaan tärkeä tekniikka edelleen tulevaisuudessa.

Tavoitteellinen tehtäväsuunnittelija on yksi mielenkiintoisimmista tekniikoista. Se on vahva tekniikka, jos haluaa luoda suhteellisen dynaamista käyttäytymistä eri tekoälyhahmoille. On yllättävää, että tätä tekniikkaa ei ole käytetty enemmän eri peleissä. Pelit, jotka ovat käyttäneet sitä, ovat olleet suosittuja. Tosin voi olla, että monet pelit eivät tarvitse näin monimutkaista päätöksentekoa ja pystyvät käyttämään esimerkiksi käytöspuita, jotka on helpompi toteuttaa. Tekniikoiden monimutkaisuus on varmasti ratkaiseva asia pelinkehityksessä. Monesti tekoäly ei ole se asia, johon panostetaan paljon. Usein kehitysresurssit laitetaan mieluummin asioihin, jotka pelaajat näkevät heti, kuten grafiikkoihin tai lisätoiminnallisuuksiin.

Hierarkkiset tehtäväverkostot ovat looginen kehitysaskel tavoitteellisesta tehtäväsuunnittelijasta. Kehittäjät halusivat muokata tehtäväsuunnittelijaa niin, ettei siinä olisi niin paljon huonoja puolia, kuten vähäinen hallinta tekoälyn päätöksenteossa suunnittelijan näkökulmasta. Tämä johti entisestään monimutkaisempaan rakenteeseen, joka voi säikäyttää mahdolliset käyttäjät poispäin tästä tekniikasta varsinkin, jos sitä verrataan muihin yksinkertaisempiin tekniikoihin.

Ohjaajajärjestelmä on tekniikka, jota yleensä sovelletaan yhdessä jonkin muun tekniikan kanssa, kuten käytöspuiden. Sillä on mahdollista luoda hallittu pelintahti. Kuten peliesimerkeissä kävi ilmi, voidaan stressitasojen avulla hallita pelaajien pelikokemusta. Tämä on kiehtova asia, koska se eroaa muista tekniikoista siinä, että hallitsemme pelitilaa yhden hahmon sijasta.

Monte Carlo -menetelmä on tärkeä tekniikka Total War -strategiapelisarjalle. Varsinkin heidän ensimmäisissä peleissään, ennen kuin Monte Carlo -menetelmä oli heidän käytössään, tekoälyä oli erittäin helppo huijata. Valitettavasti Total Rome 2, jolloin Monte Carlo tuli käyttöön ensimmäisen kerran ei ollut kovin suosittu pelaajien kesken. Pitkät odotusajat vuorojen välissä vaikuttivat tähän. Tästä voidaan nähdä, kuinka tärkeää on saada toimiva tekoäly jo heti julkaisussa.

Koneoppiminen ja neuroverkot tulevat olemaan potentiaalisesti yksi tärkeimmistä tekniikoista. Emme varmastikaan ole nähneet kaikkea, mitä voidaan pystyä tekemään niiden avulla. Tähän mennessä peleissä sitä on sovellettu suhteellisen vähän, mutta jos löydetään oikea tapa soveltaa sitä, se tulee hyödyttämään pelien tekoälyä huomattavasti. Neuroverkot eivät tule ikinä olemaan paras tapa luoda tekoälyä suurimmassa osassa peleissä, mutta sen hyötyjä on vaikea kiistää.

5 Yhteenveto

Tutkimisen aikana käytiin läpi erilaisia päätöksentekotekniikoita. Tarkoituksena oli selostaa niiden toimivuutta ja missä peleissä niitä on käytetty. Työ alkoi siitä mitä pelitekoäly on ja miten se eroaa akateemisesta tekoälystä. Pelitekoälyn tarkoitus yleisesti on luoda hauska kokemus pelaajalle. Ei ole tarkoitus luoda kaikista älykkäintä tekoälyä ikinä, vaan luoda illuusio älykkyydestä.

Tämän jälkeen siirryttiin eri päätöksentekotekniikoihin ja niiden selostukseen. Läpikäydyt tekniikat olivat yksinkertainen päätöksenteko, äärellinen tilakone, käytöspuu, tavoitteellinen tehtäväsuunnittelija, hierarkkiset tehtäväverkostot, ohjaajajärjestelmä, Monte Carlo -menetelmä, hyötypohjainen päätöksenteko ja neuroverkot. Näiden tekniikoiden lisäksi on olemassa paljon enemmänkin tekniikoita, mutta asia rajattiin suosituimpiin ja mielenkiintoisimpiin tekniikoihin pelialalla.

Yksinkertainen päätöksenteko on hyvin yksinkertainen rakenteeltaan. Se sopii vain pieniin kehittäjätiimeihin, joissa on yksi ohjelmoija. Äärellinen tilakone on seuraava askel tästä. Se säilyttää suhteellisen yksinkertaisen rakenteen, mutta sen ylläpitäminen on paljon helpompaa ja se sopii hieman suuremmille peleille sekä tiimikoolle. Käytöspuut olivat ensimmäinen kunnollinen tekniikka, jota voi käyttää melkein missä tilanteessa vain. Tätä tekniikkaa käytetään nykyjään monissa eri peleissä mukaan lukien suuret pelit, kuten esimerkiksi Bungien julkaisema Halo 2 -peli.

Tavoitteellinen tehtäväsuunnittelija oli ensimmäinen selitetty suunnittelija työssä. Tämä eroaa edellisestä tekniikoista siinä, että se ei reagoi tapahtumiin kuten käytöspuut vaan sen sijaan rakentaa listan tietyistä toimista, jotka johtavat tiettyyn lopputulokseen. Tämä tekee tästä tekniikasta paljon dynaamisemman, mutta kehittäjä menettää hieman hallintaa sen päätöksenteosta.

Hierarkkiset tehtäväverkostot ovat evoluutio tavoitteellisesta tehtäväsuunnittelijasta. Se luotiin korjaamaan heikkouksia tehtäväsuunnittelijassa. Tätä tekniikkaa ei ole käytetty kovin paljon eri peleissä, mutta se on voimakas työkalu, jos halutaan tehtäväsuunnittelijan hyvät puolet, mutta silti säilyttää suunnittelijan hallinnan päätöksenteossa.

Ohjaajajärjestelmä antoi mahdollisuuden hallita pelintahtia tekoälyllä sen sijaan, että hallittaisiin pelkästään yhtä hahmoa. Tästä peliesimerkkeinä olivat Left 4 Dead ja Alien: Isolation. Molemmat pelit hyötyivät erittäin paljon tästä järjestelmästä.

Monte Carlo -menetelmä on puurakenteinen algoritmi. Sen tarkoitus on simuloida pelin kulkua tulevaisuuteen ja siten ennustaa parhain siirto. Tämä sopii parhaiten strategiapeleihin, kuten Total War -sarjaan.

Hyötypohjaisen päätöksenteon päätökset perustuvat eri toimintojen arvoihin. Eri toimintojen hintoja vertaamalla se voi tehdä päätöksen valitsemalla parhaimman vaihtoehdon. Tämä päätöksenteko on sekoitus suunnittelijan ja käytöspuun kanssa.

Viimeiseksi tekniikaksi jäi neuroverkot, jotka ovat erilaisin käydyistä tekniikoista. Se on koneoppimistekniikka, joka ei normaalisti sovellu pelitekoälyyn. Vuosien varrella on kuitenkin löydetty tapoja hyödyntää neuroverkkoja peleissä. Neuroverkkojen perusidea on tekoäly, jota voidaan kouluttaa harjoitteluseteillä. Tämä itseoppiminen mahdollistaa neuroverkkojen hyvin monimutkaisen päätöksenteon verrattuna aikaisempiin tekniikoihin.

Opinnäytetyön aikana kuvattiin tekniikoita ja niiden ominaisuuksia. Kaikista näistä tekniikoista löytyi jonkinlaisia heikkouksia, jonka takia on kehitetty uusia ratkaisuja. Uusien ratkaisujen myötä löytyy myös aina uusia ongelmia. Tämä johtuu siitä, että pelit voivat olla hyvin erilaisia toisistaan. Ratkaisu, joka toimisi yhdessä, ei välttämättä toimisikaan ollenkaan jossain toisessa pelissä.

Lähteet

1 Rabin S, Rabin S. Game AI pro : collected wisdom of game AI professionals. Boca Raton: CRC Press, Taylor & Francis Group; 2014.

2 Millington I, Funge JD, Funge J. Artificial intelligence for games. 2nd ed ed. Boca Raton, FL: CRC Press; 2009.

3 Pittman J. The Pac-Man Dossier. 2009; Available at: https://www.gamasutra.com/view/feature/132330/the_pacman_dossier.php. Accessed 10.12., 2019.

4 Simpson C. Behavior trees for AI: How they work. 2014; Available at: https://www.gamasutra.com/blogs/ChrisSimpson/20140717/221339/Behavior_trees_for_AI_How_they_work.php. Accessed 16.11., 2019.

5 Isla D. GDC 2005 Proceeding: Handling Complexity in the Halo 2 AI. 2005; Available at: https://www.gamasutra.com/view/feature/130663/gdc_2005_proceeding_handling_.php. Accessed 17.11., 2019.

6 Ghallab M, Nau DS, Traverso P. Automated Planning : Theory & Practice. : Morgan Kaufmann; 2004.

7 Symbolic representation of game world state: Toward real-time planning in games. Proceedings of the AAAI Workshop on Challenges in Game Artificial Intelligence; 2004.

8 Three states and a plan: the AI of FEAR. Game Developers Conference; 2006.

9 Thompson T. Cybertron Intel – HTN Planning in Transformers: Fall of Cybertron. 2016; Available at: https://aiandgames.com/cybertron-intel/. Accessed 19.11., 2019.

10 Thompson T. In the Directors Chair: The AI of Left 4 Dead. 2014; Available at: https://medium.com/@t2thompson/in-the-directors-chair-the-ai-of-left-4-dead-78f0d4fbf86a. Accessed 2.11., 2020.

11 Thompson T. The Perfect Organism: The AI of Alien: Isolation. 2017; Available at: https://www.gamasutra.com/blogs/TommyThompson/20171031/308027/The_Perfect_Organism_The_AI_of_Alien_Isolation.php. Accessed 2.11., 2020.

12 Thompson T. Revisiting the AI of Alien: Isolation. 2020; Available at: https://www.gamasutra.com/blogs/TommyThompson/20200520/363134/Revisiting_the_AI_of_Alien_Isolation.php. Accessed 2.11., 2020.

13 Liu M. General Game-Playing With Monte Carlo Tree Search. 2017; Available at: https://medium.com/@quasimik/monte-carlo-tree-search-applied-to-letterpress-34f41c86e238. Accessed 3.11., 2020.

14 Thompson T. Revolutionary Warfare | The AI of Total War (Part 3). 2018; Available at: https://www.gamasutra.com/blogs/TommyThompson/20180212/314399/Revolutionary_Warfare__The_AI_of_Total_War_Part_3.php. Accessed 3.11., 2020.

15 Rabin S. Game AI pro 2 : collected wisdom of game AI professionals. Boca Raton: CRC Press, Taylor & Francis Group; 2015.

16 Rasmussen J. Are Behavior Trees a Thing of the Past? 2016; Available at: https://www.gamasutra.com/blogs/JakobRasmussen/20160427/271188/Are_Behavior_Trees_a_Thing_of_the_Past.php. Accessed 3.11., 2020.

17 The AlphaStar team. AlphaStar: Mastering the Real-Time Strategy Game StarCraft II 2019; Available at: https://deepmind.com/blog/article/alphastar-mastering-real-time-strategy-game-starcraft-ii. Accessed 4.11., 2020