Lappeenrannan teknillinen yliopisto
Tuotantotalouden tiedekunta
Tietotekniikan koulutusohjelma

Kandidaatintyö

Joonas Salminen

TIETOKONEPELIEN TEKOÄLYN RAJAT

Työn tarkastaja(t): TkT Jussi Kasurinen

Työn ohjaaja(t): TkT Jussi Kasurinen

TIIVISTELMÄ

Lappeenrannan teknillinen yliopisto
Tuotantotalouden tiedekunta
Tietotekniikan koulutusohjelma

Joonas Salminen

Tietokonepelien tekoälyn rajat

Kandidaatintyö

2014

38 sivua, 7 kuvaa, 0 taulukkoa, 0 liitettä

Työn tarkastajat: TkT Jussi Kasurinen

Hakusanat: tekoäly, tietokone, pelit, algoritmi, pelinkehitys, älykäs laskenta
Keywords: artificial intelligence, games, algorithm, game development, intelligent computing, AI

Tutkimuksessa selvitettiin, kuinka hyvä tekoäly tietokonepeliin on mahdollista toteuttaa nykytiedolla ja -tekniikalla. Tekoäly rajattiin tarkoittamaan tekoälyn ohjaamia pelihahmoja. Lisäksi yksinkertaisia tekoälytoteutuksia ei huomioitu. Työ toteutettiin tutustumalla aiheeseen liittyvään kirjallisuuteen sekä kehittäjäyhteisön web-sivustojen tietoon. Hyvän tekoälyn kriteereiksi valikoituivat viihdyttävyys ja uskottavuus. Katsaus suosituimpiin toteuttamistekniikoihin ja tekoälyn mahdollisuuksiin osoitti, että teoriassa hyvinkin edistynyt tekoäly on toteutettavissa. Käytännössä tietokoneen rajalliset resurssit, kehittäjien rajalliset taidot ja pelinkehitysprojektien asettamat vaatimukset näyttävät kuitenkin rajoittavan tekoälyn toteuttamista kaupallisessa tuotteessa.

ABSTRACT

Lappeenranta University of Technology
School of Industrial Engineering and Management
Degree Program in Computer Science

Joonas Salminen

Limits of artificial intelligence in computer games

Bachelor's Thesis

38 pages, 7 figures, 0 tables, 0 appendices

Examiners: D.Sc. Jussi Kasurinen

Keywords: artificial intelligence, games, algorithm, game development, intelligent computing, AI

The purpose of this research was to find out how good artificial intelligence it is possible to implement in a computer game using modern technology and techniques. AI was delimited to include AI controlled characters only. Simple AI implementations were left out. The research was carried out by exploring related literature and game development community websites. Believability and entertainment value were chosen for the criteria of good AI. A review through the most popular AI techniques and a look at the possibilities of AI indicated that in theory highly advanced AI is realizable. In practice, however, limited resources of computers, limited knowledge of game developers and requirements set by game development projects seem to constrain commercial AI implementations.

ALKUSANAT

Tämä työ on tehty Lappeenrannan teknillisessä yliopistossa syksyllä 2014. Haluaisin kiittää läheisiäni henkisestä tuesta työtä tehdessäni.

SISÄLLYSLUETTELO

SYMBOLI- JA LYHENNELUETTELO

FSM	Finite State Machine
KR	Knowledge Representation
NLP	Natural Language Processing
NPC	Non-Player-Character
POV	Point of Visibility

1 JOHDANTO

1.1 Tausta

Peliteollisuus on viime vuosikymmenen aikana noussut yhdeksi merkittävimmistä ja nopeimmin kasvavista viihdeteollisuuden haaroista. Vuonna 2012 pelimarkkinoiden maailmanlaajuinen arvo oli noin 79 miljardia dollaria [7] ja myynnin odotetaan saavuttavan 81,5 miljardin dollarin rajapyykin tämän vuoden aikana. [11] Yhtenä syynä pelibisneksen kukoistukseen voidaan pitää tekniikan kehitystä, joka on mahdollistanut vuosi vuodelta näyttävämmän graafisen ulkoasun ja paremman pelattavuuden tehden peleistä näin helpommin lähestyttäviä. Toisaalta suhtautuminen elektronisten pelien pelaamiseen on muuttunut: pelaamista ei enää pidetä tietokoneharrastajien keskinäisenä huvina, vaan se mielletään yhdeksi viihdemuodoksi siinä missä elokuvat ja musiikkikin.

Vuosittain julkaistaan valtava määrä erilaisia videopelejä eri laitealustoille PC:stä pelikonsolien kautta mobiililaitteisiin. Luonnollisesti jotkut pelit ovat parempia kuin toiset ja parhaimmat niistä onnistuvatkin houkuttelemaan miljoonia ihmisiä sadoiksi tunneiksi pelin pariin. Mikä sitten saa ihmisen pelaamaan tiettyä peliä yhä uudestaan? Koukuttava idea, näyttävä grafiikka ja sujuva pelattavuus eivät välttämättä vielä riitä pitämään yllä pelaajan mielenkiintoa. Sovelluksesta riippuen tarvitaan myös toimiva tekoäly. Tekoälyn tehtävänä on haastaa pelaaja ja tuoda sisältöä pelimaailmaan. Toisinaan yksinkertaisestikin toteutettu tekoäly riittää näiden vaatimusten täyttämiseen. Nykyaikaisissa suuren budjetin peleissä tarvitaan kuitenkin usein huomattavasti monimutkaisempaa lähestymistapaa.

Nykyteknologialla ja -tietotaidolla hyvän tekoälyn rakentamisen luulisi olevan helppoa. Näin ei kuitenkaan ole, sillä kunnollisen tekoälyn toteuttamiseen liittyy omat tekniikkansa, ongelmansa ja rajoitteensa. Lisäksi päänvaivaa tuottaa tekoälyn hyvyyden määrittely: milloin voidaan sanoa, että pelissä on hyvä tekoäly? Otetaan esimerkiksi EA Sportsin FIFA-pelisarja. Pelistä julkaistaan vuosittain uusi versio, jota varten pelistä pyritään korjaamaan edellisen version epäkohdat ja lisäämään uusia ominaisuuksia. Vuodesta toiseen eniten kritiikkiä saa juuri pelin tekoäly, joka tuntuu siitä huolimatta edistyvän turhan hitaasti. Kysymys siis kuuluu: onko pelitekoälyn rajat saavutettu?

1.2 Tavoitteet ja rajaukset

Tämän työn tavoitteena on selvittää, kuinka hyvä tekoäly tietokonepeliin on mahdollista toteuttaa nykytiedolla ja -tekniikalla. Lähtökohtana on pohtia tekoälyn hyvyyden määrittelyä ja sen jälkeen syventyä erilaisiin lähestymistapoihin ja tekniikoihin tekoälyn toteuttamiseksi. Kun tekniikoiden perusteet ovat tiedossa, tarkoituksena on selvittää millaiseen lopputulokseen niillä voidaan parhaimmillaan päästä, mitä rajoitteita ja ongelmia tekoälyn rakentamiseen liittyy ja kuinka näitä rajoitteita voidaan mahdollisesti kiertää nyt ja tulevaisuudessa.

Työn pitämiseksi maltillisissa mittasuhteissa on hyvä tehdä myös muutamia rajauksia. Ensinnäkin tässä tutkimuksessa rajoitutaan tarkastelemaan ainoastaan tietokonepelien tekoälyä tietokoneiden suuremman laskentatehon ansiosta verrattuna varsinaisiin pelikonsoleihin [4]. Myös tekoälyn määrittelyä on syytä rajata, sillä nykyaikaisissa tietokonepeleissä on monia komponentteja, joiden toimintaa voidaan pitää älykkäänä, kuten fysiikkamoottori tai pelitapahtumiin mukautuva kamerakulma. Koska ihmisen kaltaisen päätöksenteon ja taktisen ajattelun mallintaminen on huomattavasti hankalampaa, ja täten rajoituksille alttiimpaa, kuin esimerkiksi perusfysiikan toteuttaminen peliin, tässä tutkimuksessa tekoälyllä tarkoitetaan yksinomaan pelaajaa vastaan kilpailevien, pelaajan puolella olevien tai pelaajaa tukevien hahmojen toimintaälyä. [9]

Tekoälyn rakentamiseen on olemassa lukuisia lähestymistapoja, mutta tässä työssä on tarkoituksenmukaista käydä läpi ainoastaan yleisimmät ja tutkimuskysymykseen vastaamisen kannalta olennaisimmat tekniikat tekoälyn eri osa-alueilta. Tekniikoita käyttävistä sovelluksista rajataan pois yksinkertaiset tekoälytoteutukset, jotka hyödyntävät ainoastaan muutamia tekniikoita ja tiettyä tekoälyn osa-aluetta. Tällaisia ovat esimerkiksi tietokoneella pelattavat lautapelit, kuten shakki [9], ja Nintendon Super Mario-pelisarjan kaltaiset tasohyppelyt. Edellä mainitun kaltaiset yksinkertaiset tekoälytoteutukset ovat usein sellaisia, että niiden parantaminen ei ole enää pelikokemuksen kannalta järkevää tai rajat ovat selvästi tiedossa ja kyse on pelkästä algoritmin viilaamisesta suorituskyvyn nostamiseksi. Tästäkin syystä niiden tarkastelu on tämän työn tavoitteen kannalta turhaa.

1.3 Työn rakenne

Aiheen käsittely aloitetaan luvussa 2 paneutumalla tekoälyn käsitteeseen: mitä tekoälyllä tarkoitetaan yleisesti ja mitä sillä tarkoitetaan tietokonepeleissä. Tämän jälkeen pohditaan vielä tekoälyn hyvyyttä ja pyritään luomaan kuva siitä, minkälaista tekoälyä tässä tutkimuksessa haetaan. Luku 3 käsittelee tekoälyn toteuttamista ja esittelee joukon tekniikoita, joita voidaan soveltaa tekoälyn eri osa-alueiden rakentamiseen. Luku 4 taas tarkastelee tekoälyn haasteita, eli mitä ongelmia ja rajoitteita tekoälyn toteuttamiseen liittyy ja pohtii onko niitä mahdollista kiertää. Lisäksi luku sisältää lyhyen katsauksen pelitekoälyn nykytilaan ja tulevaisuuteen. Luvussa 5 kootaan yhteen luvuissa 3 ja 4 tehdyt havainnot sekä johtopäätökset ja esitetään yhteenveto tutkimuksen tuloksista.

2 TEKOÄLY

2.1 Tekoäly käsitteenä

Tekoälyllä tarkoitetaan tietokoneen näennäistä kykyä ajatella ja toimia elävän olennon tavoin. Käytännön tasolla tekoäly on algoritmi tai joukko algoritmeja, joilla pyritään yleensä jäljittelemään ihmisen ajatusprosesseja esimerkiksi aistihavaintoihin tai päättelyyn liittyen. Usein tekoälyn käsite rinnastetaan suoraan samannimiseen tutkimusalaan ja tutkimuksen pohjalta tuotettuihin sovellutuksiin. Tekoälyä ja sen luomista onkin tutkittu 1950-luvulta lähtien, ja se on nykyisin yksi merkittävimmistä tutkimusaloista tietojenkäsittelytieteessä. [5]

Tietokone- ja konsolipelien yleistymisen myötä tekoälyn käsite on puhekielessä laajentunut tarkoittamaan myös peleissä itsenäisesti toimivia pelihahmoja, jotka voivat olla pelaajan puolella, pelaajaa vastaan tai jotakin siltä väliltä. Pelihahmojen toimintaälystä käytetty nimitys on kuitenkin kiistanalainen, sillä pelitekoälyn toteuttamiseen liittyy elementtejä, jotka ovat ristiriidassa niin sanotun akateemisen tekoälykäsityksen kanssa. Lisäksi pelitekoälyn arviointikriteerit poikkeavat merkittävästi akateemisen puolen vastaavista.

2.2 Akateeminen tekoäly ja tekoälytutkimus

Kirjallisuudessa akateemisella tekoälyllä viitataan yleensä juuri korkeakoulutasolla harjoitettuun tekoälytutkimukseen ja sen tuloksiin. Tekoälytutkimus on jakautunut alatutkimusalueisiin, joiden tutkimuskohteet voidaan edelleen luokitella tekoälyn luonteen perusteella vahvoihin ja heikkoihin tekoälyihin. Vahvan tekoälyn tapauksessa pyrkimyksenä on luoda täydellisesti ihmisen ajatustoimintaa imitoiva järjestelmä, kuten ihmisen tavoin toimiva robotti. Tätä pidetään tekoälyn perimmäisenä tavoitteena, johon ei toistaiseksi ole vielä päästy. Heikon tekoälyn tapauksessa tarkoituksena on hyödyntää tekoälyteknologioita erilaisten käytännön ongelmien ratkaisemiseksi tietyillä sovellusalueilla [3], kuten ihmisten tunnistamisessa. Käytännöllisyytensä ansiosta suurin osa tekoälyn alatutkimusalueista on keskittynyt pääasiassa heikkoon tekoälyyn [3].

Koska älykkyyden mallintamista voidaan lähestyä niin filosofisesta, psykologisesta kuin puhtaasti teknisestä näkökulmasta, on erilaisia alatutkimusalueita ja tutkimuskohteita luonnollisesti useita [5]. Tärkeimmiksi alatutkimusalueiksi tekoälyn saralla voidaan tällä hetkellä lukea päättely, suunnittelu, koneoppiminen, luonnollisen kielen käsittely (engl. NLP, natural language processing), puheen ja kohteiden tunnistaminen sekä tiedon esittäminen (engl. KR, knowledge representation). Tutkimuskohteet vaihtelevat käytännön sovellutuksista teoreettisen tason konsepteihin. Käytännön sovellutusten puolella tekoälytutkimukseen liittyy usein olennaisesti robotiikka.

Akateemisen tekoälysovellutuksen onnistuneisuuden voidaan katsoa tulevan siitä, kuinka hyvin se mukailee ihmiselle ominaista ajatusprosessia ratkaistessaan jonkin tietyn ongelman. Yleensä ongelma halutaan vielä ratkaista optimaalisesti, mikäli mahdollista [3]. Esimerkiksi kasvojentunnistusjärjestelmän laatua voitaisiin tarkastella arvioimalla sen osoittamaa älykkyyttä päätöksenteossa. Järjestelmää, joka ottaisi kuvan kasvoista ja vertailisi sitä suoraan tietokannasta löytyviin kasvokuviin, kunnes sattuisi samankaltaisen kuvan kohdalle, ei pidettäisi kovin älykkäänä. Sen sijaan järjestelmä, joka päättelisi kuvasta skannattujen kasvonpiirteiden ja tietokannasta löytyvien kasvoprofiilien perusteella parhaan vastaavuuden, täyttäisi jo akateemisen tekoälyn kriteerit. Lisäksi se antaisi lähes aina optimaalisemman tuloksen kuin päätöksensä puhtaaseen kuvien vertailuun perustava järjestelmä.

2.3 Tekoäly tietokonepeleissä

Tietokonepelien tekoälyä toteutettaessa tavoite on periaatteessa sama kuin akateemisen tekoälynkin tapauksessa, eli mallintaa älykkyyttä. Mutta siinä missä akateemisessa tutkimuksessa pyritään jäljittelemään elävän olennon ajattelua ja ratkaisemaan ongelmia optimaalisesti, pelitekoälyn tapauksessa ei ole väliä, miten lopputulokseen päästään. Tärkeintä on, että toiminta vaikuttaa älykkäältä. Tämä johtuu yksinkertaisesti siitä, että laitteistoresurssit (suoritinaika, muisti) ovat rajalliset. [3] Tietokoneiden on pystyttävä suorittamaan pelejä reaaliajassa ilman suurempia viiveitä, ja hiemankin monimutkaisempi tekoälytoteutus saattaa vaatia valtavia määriä laskutoimituksia sekunnissa. Suorituskyvyn takaamiseksi on toteutuksen suhteen tehtävä tiettyjä kompromisseja, joiden pohjalta onkin herännyt kiistaa siitä, voiko pelitekoälyä edes pitää tekoälynä. [5]

On esitetty, että pelitekoäly koostuu osaksi algoritmeista, osaksi heuristiikoista ja osaksi huijaamisesta [5]. Peliteollisuudessa käytettävät algoritmit ammentavat pitkälti akateemisen tutkimuksen tuloksista. Akateemiselta puolelta tuttuja tekniikoita ovat esimerkiksi tilakoneet, päätöspuut ja polunetsintäalgoritmit, jotka ovat laajalti käytössä erilaisissa peleissä. Nämä algoritmit ovat yleensä syy siihen, miksi useimmat mieltävät pelitekoälyn osaksi tekoälyjen kenttää.

Heuristiikoilla ja huijaamisella sen sijaan ei ole mitään tekemistä älykkyyden kanssa. Siitä huolimatta ne ovat yhtä tärkeässä osassa pelitekoälyn rakentamista kuin algoritmitkin. Heuristiikalla tarkoitetaan tavallisesti ns. peukalosääntöä, jonka avulla ratkaisua johonkin ongelmaan lähdetään hakemaan. [5] Jos tehtävänä on esimerkiksi löytää lyhin mahdollinen reitti kahden pisteen välillä, voidaan määrittää heuristiikka, jonka avulla ratkaisua lähdetään hakemaan aina parhaaksi katsotusta suunnasta sen sijaan, että käytäisiin joka kerta kaikki reittivaihtoehdot vaihtoehdot läpi ilman erikseen laadittua järjestystä. Tämä ei välttämättä tuota optimaalista ratkaisua, mutta riittävän hyvän, jotta sitä voidaan pelissä hyödyntää.

Huijaamisella viitataan erilaisiin ad hoc -ratkaisuihin ja temppuihin, jotka ovat toisinaan välttämättömiä, jotta suorituskyky saadaan pidettyä järjellisellä tasolla [5]. Otetaan esimerkiksi tilanne, jossa tekoälyhahmon on selviytyäkseen pääteltävä pelaajan ohjastaman hahmon piilopaikka sekunnin murto-osassa. Sen sijaan, että tekoäly tuhlaisi laitteistoresursseja pelaajan paikantamiseen älykkäästi tekemällä päätelmiä aiempien havaintojensa pohjalta, se voi huijata katsomalla pelaajan sijainnin suoraan muistiin talletetuista tiedoista. Toinen hyvä esimerkki luovasta tekoälyn rakentamisesta on pelihahmojen tunteiden esittäminen. Ei ole tarvetta käyttää monimutkaisia kognitiivisia malleja, geneettisiä algoritmeja tai oppimista, jotta hahmo saadaan näyttämään esimerkiksi vihaiselta. Usein riittää esittää tunnetta ilmaiseva animaatio, kuten kulmien kurtistaminen pelihahmon kasvoilla, sopivassa kohdassa. [5]

Heuristiikkojen ja erilaisten ad hoc -ratkaisuiden hyödyntäminen pelien tekoälytoteutuksissa on hämärtänyt käsitystä siitä, mikä lasketaan tekoälyksi [5]. Siksi onkin tärkeää jo varhaisessa vaiheessa tehdä selkeä ero akateemisen tekoälyn ja

pelitekoälyn välille. Siihen, onko pelitekoäly loppujen lopuksi varsinaista tekoälyä vai ei, ei tässä työssä lähdetä ottamaan sen enempää kantaa.

2.4 Tekoälyn hyvyys

Kuten johdannossa jo aiemmin todettiin, yksi merkittävä tekijä pelaajan mielenkiinnon kannalta peliä kohtaan on se, kuinka hyvä tekoäly pelissä on. Milloin sitten voidaan sanoa, että pelissä on hyvä tekoäly? Yleensä tekoälyä rakennettaessa tavoitellaan uskottavuutta [10] ja ennen kaikkea viihdyttävyyttä [3]. Näiden tavoitteiden täyttymistä voidaan hyvällä syyllä pitää myös hyvän tekoälyn kriteereinä. Tekoälyn hyvyyttä arvioitaessa tärkeää on myös ottaa huomioon yksittäisen pelihahmon rooli pelissä eli se, onko kyseessä pelaajaa vastaan kilpaileva, pelaajan puolella oleva vai pelaajan toimintaa tukeva hahmo. [10]

Pelaajaa vastaan kilpailevien hahmojen (esimerkiksi vastustajajoukkueen pelaajat urheilupelissä tai tekoälyviholliset eli botit ensimmäisen persoonan ammuskelupelissä) tehtävänä pelissä on haastaa pelaaja pelin sääntöjen rajoissa. Koska tekoäly ottaa tässä yhteydessä usein toisen ihmispelaajan roolin, uskottavuuden voidaan katsoa syntyvän siitä, kuinka hyvin tekoäly pystyy jäljittelemään ihmisen toimintaa erilaisissa pelitilanteissa. [10] Mikäli tekoälyvastustajat eivät ole ihmisiä tai ihmisenkaltaisia olentoja, vaan esimerkiksi eläimiä, uskottavuus riippuu siitä, minkälaista käytöstä tietyntyyppiseltä oliolta odotetaan. Pelin viihdyttävyyden kannalta taas on tärkeää, että tekoälyvastustajat tarjoavat riittävästi haastetta pelaajalle. Ne eivät kuitenkaan saa olla niin älykkäitä ja taitavia, että pelissä menestyminen on mahdotonta. Toisaalta taas heikkotasoiset tekoälyvastustajat tekevät pelistä liian helpon. [8] Sopivan vaikeustason löytäminen pelin viihdyttävyyden takaamiseksi onkin siksi yksi suurimmista haasteista tekoälyvastustajia kehitettäessä.

Pelaajan puolella olevat hahmot (esimerkiksi joukkuetoverit urheilupelissä) yrittävät auttaa pelaajaa saavuttamaan tavoitteensa pelissä. Tällaisia hahmoja kutsutaan joissakin yhteyksissä myös nimellä NPC (engl. non-player-character) eli ei-pelaajahahmo. [10] Suora kommunikointi tekoälyn ohjaamien kumppaneiden kanssa on kuitenkin usein varsin rajoitettua, joten tekoälyn on mukauduttava pelaajan pelityyliin pelitapahtumien perusteella. [9] Uskottavuuden kannalta kumppanihahmojen ei tarvitse toimia ihmispelaajan tavoin, vaan tärkeintä on, että toiminta on ylipäätään älykästä. [10]

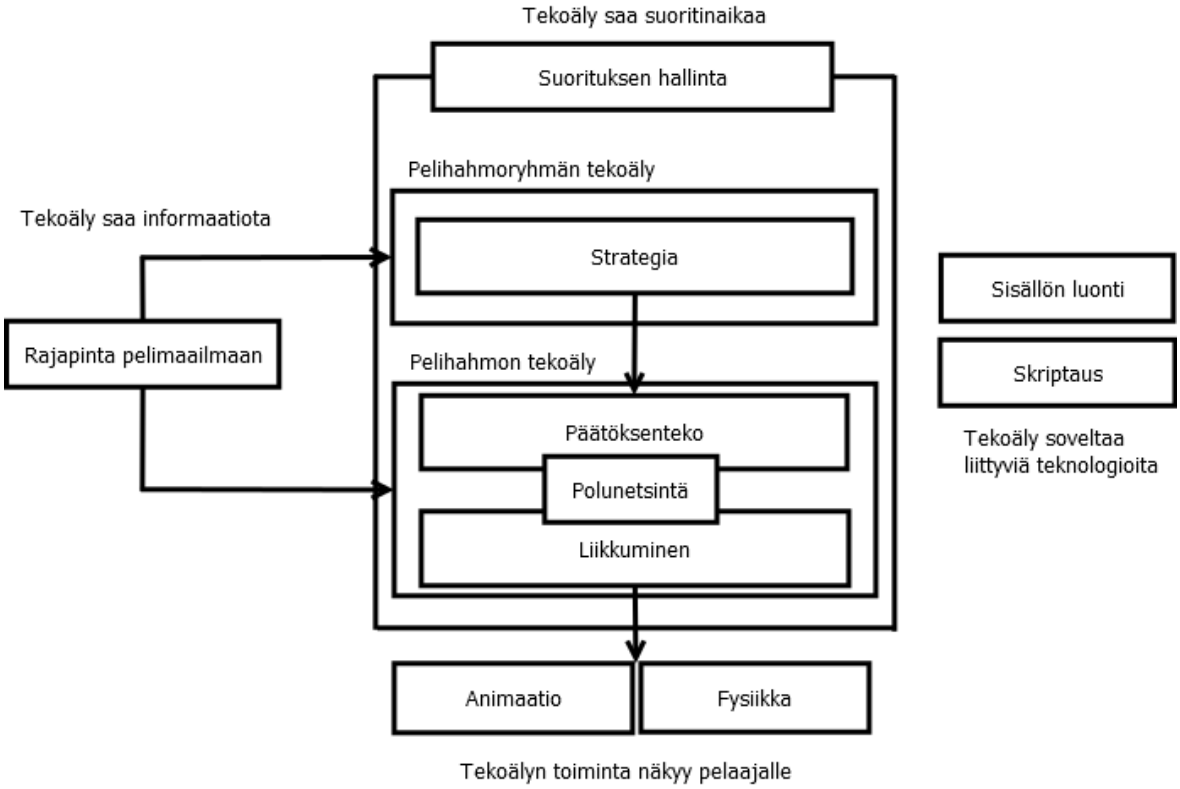
Viihdyttävyyden suhteen kriteerit ovat pitkälti samat kuin pelaajaa vastaan kilpailevien hahmojen tapauksessa: kumppaneiden tekoäly ei saa tehdä peliä liian helpoksi tai vaikeaksi pelaajalle.

Pelaajan avustamisen lisäksi tekoäly voi olla pelaajan toimintaa tukevassa roolissa. Myös tällaisia hahmoja kutsutaan ei-pelaajahahmoiksi, koska ne eivät vaadi ihmispelaajan roolin ottamista. [10] Tukihahmojen tehtävänä on kansoittaa pelimaailmaa ja tukea pelaajaa esimerkiksi vastaamalla tämän kysymyksiin. Usein kommunikointi pelaajan kanssa tapahtuu niin, että pelaajan annetaan valita tietyistä vaihtoehdoista kysymys tai muu ilmaus, johon tekoälyhahmo antaa valinnasta riippuvan valmiiksi purkitetun vastauksen. [9] Uskottavuus on tärkeässä osassa tukihahmojen tekoälyn hyvyyttä määritettäessä. Ensinnäkin käyttäytymisen tulee vaikuttaa älykkäältä: tekoälyhahmo, joka kulkee ympyrää pelimaailmassa ilman suurempaa tarkoitusta, ei näytä pelaajan silmissä kovin uskottavalta. Toisekseen reagoinnin erilaisissa kommunikointitilanteissa pelaajan kanssa tulee olla uskottavaa. Viihdyttävyyden voidaan katsoa tulevan siitä, kuinka monipuoliset kommunikointimahdollisuudet pelissä on. Jos kaikki keskustelut tukihahmojen kanssa ovat liki identtisiä, pelaaja kyllästyy nopeasti kuulemaan samoja vakiorepliikkejä.

Kaiken kaikkiaan pelin tekoälyn hyvyyttä arvioitaessa on siis mietittävä, mitä eri rooleja tekoälyllä pelissä on ja kuinka hyvin ne täyttävät tekoälylle asetetut tavoitteet viihdyttävyydestä ja uskottavuudesta. Ihmisillä saattaa kuitenkin olla erilaisia käsityksiä siitä, mikä on uskottavaa tekoälyhahmon käyttäytymistä ja milloin peli on viihdyttävä. Tähän on esitetty syyksi sitä, että pelaajien taso vaihtelee: osa on kokeneita pelaajia, kun osa taas vasta aloittelee peliharrastustaan. [10] Tässä tutkimuksessa voidaan ottaa ainakin uskottavuuden osalta ottaa lisäkriteeriksi se, että tekoälyn on oltava uskottava pelaajan tasosta riippumatta. Tekoälyn viihdyttävyyttä sen sijaan on ymmärrettävästi tarkasteltava ns. keskivertopelaajan näkökulmasta.

3 TEKOÄLYN TOTEUTTAMINEN

Ennen kuin voidaan tarkastella pelien tekoälyyn liittyviä ongelmia ja rajoituksia on tiedettävä kuinka tekoäly rakennetaan. Tekoälyn rakentamiseen ei ole olemassa yhtä oikeaa tapaa, vaan työkalut ja tekniikat on valittava aina toteutettavan pelin mukaan. Sovelluksesta riippumatta tekoälyn voidaan kuitenkin peruspiirteissään katsoa noudattavan aina tiettyä rakennetta. Funge ja Millington esittelevät kirjassaan mallin, joka jakaa pelitekoälyn osa-alueisiin sen suorittamien tehtävien luonteen perusteella. Malli on esitetty kuvassa 1.



Kuva 1. Pelitekoälyn malli [5]

Kuten Funge ja Millington kirjassaan toteavat, malli on vain yksi vaihtoehto tekoälyn rakenteen esittämiselle [5]. Selkeytensä vuoksi sitä tullaan kuitenkin käyttämään pohjana tässä tutkimuksessa, kun tarkastellaan tekoälyn toteuttamista. Seuraavissa aliluvuissa avataan tarkemmin mallin eri osien merkitystä sekä käydään läpi erilaisten tekoälyn tehtävien toteuttamisen kannalta olennaisimpia tekniikoita ja työkaluja.

3.1 Liikkuminen

Alimmalla tasolla tekoälyn tehtävänä on huolehtia pelihahmojen liikkumisesta pelimaailmassa. Jokaisella pelihahmolla on yleensä lista ominaisuuksia, jotka määrittelevät sen tilan. Liikkumisen kannalta olennaisia ovat esimerkiksi sijainti, nopeus tai jokin muu liikkumiseen vaikuttava fyysinen ominaisuus. Näitä ominaisuuksia sekä pelimaailman tilasta kerättyä tietoa käytetään geometrisessä muodossa liikkumisalgoritmien syötteenä. Liikkumisalgoritmit palauttavat vastaavasti geometrisessä muodossa tiedon siitä, miten ja minne pelihahmon tulisi liikkua. Tämä tieto välitetään liikkumisen toteuttaville fysiikkamoottorille sekä animaatioteknologialle. [5]

Liikkumisalgoritmeja on kahdenlaisia: kinemaattisia ja dynaamisia. Kinemaattiset algoritmit käyttävät pelkästään staattista dataa, kuten pelihahmon sijaintia ja orientaatiota (katseen suuntaa) liikesuunnan ja nopeuden määrittämiseen. Yksinkertaisimmillaan ne voivat esimerkiksi ottaa syötteenä vastaan pelihahmon sekä sen kohteen koordinaatit ja palauttaa nopeuden sekä suunnan kohteeseen vektorimuodossa. Kinemaattiset algoritmit eivät ota huomioon liikkeen kiihtymistä tai hidastumista, mikä voi saada liikkeen näyttämään epäuskottavalta etenkin silloin, kun nopeus muuttuu jyrkästi. Liikettä on tosin mahdollista pehmentää jakamalla sen esittäminen näytöllä useammalle kehykselle (engl. frame). [5]

Melkein kaikissa nykypeleissä pelihahmojen on kyettävä väistelemään esteitä matkalla kohteeseensa. Tyypillisesti staattisten esteiden, kuten seinien, välttelystä pitää huolen tekoälyn polunetsinnän (kts. luku 3.2) toteuttava osa. Tyypillinen ongelma ovat kuitenkin dynaamiset esteet, joiden paikka pelimaailmassa vaihtelee pelin edetessä. Tällaisia ovat esimerkiksi muut pelihahmot tai liikuteltavat esineet. Muun muassa tämän ongelman ratkaisemiseksi käytetään dynaamisia liikkumisalgoritmeja, joita voidaan myös nimittää ohjauskäyttäytymiseksi (engl. steering behaviors). [3] Toisin kuin kinemaattiset algoritmit, dynaamiset algoritmit ottavat huomioon kiihtyvyyden ja palauttavat nopeuden sijaan voiman, jolla pelihahmon liiketilaa pyritään muuttamaan. [5] Tämä luonnollisesti lisää liikkeen uskottavuutta, joka, kuten aiemmin todettiin, saattaa toisinaan olla ongelma kinemaattisen liikkeen yhteydessä.

Dynaamisia algoritmeja on lukuisia erilaisia ja niillä voidaan saada aikaan monenlaista käyttäytymistä. Mainitun esteiden välttelyn lisäksi pelihahmo voidaan esimerkiksi asettaa hakeutumaan kohteeseen, ajamaan takaa kohdetta, pakenemaan uhkaa, seuraamaan tiettyä polkua tai vaeltelemaan näennäisen vapaasti pelimaailmassa. Halutun käytöksen aikaansaamiseksi algoritmeja on yhdisteltävä sopivalla tavalla. [3] Dynaamisia algoritmeja käytetään yksittäisten hahmojen lisäksi pelihahmoryhmien käyttäytymistä mallinnettaessa. Itse asiassa suosituin dynaamisten algoritmien käyttökohde onkin parveilukäyttäytymisen (engl. flocking), kuten eläinlauman yhtenäisen liikkeen, aikaansaaminen. [5]

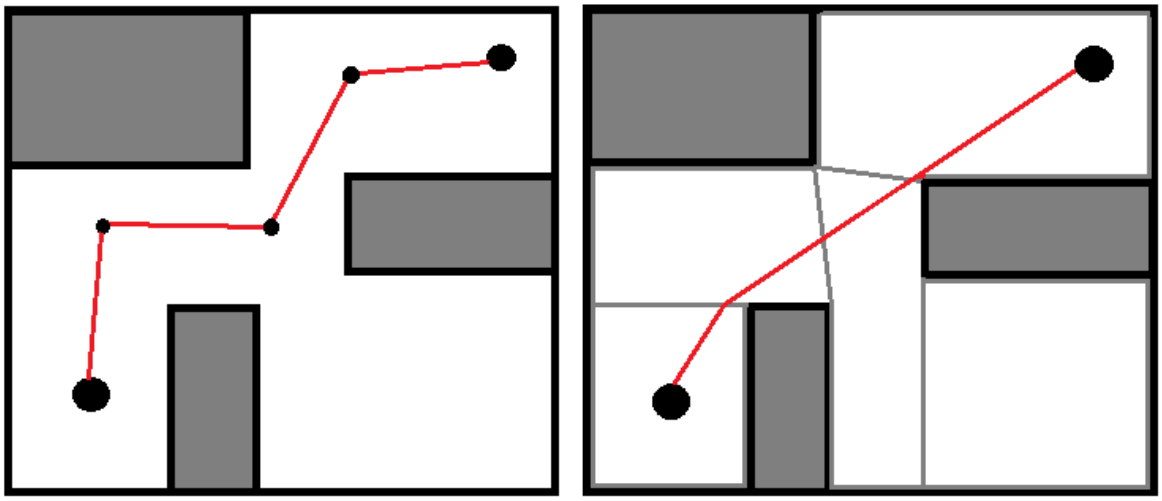
3.2 Polunetsintä

Pelitekoälyn mallissa polunetsintä voidaan ajatella päätöksentekotason ja liikkumistason välille. Usein se on upotettu osaksi liikkumisjärjestelmää, jolloin sitä voidaan kutsua tarpeen tullen. Polunetsinnässä tarkoituksena on yleensä löytää sopivin reitti paikasta A paikkaan B. Tämä reitti toimii ikään kuin seurattavana karttana pelihahmon liikkumisen toteuttavalle osalle. Reitin löytämiseksi käytetään useimmiten A* -nimistä algoritmia, josta on olemassa kymmenittäin erilaisia variaatioita. A* -algoritmi pohjautuu Dijkstran algoritmiin, jota sovelletaan enemmänkin tekoälyn strategiatasolla. [5] Siitä on kuitenkin toisinaan hyötyä myös polunetsinnässä [3].

Ennen kuin polunetsintäalgoritmi voi aloittaa sopivimman reitin hakemisen, on pelimaailma esitettävä sille sopivassa muodossa. Tämä tarkoittaa maaston yksinkertaistamista solmuista ja kaarista koostuvan graafin muotoon. Tätä yksinkertaistamisprosessia kutsutaan kvantisoinniksi. Solmuihin jakaminen voi tapahtua useilla eri tavoilla ja näitä tapoja kutsutaan ositusskeemoiksi (engl. division schemes). [5] Solmut voivat edustaa esimerkiksi näkyvyyspisteitä (engl. POV, points of visibility), eli kohteita, joilla on näköyhteys vähintään yhteen naapurikohteeseen. Reaaliaikaisissa strategiapeleissä taas on tapana jakaa maasto laattoihin tai soluihin, joiden keskipisteet toimivat reittipisteinä. [3]

Suosituin ositusskeema nykypäivänä lienee kuitenkin navigaatioverkko, jossa maasto on jaettu konvekseihin monikulmioihin [5]. Sen avulla tiedetään täsmälleen, millä alueilla

hahmojen on mahdollista liikkua, mikä mahdollistaa lyhyemmät reitit ja luonnollisemman liikkumisen. Tämä on nähtävissä kuvassa 2, jossa on esimerkinomaisesti esitetty laskettu reitti ensin näkyvyyspisteisiin ja sitten navigaatioverkkoon pohjautuen. Toisena etuna samaa verkkoa voidaan käyttää kaikkien tekoälyn ohjaamien hahmojen kohdalla riippumatta niiden ominaisuuksista, jolloin esimerkiksi erikokoisille hahmoille ei tarvitse määritellä omia reittipistejoukkoja. [8]



Kuva 2. Näkyvyyspisteisiin (vas.) ja navigaatioverkkoon (oik.) pohjautuvien reittien ero

Kun halutaan löytää sopivin (yleensä lyhin) reitti pelihahmon ja tietyn kohteen välillä käytetään käytännössä aina A* -algoritmia tai paremminkin jotakin sen variaatioista. Pohjimmiltaan A* -algoritmi käy iteroiden läpi graafin solmuja ja solmujen välisiä kaaria pyrkien löytämään reitin, jota noudattamalla reitin muodostavien kaarien kustannusten summa olisi mahdollisimman pieni. Läpikäyntijärjestys perustuu valittuun heuristiikkaan. Oikean heuristiikan valitseminen on tärkeä osa A* -algoritmin toteutusta, sillä mitä tarkempi heuristiikka on, sitä tehokkaammin algoritmi suoriutuu. Yksi laajalti käytössä oleva heuristiikka on hakusuunnan valitseminen perustuen euklidiseen etäisyyteen kahden solmun välillä. Tämä heuristiikka on hyvä esimerkki ympäristön huomioimisen tärkeydestä heuristiikan valinnassa. Avoimessa maastossa se toimii hyvin, koska esteitä ei juuri ole. Suljetussa tilassa, kuten rakennuksessa, polunetsintäalgoritmin suorituskyky sen sijaan voi laskea huomattavasti, koska heuristiikka aliarvioi etäisyyttä muun muassa seinien takia. [5]

Alla on nähtävissä mukailtuna Fungen ja Millingtonin kirjassaan esittämä pseudokoodiversio A* -algoritmista. Funktio ottaa syötteenä vastaan graafin, aloitussolmun, maalisolmun sekä heuristiikka-objektin, jonka estimate-jäsenfunktion avulla voidaan luoda arvio kustannuksesta minkä tahansa solmun saavuttamiseksi. Funktio palauttaa löydetyn reitin listarakenteena, joka sisältää reitin muodostavat kaaret.

```
def pathFindAStar(graph, start, goal, heuristic):
      # Tietue, joka sisältää solmun tiedot
      struct NodeRecord:
            node
            connection
            costSoFar
            estimTotalCost

      # Aloitussolmun tietojen alustus
      startRecord = new NodeRecord()
      startRecord.node = start
      startRecord.connection = None
      startRecord.costSoFar = 0
      startRecord.estimTotalCost = heuristic.estimate(start)

      # Listojen (avoimet ja läpikäydyt) alustus
      open = PathFindingList()
      open += startRecord
      closed = PathFindingList()

      # Iteroidaan solmut läpi
      while length(open) > 0:
            # Etsitään pienin elementti avointen listassa
            # estimTotalCost arvojen perusteella
            current = open.smallestElement()

            # Poistutaan silmukasta, jos nykyinen solmu
            # on maalisolmu
            if current.node == goal: break

            # Muuten haetaan kaaret (yhteydet muihin solmuihin)
            connections = graph.getConnections(current)
```

```
# Käydään kaaret läpi
for connection in connections:
      # Kustannusarvio solmuun kaaren päässä
      endNode = connection.getToNode()
      endNodeCost = current.costSoFar +
                        connection.getCost()

      # Jos solmu on läpikäytyjen listassa, se
      # poistetaan sieltä tai jätetään väliin
      if closed.contains(endNode):
            # Etsitään solmun tiedot
            endNodeRecord = closed.find(endNode)

            # Jätetään väliin, jos reitti ei parane
            if endNodeRecord.costSoFar <= endNodeCost:
                  continue:

            # Muuten poistetaan läpikäytyjen listasta
            closed -= endNodeRecord

            # Käytetään solmun vanhoja kustannusarvoja
            # heuristiikan laskentaan, jolloin vältetään
            # ylimääräinen funktiokutsu
            endNodeHeuristic = endNodeRecord.cost -
                                    endNodeRecord.costSoFar

      # Jos solmu avointen listassa, se jätetään
      # väliin, mikäli reitti ei parane
      else if open.contains(endNode):
            # Etsitään solmun tiedot
            endNodeRecord = open.find(endNode)

            # Jätetään väliin, jos reitti ei parane
            if endNodeRecord.costSoFar <= endNodeCost:
                  continue:

            # Lasketaan heuristiikka-arvo vastaavasti kuin
            # läpikäytyjen listan tapauksessa
            endNodeHeuristic = endNodeRecord.cost -
                                    endNodeRecord.costSoFar
```

```
                # Muuten kyseessä solmu, jossa ei ole käyty
                # Tehdään uusi tietue
                else:
                        endNodeRecord = new NodeRecord()
                        endNodeRecord.node = endNode

                        # Käytetään heuristiikkafunktiota, koska
                        # tietoja kustannuksista ei ole
                        endNodeHeuristic = heuristic.estimate(endNode)

                # Mikäli päästään tänne asti, päivitetään solmun
                # tietoja
                endNodeRecord.cost = endNodeCost
                endNodeRecord.connection = connection
                endNodeRecord.estimTotalCost = endNodeCost +
                                               endNodeHeuristic

                # Lisätään solmu avointen listaan, jos ei ole jo
                if not open.contains(endNode):
                        open += endNodeRecord

        # For-silmukka päättyy tässä nykyisen solmun osalta
        # Poistetaan nykyinen solmu avointen listasta ja lisätään
        # läpikäytyjen listaan
        open -= current
        closed += current

# While-silmukasta päästään tähän, kun kaikki solmut on
# käyty läpi tai maalisolmu on löytynyt
if current.node != goal:
        # Jos viimeisin käsitelty solmu ei vastaa haluttua
        # maalisolmua, ratkaisua ei löytynyt
        return None

else:
        # Muodostetaan lista, joka tulee sisältämään reitin
        path = []

        # Kerätään kaaret listaan lopusta alkuun
        while current.node != start:
```

```
            path += current.connection
            current = current.connection.getFromNode()

        # Käännetään listaelementtien järjestys ja palautetaan tulos
        return reverse(path)
```

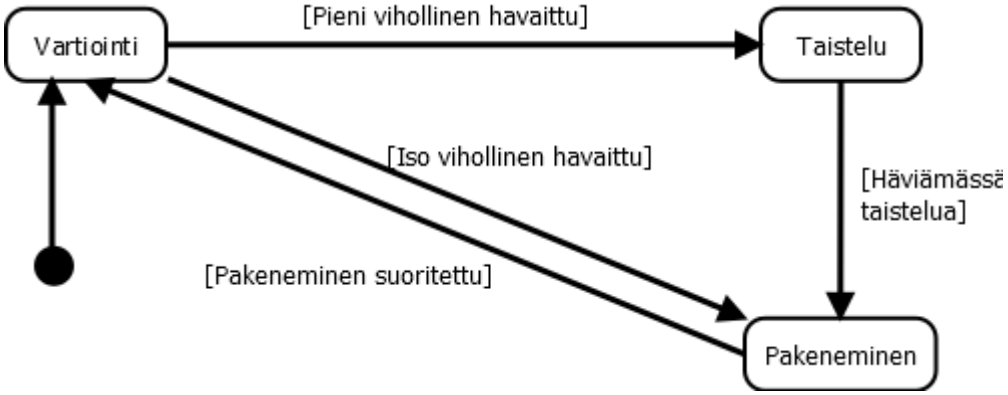
Dijkstran algoritmi on periaatteessa A* ilman heuristiikkaa. Täten se palauttaa lyhimmät reitit jokaiseen graafin solmuun. Koska reiteistä ei loppujen lopuksi tarvita kuin yhtä, voidaan tätä perinpohjaista läpikäymistä pitää resurssien tuhlaamisena, mikä on perimmäinen syy Dijkstran algoritmin vähäiselle käytölle pelinkehittäjien keskuudessa. [5] Siitä voi kuitenkin olla hyötyä, mikäli pelihahmon on esimerkiksi paikannettava jokin useista samanlaisista kohteista pelimaailmassa. Siinä missä A* -algoritmi olisi ajettava useamman kerran, Dijkstran algoritmilla yksi ajokerta riittäisi lähimmän esiintymän paikallistamiseksi. [3]

3.3 Päätöksenteko

Itsestään selvin tekoälyn tehtävistä lienee päätöksenteko. Erilaisia päätöksentekotekniikoita on olemassa lukuisia, mutta pohjimmiltaan ne kaikki toimivat samankaltaisesti. Algoritmit käyttävät syötteenä pelihahmon sisäistä tietoa sekä pelimaailmasta saatua ulkoista tietoa ja palauttavat toimenpiteen, joka halutaan suorittaa. Toimenpide voi olla vaikkapa hahmon liikuttaminen, jolloin pyyntö ohjataan tekoälyn polunetsinnästä ja liikkumisesta vastaaville tasoille, jotka edelleen välittävät sen omien operaatioidensa kautta fysiikkamoottorille tai animaatioteknologialle. Mikäli pelihahmo taas päättää esimerkiksi hyökätä, voidaan pyyntö välittää suoraan animaatioteknologialle, joka käynnistää sopivan hyökkäysanimaation. Kyseessä voi olla myös hahmon tai pelimaailman tilaa muuttava toimenpide, kuten haavojen parantaminen parannusjuomalla tai esineen poimiminen maasta. Tällaiset toiminnot ovat usein huomaamattomampia pelaajalle, sillä niitä ei ole tapana animoida tekoälyn kohdalla. [5]

Mahdollisesti käytetyin päätöksentekotekniikka peleissä lajityypistä riippumatta on äärellinen tilakone (engl. FSM, Finite State Machine). Äärelliset tilakoneet ovat yksinkertaisia ja nopeita toteuttaa sekä testata. Lisäksi ne mukautuvat helposti muihin tekniikoihin, ja niiden vaatima prosessointiteho on alhainen. [3] Yksinkertaisimmillaan

tilakoneiden voidaan katsoa koostuvan neljänlaisista elementeistä: tiloista, siirtymistä tilojen välillä, siirtymät laukaisevista säännöistä ja tapahtumista, jotka laukaisevat sääntöjen tarkistamisen [8]. Kuvassa 3 on esitetty mahdollinen tilakone yksikertaiselle tekoälyvastustajalle.

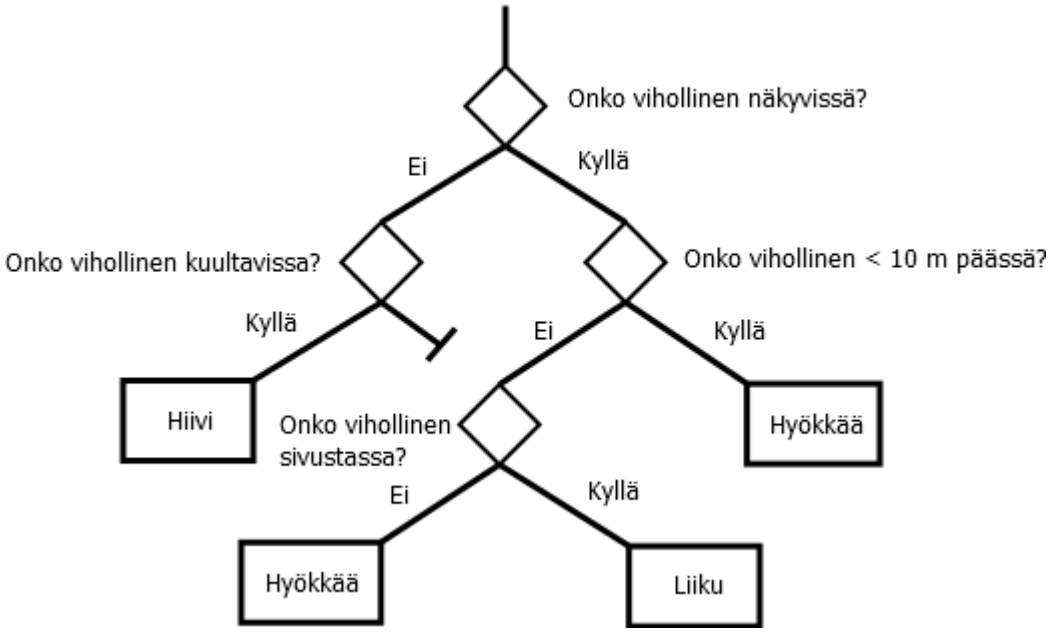


Kuva 3. Yksinkertainen tilakone [5]

Käytännössä tilojen määrä voi kasvaa hyvinkin suureksi, jolloin ei ole järkevää sisällyttää kaikkea yhteen tilakoneeseen. Ratkaisu tähän on rakentaa tietyille toiminnoille omat tilakoneensa, jotka toimivat varsinaisen tilakoneen alitiloina. Tällöin puhutaan hierarkkisesta tilakoneesta. Hierarkian avulla tilojen kokonaismäärä pysyy järkevänä, ja toimintojen priorisointi poikkeustilanteissa sekä niiden jälkeen on selkeämpää. [5]

Toinen suosittu päätöksentekotekniikka on päätöspuiden käyttäminen. Äärellisten tilakoneiden tavoin päätöspuut ovat helppoja ja nopeita toteuttaa, mutta voivat kasvaa nopeasti hyvinkin suuriksi. Puun läpikäyminen alkaa juuresta, jossa tarkastellaan jonkin ehdon toteutumista, tehdään päätös ja haaraudutaan sen perusteella johonkin alemman tason solmukohdista. Solmukohdassa tehdään jälleen uusi päätös ja edetään vastaavasti seuraavalle tasolle. Solmukohtien läpikäymistä jatketaan, kunnes päädytään suoritettavaan toimintoon puun latvassa. [5] Yksi esimerkki päätöspuusta on nähtävissä kuvassa 4.

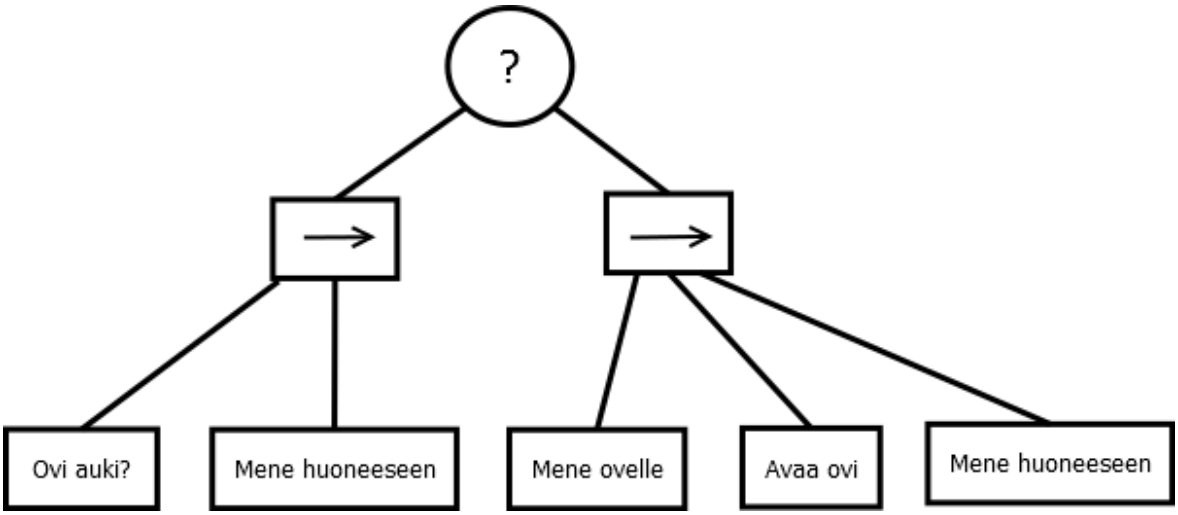
Päätöksentekoa voidaan mallintaa myös käytöspuiden avulla. Päätöspuiden tavoin käytöspuissa edetään tarkastelemalla ehtojen toteutumista ja suorittamalla sitten asiaankuuluvat toimenpiteet. Erona on kuitenkin se, että siinä missä päätöspuiden kohdalla päädytään lopulta yhteen suoritettavaan toimintoon, käytöspuissa erilaisia toimintoja



Kuva 4. Yksinkertainen päätöspuu [5]

voidaan ketjuttaa. Tämä tapahtuu koostesolmujen (engl. composite node) avulla, joita on kahta tyyppiä: valitsin- ja sarjakooste. Valitsinkooste (merkitään ?) haarautuu toimenpiteisiin, joita valitaan yksi kerrallaan suoritettavaksi, kunnes jokin niistä onnistuu ja valintatehtävä voidaan lopettaa näin onnistuneena. Mikäli yksikään toimenpide ei onnistu, valintatehtävä on luonnollisesti epäonnistunut. Sarjakooste (merkitään →) taas haarautuu ehtojen tarkistamiseen ja toimenpiteisiin, jotka on suoritettava peräkkäin onnistuneesti. Jos yksikin ehto tai toimenpide epäonnistuu, sarjatehtävä on epäonnistunut. [5] Kuvassa 5 on esitetty yksinkertainen käytöspuu, jonka mukaan pelihahmo voisi toimia kohdatessaan oviaukon.

Usein tilakoneissa uuteen tilaan siirtyessä tai puurakenteissa seuraavaan kohtaan edetessä tarkasteltava seikka ei ole selkeästi tosi tai epätosi, vaan jotakin siltä väliltä. Otetaan esimerkiksi autopeli, jossa tekoälyn ohjaaman kuskin on jarrutettava lähestyessään mutkaa. Pelkkä tieto siitä, jarrutetaanko vai ei, ei kuitenkaan riitä, vaan on tiedettävä kuinka kovaa jarrutetaan. Loivissa mutkissa aina täydellä teholla jarruttava tekoälykuski ei anna kovin älykästä kuvaa itsestään. Tästä syystä monissa peleissä käytetään esimerkiksi sumean logiikan (engl. fuzzy logic) kaltaista järjestelmää, jossa lasketaan arvoja (esim. välillä 0-1), jotka kuvaavat kuinka paljon jokin ratkaistava asia kuuluu tietyn päätöksen piiriin. Esimerkin tapauksessa voitaisiin laskea, kuinka paljon tietyntyyppiseen mutkaan ajaminen



Kuva 5. Yksinkertainen käytöspuu [5]

kuuluisi jarrutuspäätöksen piiriin (loivan mutkan tapauksessa arvo voisi olla 0,2), ja sen perusteella määrätä jarrutusvoima, eli tässä vaikkapa 20 % kokonaisjarrutusvoimasta. [5]

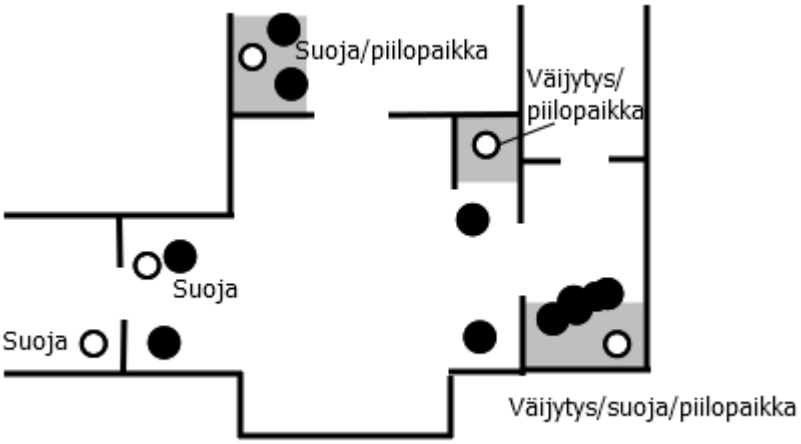
Aiemmin kuvattujen tekniikoiden ohella suosiota saanut tapa tekoälyn päätöksenteon toteuttamiselle on komentosarjojen, eli skriptien käyttö. Tärkeimpänä syynä komentosarjojen suosiolle on se, että ne helpottavat huomattavasti koodin muokkaamista ja testaamista. Esimerkiksi äärellisen tilakoneen tapauksessa pääpelikoodiin kirjatut tilat voidaan korvata erillisillä ajettavilla skripteillä, jolloin pääpelikoodia ei tarvitse kääntää joka kerta uudestaan, ja pelihahmon käyttäytymistä on näin helppo säätää. Lisäksi komentosarjakielet ovat monesti helppotajuisia, mikä mahdollistaa ei-ohjelmointitaustaisten henkilöiden osallistumisen säätöjen tekemiseen. Samoista syistä skriptejä käytetään paljon myös muilla pelien osa-alueilla, kuten kameran, äänten tai animaation ohjaamisessa. [3]

3.4 Taktiikka ja strategia

Pelitekoälyn mallissa strategiatason tehtävänä on huolehtia suuremman mittakaavan tavoitteiden asettamisesta ja saavuttamisesta. Käytännössä strategiatasolla tehdään siis päätelmiä pelitilanteesta ja välitetään tätä taktista tietoa eteenpäin yhden tai useamman pelihahmon käytettäväksi suoraan päätöksenteossa tai toisinaan polunetsinnässä. Taktinen tieto voi olla esimerkiksi tietoa pelikentän taktisista sijainneista, kuten suojapaikoista, tai

tietoa maaston luonteesta tai vihollisen vaikutusalueesta. Taktisen tarkastelun mahdollistamiseksi pelikenttä esitetään yleensä joko taktisten reittipisteiden tai taktisen analysointikartan muodossa. [5]

Polunetsinnän yhteydessä pelikenttä jaettiin solmuihin, jotka olivat yhteydessä toisiinsa kaarien avulla. Strategisella tasolla näitä solmuja kutsutaan reittipisteiksi (engl. waypoint). Olennaisena erona polunetsinnässä käytettyihin solmuihin on kuitenkin se, että nyt pisteet voivat sisältää tietoa sijaintinsa taktisista ominaisuuksista. Taktiset reittipisteet voivat olla esimerkiksi suoja, piilo- tai väijytyspaikkoja. Reittipiste voi luonnollisesti omata useita taktisia ominaisuuksia samanaikaisesti, eli esimerkiksi hyvä suojapaikka voi samanaikaisesti olla myös loistava piilopaikka. Taktiset ominaisuudet asetetaan yleensä suoraan kenttäsuunnittelijan toimesta tai luodaan laskennallisesti. Vaikka taktista reittipistegraafia on toisinaan mahdollista hyödyntää myös polunetsinnässä, Funge ja Millington suosittelevat käyttämään erillisiä graafeja polunetsintään ja reittipisteiden analysointiin. [5] Syy tähän nähdään hyvin kuvassa 6, jossa taktiset reittipisteet on merkattu valkoisilla palloilla.

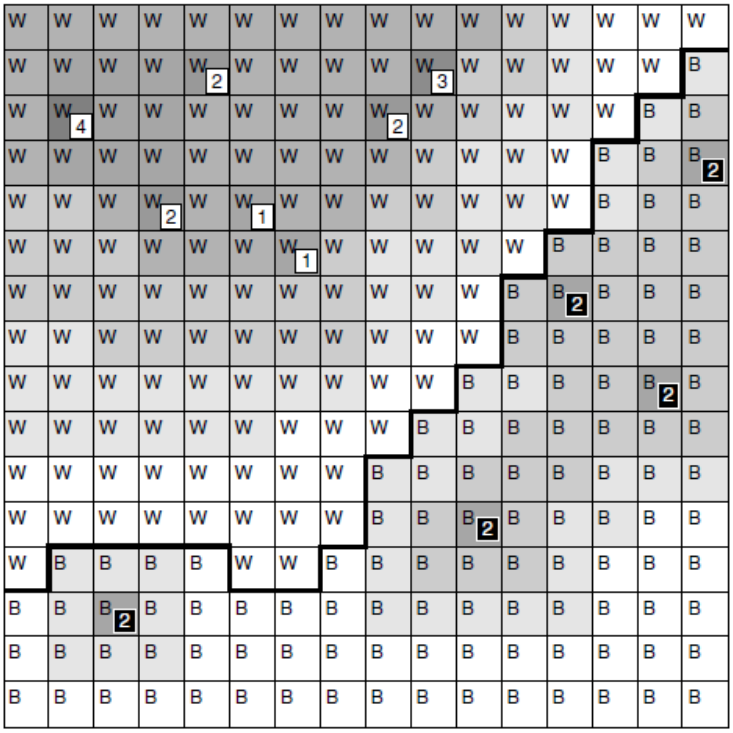


Kuva 6. Taktiset reittipisteet [5]

Muita, pitkälti reittipistejärjestelmän kaltaisia, tapoja saada tietoa pelikentän taktisesta tilanteesta ovat taktiset analyysit, joista käytetään pääasiassa vaikutusaluekarttaa (engl. influence map) ja maastoanalyysiä (engl. terrain analysis). Vaikutusaluekartta kertoo esimerkiksi reaaliaikaisessa strategiapelissä yksinkertaisesti tekoälyn ja pelaajan ohjaamien joukkojen vaikutuksen kussakin pelikentän osassa. Vaikutuksen suuruuden tietyllä alueella

voi pelistä riippuen määrätä esimerkiksi joukkojen läheisyys, määrä, aseistus tai kunto. Maastoanalyysissa taas on tarkoituksena selvittää laskennallisesti maaston ominaisuuksien vaikutus alueittain. Tarkasteltavia ominaisuuksia voivat olla muun muassa maaston vaikeakulkuisuus ja paikan aukeus. Kummankin analysointitavan kohdalla pelikentän osittamiseen voidaan käyttää polunetsinnän puolelta tuttuja tekniikoita. [5]

Kuvassa 7 on esitetty vaikutusaluekartta reaaliaikastrategianpelin pelikentästä. Kentällä on kaksi joukkuetta, mustat ja valkoiset, joilla on kummallakin joukkoja kentän eri alueilla. Kuvassa näkyvät numerot kuvaavat joukkojen määrää näillä alueilla. Jokainen alue on kuitenkin joko mustien (B) tai valkoisten (W) vaikutuksen alla riippumatta siitä, onko juuri siinä kohtaa joukkoja vai ei. Mitä tummempi tietyn joukkueen hallitsemaa aluetta kuvaava ruutu on, sitä voimakkaampi joukkojen vaikutus on siellä. Kuvaan on piirretty myös hallinta-alueiden raja.



Kuva 7. Vaikutusaluekartta [5]

Monesti strategisella tasolla tehdyt päätökset koskevat useita tekoälyhahmoja, jotka toimivat yhteistyössä jonkin tavoitteen saavuttamiseksi. Tällöin tarvitaan monikerroksista tekoälyä, jossa kullakin pelihahmolla on oma tekoälynsä, mutta ne noudattavat samaa

suuremman mittakaavan suunnitelmaa. Toisinaan tekoäly toimii kuitenkin yhdessä pelaajan kanssa, jolloin strategia tulee suoraan pelaajalta tai epäsuorasti pelaajan aikeita ennakoimalla. Varsinkin jälkimmäisessä tapauksessa monikerroksinen lähestymistapa voi aiheuttaa ongelmia, mikäli niitä ei oteta huomioon esimerkiksi kenttäsuunnittelussa. Monikerroksinen tekoäly voidaan kuitenkin toteuttaa monella tavalla, joten oikean arkkitehtuurin valitseminen helpottaa myös pelaajan sovittamista kuvioon. [5]

3.5 Vuorovaikutus pelimaailman kanssa

Useimmat tekoälyalgorimit hyödyntävät pelihahmon sisäisen tiedon, kuten varustuksen, terveystilanteen tai liiketilan, ohella ulkoista, pelimaailmasta saatua tietoa, kuten muiden pelihahmojen sijaintia tai ovien tilaa (auki vai kiinni). Mikäli tekoälystä halutaan tehdä uskottava, tätä ulkoista tietoa ei voida antaa suoraan tekoälyn ohjaamille hahmoille, vaan pelissä on oltava jonkinlainen tapa välittää sitä tasaisin väliajoin tekoälyn eri osille. Näin saadaan aikaan illuusio siitä, että pelihahmo havainnoi ympäristöään ja reagoi havaintoihinsa. [5]

Funge ja Millington esittelevät kirjassaan kaksi tekniikkaa, joita käytetään tekoälyn ja pelimaailman välisessä vuorovaikutuksessa: kiertokysely (engl. polling) ja tapahtumien välitys (engl. event passing). Kiertokysely toimii siten, että tekoälyhahmot tekevät jatkuvalla syötöllä kyselyitä niitä kiinnostavista kohteista pelimaailmassa. Tekoälyhahmojen määrän kasvaessa käytetään keskusta, jonka läpi kyselyt kulkevat, jotta niitä voidaan paremmin hallita. Tapahtumien välitys taas toimii siten, että erillinen koodinpätkä tekee tarkistuksia pelimaailman tilasta ja välittää tiedon kiinnostavista tapahtumista asianosaisille tekoälyhahmoille. Lähestymistavan valinta riippuu sovelluksesta: joskus kiertokysely on parempi vaihtoehto, toisinaan taas tapahtumien välitys. Myös yhdistelmät ovat mahdollisia, tosin hankalampia hallita. [5]

3.6 Suorituksen hallinta

Koska käytettävissä oleva prosessoriaika on rajallinen, on varsinkin monimutkaisempien tekoälyjen tapauksessa tekoälyn eri osa-alueiden tehtäviä aikataulutettava. Aikataulutus tapahtuu jakamalla käytössä oleva prosessoriaika kehyksittäin sitä tarvitsevien algoritmien

kesken erillisen suorituksenhallintakoodin (aikatauluttajan) avulla. Saadun prosessoriajan määrä vaihtelee tekoälyn osien välillä riippuen niiden prioriteetista. Suoritustaajuus, eli kuinka monen kehyksen välein jotakin tehtävää suoritetaan, voi olla erikseen määritelty ja/tai pohjautua prosessoriajan tavoin tehtävän prioriteettiin. Mikäli on hallittava suurta joukkoa itsenäisesti toimivia tekoälyhahmoja, aikatauluttajia on useita. Tällöin ne muodostavat hierarkian, jonka huipulla on yksittäisten hahmojen aikatauluttajien toimintaa koordinoiva aikatauluttaja. [5]

Huomionarvoinen asia aikataulutettaessa on myös se, että usein on tarpeen huolehtia vain pelaajan havaintoalueella olevien tekoälyhahmojen toiminnan uskottavuudesta. Tietyllä etäisyydellä pelaajasta oleville tekoälyhahmoille annetaan tällöin vapaus liikkua epäuskottavasti tai kävellä seinien läpi ja säästetään näin arvokasta prosessoriaikaa. Niin kauan kuin pelaaja ei näe tätä outoa käyttäytymistä, illuusio siitä, että hahmot tekevät pelaajan poissa ollessakin jotain järkevää, säilyy. [5]

Aikataulutuksen ohella suorituksen hallintaan liittyvät olennaisesti erilaiset huijaustekniikat, joihin viitattiin lyhyesti luvussa 2.3. Huijaustekniikoita käytetään etenkin reaaliaikastrategiapeleissä, joissa monikerroksinen tekoäly raskaine polunetsintä- ja analysointimenetelmineen tuottaa monesti suorituskykyongelmia. Tyypillisiä huijaustekniikoita niissä ovat esimerkiksi sellaisen tiedon hankkiminen, jota pelaaja ei voisi hankkia, tyhjästä ilmestyvät vastustajajoukot sekä pelin sääntöjen hienovarainen kiertäminen vaikkapa suomalla tekoälylle ilmaisia varusteita tai rakennuksia. Muista peligenreistä mainittakoon autoilupelit, joiden yhteydessä taas puhutaan usein ns. kuminauhatekoälystä, jolla tarkoitetaan tekoälykuskien tapaa pysytellä mukana pelaajan vauhdissa riippumatta siitä, kuinka hyvin tämä ajaa.

Huijaustekniikoilla pyritään yleensä antamaan tasoitusta huomattavasti tietokonetta älykkäämpää ihmispelaajaa vastaan ja korvaamaan näin varsinaista älykkyyttä osoittavia tekniikoita hyvän suorituskyvyn ylläpitämiseksi. Esimerkiksi kuminauhatekoälyn tapauksessa sen sijaan, että tekoälykuskit hakisivat älykkäästi optimaalista ajolinjaa pelaajan voittamiseksi, ne ajavat aina tiettyä ajolinjaa ja autoa nopeutetaan tai hidastetaan pelaajan etäisyydestä riippuen. Hyvin toteutettuna tekoälyn huijaaminen voi tarjota

tasaisen vastuksen pelaajalle ja nostaa tekoälyn täysin uudelle tasolle. Huonosti toteutettuna se taas voi pilata illuusion älykkyydestä ja laskea pelin viihdyttävyysarvoa merkittävästi.

3.7 Työkaluketjut

Työkaluketjut tarjoavat pelikehittäjille välineet sisällönluontiin, kuten äänten, animaation, tekstuurien ja mallien tuottamiseen. Erilaiset työkaluketjut ovat tärkeä osa nykyaikaista pelikehitystä, sillä ne standardoivat sisällönluomisprosessin niin, että sitä voidaan hyödyntää useissa peleissä. Työkaluketjut ovat tärkeässä osassa myös tekoälyn rakentamisessa. Jotta muun muassa polunetsintä- ja liikkumisalgoritmeja voitaisiin hyödyntää tehokkaasti, niiden on saatava tarvitsemansa tieto oikeassa muodossa. Esimerkiksi luvussa 3.2 puhuttiin polunetsinnän yhteydessä kvantisoinnista, jossa pelikenttä yksinkertaistettiin graafin muotoon. Tämänkaltainen sisällönluonti tekoälyä varten tapahtuu juuri hyödyntämällä työkaluketjuja. [5]

Työkaluketjut nopeuttavat huomattavasti tekoälyn kehitystyötä huolehtimalla sisällönluonnista ja tarjoamalla pohjan tekoälyn kehittämiselle. Ne sisältävät valmiita työkaluja esimerkiksi tekoälyn suunnitteluun (esim. tilakoneiden laatiminen) ja toteuttamiseen (esim. skriptaustyökalut) sekä virheidenetsintään. Valmiiden työkalujen ansiosta tekoälyn säätäminen ei ole enää pelkästään ohjelmoijien harteilla, vaan siitä ovat yhtä lailla vastuussa muun muassa kenttäsuunnittelijat. [5]

4 TEKOÄLYN HAASTEET JA MAHDOLLISUUDET

Luvussa 3 esitellyt lähestymistavat ja tekniikat antavat hyvät lähtökohdat tekoälyn toteuttamiselle, mutta niihin liittyy omat rajoitteensa ja ongelmansa. Tekninen puoli ei kuitenkaan ole ainoa ongelmien aiheuttaja. Hyvän tekoälyn toteuttaminen tietokonepeliin on monivaiheinen prosessi, jossa on otettava huomioon monia asioita aina laitteistotasolta pelinkehitysprojektin tasolle asti. Tämä tuo luonnollisesti mukanaan joukon uusia haasteita.

Haasteista huolimatta tekoälyä kehitettäessä sovelletaan toisinaan kokeellista teknologiaa ja epätavallisempia tekniikoita. Hyvänä esimerkkinä tästä ovat akateemisen tekoälyn puolelta tuttuja oppimisalgoritmeja hyödyntävät lähestymistavat, jotka mahdollistavat muun muassa tekoälyhahmojen mukautumisen tietyn pelaajan pelityyliin. Vaikka toimivia ratkaisuja löytyy lähinnä indiepelien puolelta (esimerkiksi geneettisiä algoritmeja käyttävä Warning Forever), lähestymistavat kertovat kuitenkin tekoälyn potentiaalista, joka myös suurempien pelinkehittäjästudioiden on syytä huomioida.

Tässä luvussa tarkoituksena on luoda kokonaiskuva pelitekoälyn haasteista ja mahdollisuuksista nyt sekä tulevaisuudessa. Kokonaiskuvan avulla pyritään saamaan jonkinlainen käsitys siitä, missä pelitekoälyn rajat lopulta kulkevat. Viimeisessä aliluvussa 4.3 kootaan yhteen tehdyt päätelmät ja esitetään johtopäätökset tutkimusongelman suhteen.

4.1 Rajoitteet ja ongelmat

Tekoälyn toteuttamiseen liittyvät rajoitteet ja ongelmat voidaan jakaa kolmeen luokkaan: tekniset rajoitteet, osaamiseen liittyvät rajoitteet ja pelinkehitysprojektien asettamat rajoitteet. Jako perustuu pitkälti kirjallisuudessa ja pelitekoälyn kehittäjien lausunnoissa esille nousseisiin seikkoihin. Seuraavassa esiteltävistä rajoitteista suurin osa tullee siis väistämättä vastaan etenkin suuren budjetin pelinkehitysprosessien yhteydessä.

Suurin ja selkein rajoittava tekijä tekoälyä rakentaessa on tekniset rajoitteet, joka käsittää laitteiston asettamat rajoitukset ja algoritmien rajoitteet. Laitteistopuolelta tekoälyn käyttöön annetut resurssit, suoritinaika ja muisti, ovat luonnollisesti rajalliset. Fungen ja

Millingtonin mukaan tekoälylle varattu suoritinaika on tyypillisesti noin 20 % kokonaissuoritinajasta, joskus se voi kuitenkin olla jopa 50 % tai enemmän. Yksinkertaisen tekoälyn tapauksessa suoritinaikaa saattaa olla riittävästi, mutta monimutkainen, useista kerroksista ja lukuisista aikaavievistä algoritmeista koostuva tekoäly voi äkkiä laskea suorituskyvyn pohjalukemiin, jos sen ohella ajetaan vielä esimerkiksi fysiikkamallinnusta. Tällöin tärkeässä osassa on suorituksen hallinta, josta puhuttiin luvussa 3.6. Taitavalla suoritinresurssien hallinnalla monimutkainenkin tekoäly voidaan saada toimimaan sulavasti. Suorituskykyä voidaan parantaa myös hyödyntämällä nykyaikaisten moniydinprosessorien ominaisuuksia rakentamalla koodi siten, että tekoälyn eri osia on mahdollista ajaa rinnakkain. [5]

Muistin osalta ongelma ei ole Fungen ja Millingtonin mukaan niinkään muistin rajallinen määrä, vaan siihen käsiksi pääseminen. Koska tiedon hakeminen muistista vie huomattavasti aikaa, prosessori operoi välimuistiin kopioiduilla varsinaisen muistin tiedoilla. Varsinainen muisti pidetään ajan tasalla lähettämällä välimuistin tiedot sinne prosessorin suoritussyklien välillä. Välimuisti sisältää kuitenkin vain osan varsinaista muistia, joten mikäli tarvittavaa tietoa ei löydy sieltä, se joudutaan noutamaan varsinaisesta muistista, mikä saattaa seisauttaa tekoälyn toiminnan. Tekoälyalgoritmien suorituskyvyn kannalta on siis tärkeää, että oikea tieto löytyy välimuistista oikeaan aikaan. [5]

Käytettävissä olevan suoritinajan maksimoiminen ei kuitenkaan ratkaise varsinaista ongelmaa, joka on Fungen ja Millingtonin mukaan se, että käytössä olevat algoritmit ovat yksinkertaisesti hitaita [5]. Runsaasti suoritinaikaa kuluttavat algoritmit, kuten polunetsintä ja strateginen päättely pyritään viilaamaan mahdollisimman tehokkaiksi. Jokaisella pelinkehittäjällä lienee esimerkiksi oma optimoitu versionsa suositusta A* -algoritmista. Tästä huolimatta tekoälyä joudutaan monesti yksinkertaistamaan suorituskyvyn parantamiseksi. Esimerkiksi reaaliaikastrategiapelissä Total War: Rome II (The Creative Assembly, 2013) hyödynnettiin polunetsinnässä approksimoituja reittejä ja strategisella puolella valmiiksi laskettuja vaikutusaluekarttoja [1]. Tämänkaltainen yksinkertaistaminen ei vielä vaikuta juurikaan tekoälyn laatuun, mutta jos tekoälyn on sorruttava esimerkiksi huijaamiseen suorituskyvyn takaamiseksi, pelaaja saattaa huomata tämän hyvin nopeasti. Tällöin pelin viihdyttävyys luonnollisesti kärsii.

Pelinkehittäjien osaamiseen liittyvät rajoitteet on mahdollista kyseenalaistaa rajoitteina: mikä estää tekoälyn toteuttamisesta vastaavia opettelemasta monimutkaisempaa teknologiaa niin, että heidän osaamistaan voidaan hyödyntää tekoälyn parantamisessa? Ongelma on kuitenkin siinä, että hienostuneempien tekoälytekniikoiden opettelu vie aikaa, jota suurilla pelistudioilla ei tiukkojen julkaisuaikataulujen takia yksinkertaisesti ole. Usein monimutkaisemmista tekniikoista ei myöskään ole juuri käytännön kokemusta sattuneesta syystä. Täten turvallisempi vaihtoehto on käyttää yksikertaisempia lähestymistapoja, jolloin tekoäly toimii keskinkertaisesti mutta riittävän sujuvasti pitääkseen pelaajat tyytyväisinä.

Myös pelinkehitysprojekti asettaa omat rajoitteensa tekoälylle. Tekoälyn rakentaminen on vain yksi osa pelinkehitystyötä, ja käytössä olevat rahalliset sekä ajalliset resurssit ovat rajalliset. Kuten luvussa 3.7 todettiin, monet pelistudiot hyödyntävät lähes poikkeuksetta erilaisia työkaluketjuja kehitystyössään. Koska käytetyt työkalut sanellaan projektitasolla, tämä rajoittaa tekoälyn toteuttamisessa käytettäviä tekniikoita [5]. Tästä syystä esimerkiksi neuroverkkoihin perustuvaa oppivaa tekoälyä ei juurikaan hyödynnetä kaupallisissa peleissä.

4.2 Nykytila, mahdollisuudet ja tekoälyn tulevaisuus

Peleihin keskittyvän web-sivusto GameSpotin vuonna 2010 julkaistussa artikkelissa esitetyt väitteet tekoälyn tilasta peleissä pätevät edelleen melko hyvin. Useimpien kaupallisten pelien tekoäly katsotaan riittäväksi silloin, kun se ei ole selkeästi puutteellinen. Taustalla väitetään olevan se, että koska pelaajat harvoin kiinnittävät huomioita erityisen hyvin toimivaan tekoälyyn, uusiin innovaatioihin ei kannata tuhlata aikaa ja rahaa. [6] Tämänkaltainen ajattelu näkyy monien suuren budjetin pelien tekoälyssä.

Tekoälyvastustajien tarjoama haaste pelaajalle on usein vähäinen: esimerkiksi räiskintäpeleissä vihollisilla on tapana olla enemmänkin ns. tykinruokaa, kuin varsinaisesti haastaa pelaaja käyttämällä vaihtelevia taktiikoita taistelutilanteissa. Mikäli haastetta on, sitä yritetään yleensä tarjota nostamalla vastustajien sietokykyä ja/tai pelaajan luodeista kärsimää vahinkoa sen sijaan, että parannettaisiin itse tekoälyä. Myös pelaajan puolella

olevien hahmojen toiminta herättää monesti pelaajissa vastareaktioita. Varsinkin urheilupeleissä tekoälyn ohjaamien joukkuetovereiden toimintaa kritisoidaan siitä, että ne eivät osaa toimia pelaajan aikeiden mukaisesti, kuten hakeutua syöttöpaikkoihin tai edetä hyökkäyksissä pelaajan haluamalla tavalla. Pelaajan toimintaa tukevien hahmojen nykytilasta voidaan kuitenkin todeta, että se on useimmiten satunnaisia bugeja lukuun ottamatta kiitettävällä tasolla. Esimerkiksi Grand Theft Auto V:n (Rockstar, 2013) hiekkalaatikkopelin maailmaa kansoittavat sivulliset hahmot ovat saaneet kehuja kehitysyhteisöltä [2].

Vaikka iso osa pelistudioista pelaa varman päälle tekoälyn suhteen, on olemassa myös poikkeuksia, kuten Halo- ja F.E.A.R. -pelisarjat, joiden tuomat innovaatiot mm. tekoälyvastustajien taistelun toteuttamiseen uudistivat aikoinaan pelinkehittäjien käsityksiä aiheesta. Nykyisin tekoälyn mahdollisuuksista antavat tasaisin väliajoin osviittaa myös erilaiset indiepuolen pelit, jotka rakentuvat tyypillisesti jonkin tietyn innovaation varaan. Tällainen innovaatio voi olla esimerkiksi pelaajan pelityyliin sopeutuvat tai sen oppivat tekoälyvastustajat tai mahdollisuus kommunikoida luonnollisella kielellä interaktiivisesti ei-pelaajahahmojen kanssa. Usein näistä innovaatioista haluttaisiinkin ottaa hyöty irti kaupallisten pelien puolella. Etenkin oppiva tekoäly nähdään monessa pelitalossa yhtenä tämän hetken suurista mahdollisuuksista, koska sillä on potentiaalia parantaa pelin uskottavuutta ja viihdyttävyyttä [5].

GameSpotin haastattelemat pelialan vaikuttajat uskovat pelien olevan tulevaisuudessa immersiivisempiä ja tarjoavan interaktiivisemman pelikokemuksen teknologian, työkalujen ja algoritmien kehittyessä [6]. Sopivan älykkäiden tekoälyvastustajien ja järkevästi toimivien pelaajan puolella olevien tekoälyhahmojen rakentaminen tulee kuitenkin luultavasti olemaan yksi keskeisimmistä haasteista myös tulevaisuudessa. Mutta mikäli oppimisalgoritmeja opitaan hyödyntämään laajemmin, tähän tarvittava koodaustyö voi helpottua huomattavasti.

4.3 Johtopäätökset

Luvussa 2.3 määriteltiin hyvän tekoälyn kriteereiksi viihdyttävyys ja uskottavuus, jotka sen on erilaisissa rooleissa pelissä kyettävä täyttämään. Monissa nykypeleissä

uskottavuuspuoli vaikuttaisi olevan hyvällä mallilla. Yhä harvemmin nähdään toisiinsa puutteellisen liikkumistoteutuksen takia törmäileviä tai yllättävän päätöksentekotilanteen tulessa paikalleen jämähtäviä pelihahmoja. Myös kommunikointi pelihahmojen kanssa on viime vuosina muuttunut monipuolisemmaksi. Ilmaukset valitaan edelleen valmiista vaihtoehdoista, mutta niiden sisältö ja pelihahmon asenne pelaajaa kohtaa voivat riippua aiemmista keskusteluista tai pelaajan toimista pelimaailmassa. Teoriassa uskottavuutta voisi parantaa vielä tekoälyn kyky oppia ja sopeutua pelaajan pelityyliin, mutta käytännössä oppimistekniikoiden soveltaminen vaatii luvussa 4.1 esiteltyjen ongelmien ratkaisemista.

Useimpia nykypelejä voidaan pitää yleisesti ottaen viihdyttävinä. Iso osa viihdyttävyydestä rakentuu kuitenkin monessa pelissä muiden asioiden, kuten näyttävän grafiikan tai kiehtovan tarinan varaan tekoälyn toimiessa sulavasti mutta keskinkertaisesti taustalla. On kuitenkin myös paljon pelejä, joissa nimenomaan tekoälyn viihdyttävyys korostuu pelin onnistuneisuuden määrittävänä tekijänä. Tällöin nousevat parhaiten esiin myös tekoälyn toteuttamiseen liittyvät ongelmat. Nykytiedon ja -tekniikoiden puolesta pelinkehittäjät voisivat saada aikaan hyvinkin monipuolista ja hienostunutta tekoälyä, mutta hyvät yritykset kaatuvat usein työn monimutkaisuuteen. Vaikka tekniikoiden ymmärtämiseen liittyvät haasteet voitettaisiin, pelaajan puolella olevien ja pelaajaa vastaan kilpailevien pelihahmojen tasapainottaminen oikean haastavuustason löytämiseksi on edelleen hankalaa. Kun samaan aikaan on huolehdittava vielä kohtuullisen suorituskyvyn ylläpitämisestä, täydellisen tekoälyn luomisen voidaan sanoa olevan mahdotonta.

Kuinka hyvä tekoäly sitten on mahdollista toteuttaa? Teoriassa, mikäli kaikki menee suunnitelmien mukaisesti, on mahdollista rakentaa tekoäly, joka luo vakuuttavan illuusion älykkyydestä kommunikoimalla pelaajan kanssa uskottavasti monipuolisella keskustelujärjestelmällä, toimimalla uskottavasti ja järkevästi pelimaailmassa lähes kaikissa tilanteissa ja tarjoamalla sopivasti haastetta sekä vaihtelua sopeutumalla pelaajan pelityyliin. Käytännössä suorituskykyrajoitteet, tekijöiden osaamiseen liittyvät rajoitteet ja pelinkehitysprojektien asettamat rajoitteet kuitenkin hankaloittavat toteutusprosessia siinä määrin, että ainakaan lähivuosina tähän ei aivan päästä, ja puutteita tulee löytymään aina joltakin osa-alueelta.

Tässä vielä koottuna tärkeimmät havainnot koskien pelitekoälyjen hyvyyttä:

- Suurin osa peleistä onnistuu luomaan riittävän vakuuttavan illuusion älykkyydestä täyttäen näin kiitettävästi vaatimuksen tekoälyn uskottavuudesta.
- Tekoälyn rakentamisessa ei usein oteta riskejä, minkä seurauksena tekoäly toimii sulavasti mutta keskinkertaisesti.
- Oikean haastavuustason löytäminen pelaajan puolella oleville ja pelaajaa vastaan kilpaileville pelihahmoille ja sitä kautta viihdyttävämmän pelikokemuksen tarjoaminen on yksi suurimmista haasteista tekoälyn rakentamisessa.
- Tiedossa olevat perustekniikat yhdistettynä epätavallisempiin tekniikoihin riittäisivät teoriassa nykyistä viihdyttävämmän tekoälyn toteuttamiseen. Tekoäly voisi esimerkiksi sopeutua pelaajan pelityylin ja tarjota näin tasaisesti haastetta läpi pelin.
- Epätavallisempia tekniikoita on kuitenkin hankalaa soveltaa suuremmassa mittakaavassa, sillä suorituskykyongelmat, osaamisen puute ja pelinkehitysprosessien asettamat rajoitteet voivat tuottaa ylitsepääsemättömiä ongelmia.

5 YHTEENVETO

Toimivan tekoälyn rakentaminen tietokonepeliin on monen tekijän summa. Kaiken pohjana on joukko yhteen sovitettuja algoritmeja ja tekniikoita, jotka toteuttavat pelihahmojen strategisen ajattelun sekä päätöksenteon ja sitä kautta polunetsinnän sekä liikkumisen tai muiden toimenpiteiden hallinnan. Parhaan suorituskyvyn mahdollistamiseksi algoritmit pyritään hiomaan huippuunsa käyttämällä sopivia heuristiikkoja ja optimointeja sekä hyödyntämällä tarvittaessa ad hoc -ratkaisuja sekä huijauksia. Tärkeä osa suorituskyvyn ylläpitoa on myös suoritinresurssien hallinta ajastamalla tekoälyn suorittamat osatehtävät kehyksittäin. Varsinainen kehitystyö tapahtuu usein hyödyntämällä valmiita työkaluketjuja, joiden avulla suunnittelu, virheiden etsintä ja sisällönluonti algoritmeja varten helpottuvat huomattavasti.

Tekoälyn toteuttamista rajoittavia tekijöitä ovat tekniikkaan, tekijöiden osaamiseen sekä pelinkehitysprojekteihin liittyvät rajoitteet. Tietokoneiden rajalliset resurssit sekä käytettävien algoritmien yleinen hitaus aiheuttavat monesti ongelmia optimoinnista huolimatta, mikä johtaa tekoälyn yksinkertaistamiseen. Tekijöiden osaaminen rajoittuu kokemuksen puutteen takia tavallisimpiin tekniikoihin, jolloin hienostuneempia tekniikoita ei saada hyötykäyttöön suuren budjetin peleissä. Pelinkehityksen projektiluonteesta johtuen aikaa ja rahaa on käytettävissä rajallisesti, joten uusien tekniikoiden opettelu ei kannata. Myös työkaluketjujen tärkeys nykyisissä peliprojekteissa rajoittaa tekoälyä, sillä se käytännössä sanelee, minkä tekniikoiden pohjalta tekoälyä lähdetään kehittämään.

Tekoäly toimii peleissä erilaisissa rooleissa joko pelaajan puolella, pelaajaa vastaan tai sivullisena hahmona. Hyväksi tekoälyksi voidaan määritellä sellainen tekoäly, joka täyttää viihdyttävyyden ja uskottavuuden kriteerit roolista riippumatta. Nykypeleissä uskottavuus toteutuu useimmiten kiitettävästi, mutta viihdyttävyys tökkii etenkin tekoälyvastustajien tarjoaman heikon haastavuustason sekä tekoälykumppaneiden ja pelaajan usein ristiriitaisten tavoitteiden takia. Teoriassa on mahdollista kehittää tekoäly, joka selviytyy lähes joka tilanteessa ja pystyy sopeutumaan pelaajan pelityyliin, mutta käytännössä tämä vaatii aiemmin kuvattujen rajoitteiden voittamista, joka ei luultavasti tapahdu aivan lähivuosina. Työkalut ja teknologia kuitenkin kehittyvät jatkuvasti, ja indiepelit ovat osoittaneet monien vielä laajemmin hyödyntämättömien tekniikoiden potentiaalin.

LÄHTEET

1. AiGameDev, 2014, Challenges in AI for Games: Speaker Spotlight for Game/AI Conf. 2014, http://aigamedev.com/open/upcoming/spotlight-challenges-2014/ [viitattu 24.11.2014].
2. AiGameDev, 2014, Games of the Year: The 2013 AiGameDev.com Awards for Game AI, http://aigamedev.com/open/editorial/2013-awards/ [viitattu 24.11.2014].
3. Buckland, M., Programming Game AI by Example, 1st edition, Wordware Publishing Inc., USA, 2005.
4. Forbes, 2013, Why The PC Is Better Than The Xbox One And PS4, http://www.forbes.com/sites/antonyleather/2013/11/29/why-the-pc-is-better-than-the-xbox-one-and-ps4/ [viitattu 15.10.2014].
5. Funge, J., Millington, I., Artificial Intelligence for Games, 2nd edition, Elsevier Inc., USA, 2009.
6. GameSpot, 2010, The Future of Artificial Intelligence in Games, http://www.gamespot.com/articles/the-future-of-ai-in-games/1100-6283722/ [viitattu 24.11.2014].
7. Gartner, 2013, Gartner Says Worldwide Video Game Market to Total $93 Billion in 2013, http://www.gartner.com/newsroom/id/2614915 [viitattu 15.10.2014].
8. Kyaw, A.S., Peters, C., Swe, T.N., Unity 4.x Game AI Programming, 1st edition, Packt Publishing Ltd., UK, 2013.
9. Laird, J.E., van Lent, M., Human-level AI'S Killer Application: Interactive Computer Games, In: Proceedings of AAAI 2000, pp. 1171–1178.
10. Livingstone, D., Turing's Test and Believable AI in Games, *ACM Computers in entertainment*, Vol. 4, No. 1, January 2006.
11. Newzoo, 2014, Asia-Pacific Contributes 82% of the $6Bn Global Games Market Growth in 2014, http://www.newzoo.com/insights/asia-pacific-contributes-82-6bn-global-games-market-growth/ [viitattu 15.10.2014].