Tampereen yliopisto

Ville Niemi

VIDEOPELEISSÄ KÄYTETTÄVÄT TEKOÄLYMENETELMÄT

Informaatioteknologian ja viestinnän tiedekunta
Kandidaattitutkielma
Huhtikuu 2020

TIIVISTELMÄ

Ville Niemi: Videopeleissä käytettävät tekoälymenetelmät
Kandidaattitutkielma
Tampereen yliopisto
Tietojenkäsittelytieteiden tutkinto-ohjelma
Huhtikuu 2020

Tässä tutkielmassa tarkastellaan videopeleissä käytettäviä tekoälymenetelmiä ja selvitetään kyseisten tekoälymenetelmien toimintaa kirjallisuuskatsauksen avulla. Tutkielmassa käsiteltävät tekoälymenetelmät on pyritty valitsemaan niiden suosion perusteella niin kirjallisuudessa kuin videopeleissäkin. Valikoituneet tekoälymenetelmät ovat: *äärellinen automaatti* (finite state machine), *käytöspuu* (behavior tree), minimax-algoritmi (minimax algorithm) ja Monte Carlo -puuhaku (Monte Carlo tree search).

Äärellinen automaatti on matemaattinen malli, jossa järjestelmälle on määritelty tietyt tilat, joissa se voi olla yhdessä kerrallaan. Äärellisiä automaatteja on helppo suunnitella pienessä mittakaavassa, mutta ne ovat toiminnaltaan ennustettavia. Käytöspuu on myös matemaattinen malli, joka havainnollistaa siirtymiä äärellisen tehtäväjoukon välillä. Modulaarisuuden ansiosta käytöspuilla pystytään toteuttamaan pienien osien avulla monimutkaisia kokonaisuuksia. Minimax-algoritmia käytetään kahden pelaajan vuoropohjaisissa peleissä. Algoritmi muodostaa mahdollisista siiroista pisteytetyn puun, jonka perusteella algoritmi pyrkii valitsemaan itselleen parhaan mahdollisen siirron, ottaen huomioon, että vastustaja pyrkii valitsemaan algoritmille itselleen huonoimman mahdollisen siirron. Monte Carlo -puuhaku on algoritmi kahden pelaajan peleihin. Algoritmi käyttää hyödykseen Monte Carlo -simulaatiota, jossa peliavaruutta tutkitaan suorittamalla useita pelin loppuun pelaavia simulaatioita, jonka perusteella pyritään valitsemaan paras mahdollinen peliliike.

Jokaisen tekoälymenetelmän kohdalla kerrotaan aluksi yleisesti kyseisen tekoälymenetelmän toiminnasta, jonka jälkeen kuvataan toimintaa syvällisemmin esimerkin kautta. Lopuksi vertaillaan aiemmin esiteltyjä tekoälymenetelmiä.

Avainsanat: tekoäly, videopelit, äärellinen automaatti, käytöspuu, minimax-algoritmi, Monte Carlo -puuhaku

Sisällysluettelo

1 Johdanto 1

2 Tekoäly 2

3 Videopelien tekoälymenetelmät 3

3.1 Äärellinen automaatti 3

3.2 Käytöspuu 4

3.3 Minimax-algoritmi 6

3.4 Monte Carlo -puuhaku 8

3.5 Menetelmien vertailu 9

4 Yhteenveto 10

Lähdeluettelo 11

1 Johdanto

Videopelien suosio on kasvanut viime aikoina kovaa vauhtia ja suurin osa suomalaisista on pelannut videopelejä edes joskus. Videopelejä voi pelata monella eri laiteella, muun muassa puhelimella, tietokoneella ja erilaisilla konsoleilla, kuten Playstationilla. Videopelejä voi pelata yksin tai muita ihmispelaajia vastaan, mutta myös tekoälyn kanssa on mahdollista ottaa mittaa. Videopelitekoälyn avulla ei tarvita toista ihmispelaajaa pelaamaan peliä toisen ihmispelaajan kanssa, jolloin ei tule ongelmaa pelikumppanin löytymisestä. Videopelitekoälyn vaikeusastetta myös mahdollista säätää sopivalle tasolle suhteessa ihmispelaajaan. Näin on mahdollista saada juuri sopivan haastava vastustaja esimerkiksi shakkiin.

Tekoälyä on tutkittu jo 1950-luvulta alkaen ja tekoälyn hyödyntäminen videopeleissä on yksi tieteenalan tutkimuskohteista (Kaplan & Haenlein, 2019). Yannakakis ja Togelius (2018) ovat alan tutkijoita ja kirjoittaneet kattavan perusteoksen videopeleistä ja niissä hyödynnettävästä tekoälystä. Teos on yksi tutkielman olennaisimmista lähteistä. Myös Chaslot (2010) sekä Millington ja Funge (2009) ovat tehneet tutkimusta videopeleissä käytettävästä tekoälymenetelmistä ja heidän tutkimuksiansa hyödynnetään myös tässä tutkielmassa.

Tässä kandidaattitutkielmassa on tarkoitus perehtyä kirjallisuuskatsauksen avulla muutamiin videopeleissä käytettäviin tekoälymenetelmiin ja tuoda esiin niiden toimintaa. Tutkielmassa esiteltävät tekoälymenetelmät on pyritty valitsemaan perustuen niiden suosioon videopeleissä käytettävänä tekoälynä ja suosioon videopelitekoälytutkimuksessa. Videopelit, joita käytetään esimerkkeinä ovat yksin- tai kaksinpelejä (Ms Pac-Man, shakki ja ristinolla). Shakki ja ristinolla ovat perinteisesti lautapelejä, mutta niitä on myös mahdollista pelata tietokoneella ja ne ovat tuttuja monelle, joten ne sopivat hyvin esimerkeiksi. Esiteltyjä tekoälymenetelmiä on kuitenkin mahdollista soveltaa perinteisimpiinkin videopeleihin, kuten StarCraftiin tai League of Legendsiin.

Tutkielman toisessa luvussa käsitellään tekoälyn määritelmää yleisesti eli mitä tekoäly on. Kolmannessa luvussa esitellään neljä eri tekoälymenetelmää, joita voidaan käyttää videopeleissä hyödyksi. Tekoälymenetelmien toimintaperiaate pyritään aluksi esittelemään lyhyesti ja kertomaan minkälaisiin käyttötarkoituksiin menetelmät soveltuvat. Sen jälkeen selvennetään tekoälymenetelmien toimintaa syvällisemmin. Tekoälyme-

netelmät ovat: äärellinen automaatti, käytöspuu, minimax-algoritmi ja Monte Carlo -puuhaku. Kohdassa 3.5 vertaillaan aiemmin esiteltyjä tekoälymenetelmiä. Neljännessä luvussa tehdään yhteenveto tutkielmasta.

2 Tekoäly

Tekoälylle (artificial intelligence) on vaikea antaa yksiselitteistä määritelmää, sillä älykkyyttä on itsessään vaikea määritellä. Turingin kokeen mukaan kone on älykäs, jos koneen kanssa keskusteleva ihminen ei osaa erottaa konetta ihmisestä (TTP, 2020). Tieteen termipankki (2020) määrittelee tekoälyn mikroprosessoritekniikkaan perustuvana järjestelmänä, joka pystyy älykkäänä pidettyyn toimintaan.

Oxfordin tietojenkäsittelytieteen sanakirjassa tekoäly kuvataan tieteenalana, joka keskittyy rakentamaan tietokoneohjelmia. Tietokoneohjelmien on tarkoitus ratkaista tehtäviä, jotka vaativat ihmismäistä älykkyyttä (Butterfield et al., 2016). Tekoäly nähdään siis älylliseen toimintaan pystyvinä tietokoneohjelmina ja niitä tutkivana tieteenalana.

Yleensä tieteellisissä piireissä sellaisia älykkyyttä vaativia tehtäviä, joihin on olemassa yksinkertainen päätöksentekomenettely, ei lasketa kuuluvaksi tekoälyn piiriin, mutta taas ihmiselle ominaista havainnointikykyä vaativat tehtävät, kuten kuvan tunnistus, lasketaan tekoälyä vaativiin tehtäviin (Butterfield et al., 2016). Lasken tutkielmassani sellaiset ongelmat tekoälyn piiriin, jotka on mahdollista ratkaista päätöksentekomenettelyllä, koska näitä menetelmiä käytetään videopeleissä ja niistä puhutaan tekoälynä. McCorduck ja muut (2004) toteavat kirjassaan, että kun tietokone kykenee tekemään jotain älykkyyttä vaativaa, sitä ei pidetäkään enää tekoälynä. Ongelma nähdään ennemmin algoritmisena ja tilastollisena. On myös sanottu, että tekoäly on sitä, mitä tietokone ei pysty vielä ratkaisemaan (Heikkinen, 2017).

Tekoäly tieteenalana nähdään *älykkäiden agenttien (intelligent agent)* suunnitteluna ja tutkimuksena. Agentilla tarkoitetaan toimijaa, joka toimii ympäristössä suorittaen tehtävää. Se mitä agentti tekee, riippuu:

1. Agentin ympäröivää maailmaa koskevasta tiedosta.
2. Agentin aiemmista kokemuksistaan oppimasta tiedosta.
3. Päämääristä, johon agentti pyrkii.
4. Agentin itsestään ja ympäristöstään tekemistä havainnoista.

Näiden syötteiden perusteella agentti pyrkii suorittamaan toimintoja. Agentti voi olla esimerkiksi robotti, joka havainnoi sensoreiden avulla ympäristöä ja käyttää muistiinsa

aiemmista kokemuksista tallennettua tietoa, jotta osaisi toimia parhaalla tavalla saavuttaakseen päämäärän, kuten paketin toimittamisen kohteeseen. (Poole et al., 1998)

Agentti voi myös olla shakkia pelaava tietokoneohjelma, jolla on ennestään tiedossa shakin säännöt ja kokemusta aikaisemmista peleistä ja siirroista aiheutuvista seurauksista. Agentin tavoitteena on voittaa vastustaja asettamalla vastustajan kuningas hyökkäyksen kohteeksi ilman että vastustaja pystyy torjumaan hyökkäystä. Agentti myös havainnoi vastustajan siirtoja ja yleistä pelitilannetta päätöksenteon tueksi.

3 Videopelien tekoälymenetelmät

Tässä luvussa käsitellään erilaisia tekoälymenetelmiä, joiden avulla voidaan toteuttaa tekoälytoiminallisuutta myös videopeleihin. Tekoälymenetelmät on pyritty valitsemaan niiden suosion perusteella.

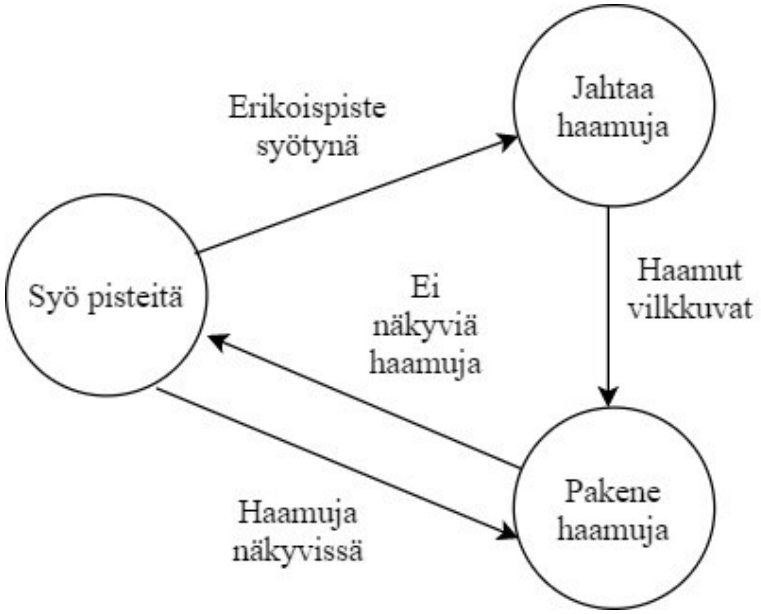
3.1 Äärellinen automaatti

Äärellinen automaatti (finite state machine) on matemaattinen malli, jossa järjestelmällä tietty on määrä eri tiloja, jossa sen on mahdollista olla. Järjestelmä voi olla vain yhdessä tilassa kerrallaan. Nykyisestä tilasta voidaan siirtyä seuraavaan tilaan vasta kun nykyisen tilan ehto on suoritettu. (Yannakakis & Togelius, 2018; Butterfield & Szymanski, 2018)

Yksinkertainen esimerkki äärellisestä automaatista on lukko. Lukon tilat ovat *suljettu* ja *auki*. Mahdollisia siirtymiä ovat *lukon avaaminen* ja *lukon sulkeminen*. Lukko voi olla kerrallaan vain yhdessä tilassa, auki tai suljettu, ja samaa siirtymää ei voi suorittaa kahta kertaa peräkkäin, eli lukkoa ei voi aukaista uudestaan ennen lukon sulkemista.

Äärelliset automaatit olivat suosituin tekoälymenetelmä videopeleissä aina 2000-luvun ensikymmenen puoliväliin asti. Äärellisiä automaatteja on todella yksinkertaista suunnitella ja toteuttaa, mikä on vaikuttanut niiden suosioon. Toisaalta suuressa mittakaavassa äärellisten automaattien suunnittelu muuttuu hankalaksi, joten ne soveltuvat vain tietynlaisiin käyttötarkoituksiin videopeleissä. Äärellisten automaattien haittapuolena pidetään myös sitä, että niiden käytös on usein ennustettavaa. (Yannakakis & Togelius, 2018)

Äärellisiä automaatteja voidaan kuvata graafisesti *tilakaavioilla* (state diagram). Tilakaavioissa ympyröillä kuvataan järjestelmän mahdollisia tiloja ja kaarilla kuvataan eri tilojen välisiä mahdollisia siirtymiä. (Butterfield & Szymanski, 2018)



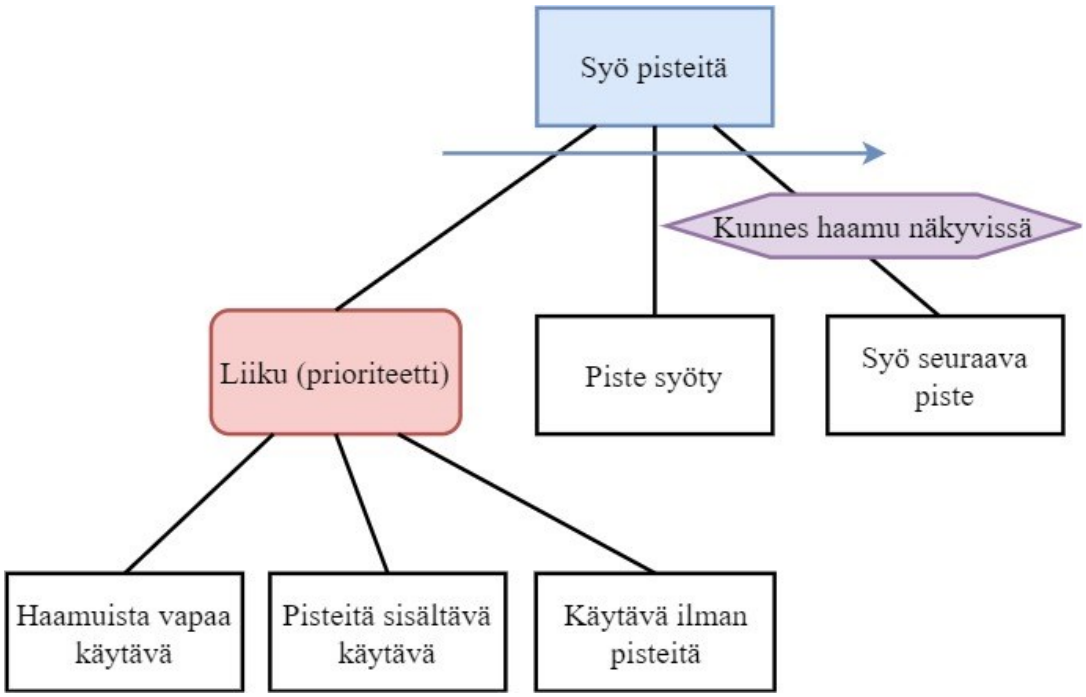
Kuva 1. Äärellinen automaatti, jolla ohjataan Ms Pac-Man hahmoa (Yannakakis & Togelius, 2018).

Yannakakis ja Togelius (2018) havainnollistavat kirjassaan äärellisen automaatin toimintaperiaatetta Ms Pac-Man peliin sijoittuvalla esimerkillä. Siinä äärelliseen automaattiin pohjautuva tekoäly ohjaa pelihahmoa, Ms Pac-Mania, jonka tehtävänä on saada kerättyä kaikki pisteet ja vältettyä haamuja. Kuvassa 1 on yksinkertaistettu tilakaavio pelihahmon ohjaimen toimintaperiaatteesta. Kuva 1 havainnollistaa äärellistä automaattia, jonka tiloja ovat *syö pisteitä, jahtaa haamuja* ja *pakene haamuja*. Siirtymiä ovat *erikoispiste syötynä, haamut vilkkuvat, ei näkyviä haamuja* ja *haamuja näkyvissä*. Syö pisteitä -tilassa Ms Pac-Man liikkuu pelikentällä satunnaisesti syöden pisteitä. Kun Ms Pac-Man syö erikoispisteen, siirtyy Ms Pac-Man jahtaa haamuja -tilaan ja rupeaa jahtaamaan pelikentällä olevia haamuja pyrkien syömään ne. Ms Pac-Man jatkaa toimintaansa kuvan 1 esittämän tilakaavion mukaisesti niin kauan kun pelissä on pisteitä jäljellä.

3.2 Käytöspuu

Käytöspuu (behavior tree) on äärellisen automaatin tapaan matemaattinen malli, joka mallintaa siirtymiä äärellisen tehtäväjoukon välillä. Käytöspuiden vahvuuksia äärellisiin automaatteihin nähden on niiden modulaarisuus: pienien osien avulla voidaan toteuttaa monimutkaisia kokonaisuuksia. Käytöspuiden suurin ero äärellisiin automaatteihin nähden on se, että käytöspuut koostuvat ennemmin toimintamalleista kuin tiloista. Käytöspuiden suosioon on myös vaikuttanut niiden helppo suunniteltavuus, testaus ja virheiden

korjaus. Käytöspuiden suosio alkoi videopeleissä vuoden 2005 jälkeen Halo 2- ja Bioshock -pelien myötä. (Yannakakis & Togelius, 2018; Colledanchise & Ögren, 2017)



Kuva 2. Käytöspuu, joka havainnollistaa Ms Pac-Man -peliin kehitettyä tekoälyä (Yannakakis & Togelius, 2018).

Käytöspuita kuvataan puurakenteella, kuten kuvassa 2. Puussa on *juurisolmu* (root node), *vanhempia* (parent) ja niiden *lapsisolmuja* (child node). Puun suoritus alkaa juurisolmusta, ja lapsisolmut voivat palauttaa vanhemmalle kolmenlaisia arvoja, k*äynnissä, onnistunut* ja *epäonnistunut*, sen mukaan mikä tilanne kyseisessä solmussa on. Käytöspuiden solmut koostuvat kolmesta eri solmutyypistä: *järjestys* (sequence)*, valitsin* (selector) ja *koristelija* (decorator). Kuvassa 2 järjestyssolmua *syö pisteitä* kuvataan sinisellä. Jos järjestyssolmun lapsisolmu palauttaa *onnistunut*-arvon siirrytään järjestyssolmun seuraavaan lapsisolmuun. Jos kaikki lapsisolmut onnistuvat, järjestyssolmun vanhempikin onnistuu, mutta muuten järjestyssolmu epäonnistuu.

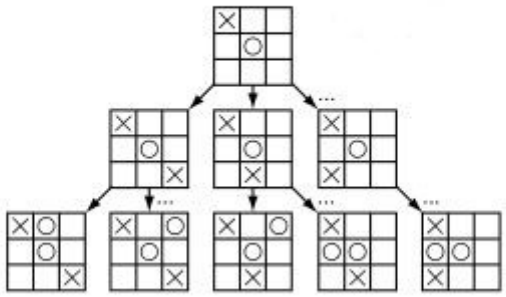
Valitsinsolmua kuvataan kuvassa 2 taas punaisella. Valitsinsolmuja on kahta päätyyppiä: *prioriteetti* (priority) ja *todennäköisyys* (probability). Kuvassa 2 on prioriteettivalitsinsolmu. Prioriteettivalitsin valitsee haluttuun järjestykseen asetetusta listasta ensimmäisen lapsisolmun. Jos lapsisolmu onnistuu, niin onnistuu myös vanhempi. Jos taas lapsisolmu epäonnistuu, siirrytään listassa seuraavaan lapsisolmuun. Jos mikään lapsisol-

muista ei palauta onnistunut-arvoa, vanhempi epäonnistuu. Todennäköisyysvalitsimen tapauksessa lapsisolmuilla on ennalta asetetut todennäköisyydet, joidenka perusteella valitaan yksi lapsisolmu. Jos lapsisolmun käytös epäonnistuu, niin vanhempi epäonnistuu, ja jos lapsi onnistuu, niin vanhempikin onnistuu.

Viimeistä solmutyyppiä kutsutaan koristelijaksi. Kuvassa 2 koristelijasolmua kuvataan violetilla. Koristelijasolmulla lisätään käytöspuuhun monimutkaisuutta ja lapsisolmun toimintaan monipuolisuutta. Koristelijasolmuilla voidaan esimerkiksi lisätä ehtoja sille, kuinka kauan lapsisolmua suoritetaan. Kuvassa 2 pelihahmo syö pisteitä niin kauan kunnes havaitaan haamuja tarpeeksi lähellä.

3.3 Minimax-algoritmi

Minimax-algoritmi (minimax algorithm) on algoritmi, jota käytetään kahden pelaajan vuoropohjaisissa peleissä, kuten shakissa ja ristinollassa. Algoritmi rakentaa puun pelin mahdollisista siirroista. Siirrot vuorotellevat omien ja vastustajan siirtojen välillä. Laskentaa jatketaan tiettyyn pisteeseen saakka. Puussa olevat solmut kuvaavat mahdollisia siirtoja. Siirrot on pisteytetty ennalta määrättyjen sääntöjen perusteella, suurempi pistemäärä tarkoittaa parempaa siirtoa. Algoritmi pyrkii valitsemaan itselleen parhaiten pisteytetyn siirron. (Butterfield et al., 2016)



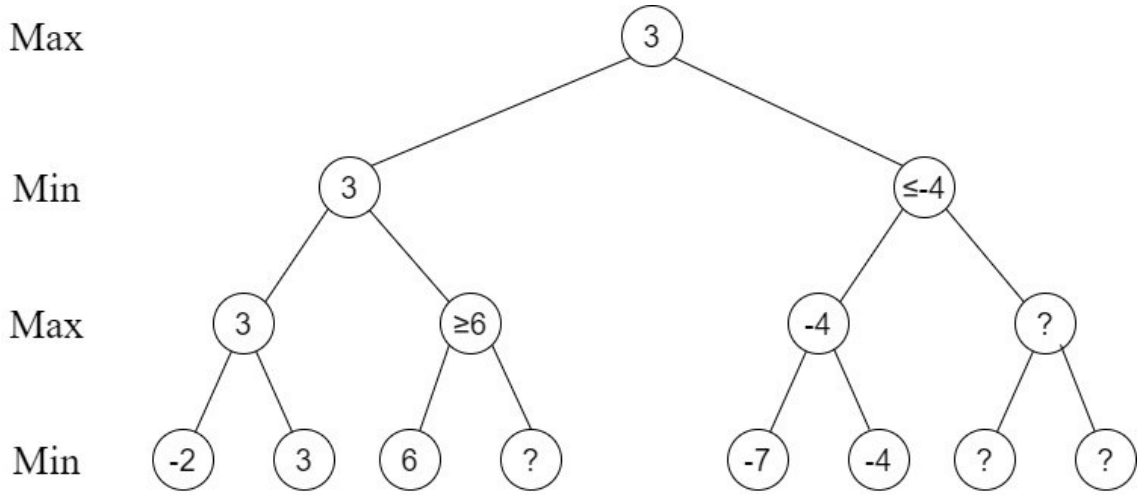
Kuva 3. Puu, joka kuvaa minimax-algoritmin toimintaa ristinolla-pelissä (Millington & Funge, 2009).

Kuva 3 kuvaa ristinolla-pelissä mahdollisia siirtoja. Kuvan 3 juurisolmu kuvaa pelin nykytilannetta ja juurisolmun lapset kuvaavat mahdollisia seuraavia siirtoja ristipuolta pelaavalle pelaajalle. Jokaisella solmulla on niin monta lapsisolmua kuin mahdollisia siirtoja on, esimerkiksi kuvan 3 juurisolmulla on todellisuudessa seitsemän lapsisolmua.

Kaikkia mahdollisia siirtoja ei esitetä kuvassa 3 niiden suurehkon määrän vuoksi. (Millington & Funge, 2009)

Jotta algoritmi voi tehdä päätöksen seuraavasta siirrosta, täytyy sen tietää, mikä mahdollisista siirroista (solmuista) on paras. Tähän algoritmi tarvitsee ennalta määriteltyä funktiota, joka arvioi ennalta annettujen sääntöjen perusteella parasta mahdollista siirtoa pelin tietyssä pelitilanteessa. Funktio pisteyttää jokaisen mahdollisen siirron tietyssä pelitilanteessa. Mitä suurempi luku on, sitä parempi siirto on pelaajalle itselleen, ja mitä pienempi luku on, sitä hyödyllisempi siirto on vastustajalle. Kun kaikki mahdolliset siirrot on pisteytetty, valitsee algoritmi itselleen hyödyllisimmän eli suurimpaan pistemäärään johtavan siirron. Minimax-algoritmi olettaa, että vastustaja pyrkii aina vastustajalle itselleen hyödyllisimpään siirtoon eli pienimmän pistemäärän omaavaan siirtoon. Algoritmin täytyy ottaa tämä huomioon omaa siirtoa valitessaan. Yleensä algoritmille annetaan myös maksimisyvyys eli kuinka pitkälle siirtoja lasketaan. Tämän tehdään siksi, että suoritus ei kestäisi kohtuuttoman pitkään, sillä joissain peleissä, kuten shakissa, voi olla lähes loputtomasti mahdollisia siirtoja. (Millington & Funge, 2009)

Minimax-algoritmin toimintaa voidaan myös tehostaa käyttämällä *alfa-beeta-karsintaa* (alpha-beta pruning). Karsinnassa jätetään laskematta osalle mahdollisista siirroista pisteet, jos voidaan todeta, että vastustaja ei tule koskaan valitsemaan kyseisiin siirtoihin johtavia siirtoja, koska vastustajan oletetaan valitsevan aina pienimmän pistemäärän omaava siirto. (Millington & Funge, 2009)

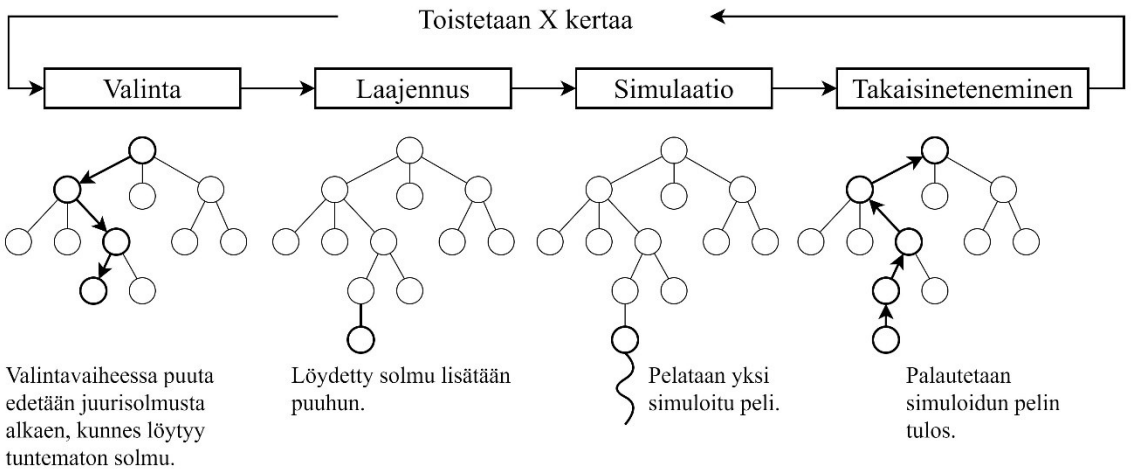


Kuva 4. Minimax-algoritmin muodostama puu, jossa laskettuna siirtojen pisteet. Osa siirroista on karsittu pois alfa-beeta-karsinnalla.

Kuva 4 havainnollistaa kuvitteellisesta pelistä muodostettua puuta, jonka solmuihin on laskettu siirtojen pisteet. Siirrot vaihtelevat minimax-algoritmin (max-taso) ja vastustajan välillä (min-taso). Max eli suurempaan pistemäärään pyrkivä pelaaja valitsee sellaisen siirtopolun, josta seuraa suurin mahdollinen pistemäärä ottaen huomioon vastustajan tekevän aina pienimpään pistemäärään johtavan siirron vuorollaan. Kuvassa 4 osa siirroista on jätetty pisteyttämättä (kysymysmerkit), koska aiemmin mainittujen sääntöjen nojalla voidaan olettaa, että niihin johtaviin siirtoihin ei koskaan päädytä. Näin vältytään turhalta siirtojen pisteytykseltä.

3.4 Monte Carlo -puuhaku

Monte Carlo -puuhaku (Monte Carlo tree search) on algoritmi yhden ja kahden pelaajan peleihin. Monte Carlo -puuhaku perustuu *Monte Carlo -simulaatioon* (Monte Carlo simulation), jonka avulla pyritään satunnaiseen peliavaruuden tutkimiseen suorittamalla useita simulaatioita, joissa jokaisessa peli pelataan loppuun yleensä satunnaisin siirroin. Simuloitujen pelien perusteella pyritään päättelemään, minkä siirron kautta olisi todennäköisintä päätyä voittoon. (Chaslot, 2010; Gelly & Silver, 2011)



Kuva 5. Monte Carlo -puuhaun eri vaiheet (Chaslot, 2010).

Monte Carlo -puuhaun toiminta voidaan jakaa neljään eri vaiheeseen: *valinta* (selection), *laajennus* (expansion), *simulaatio* (simulation) ja *takaisineteneminen* (backpropagation). Kuvassa 5 havainnollistetaan algoritmin eri vaiheita. Vaiheita toistetaan niin pitkään, kuin algoritmille on annettu aikaa. Puun solmut kuvaavat pelin eri tilanteita. (Chaslot, 2010)

Valintavaiheessa edetään puuta pitkin juurisolmusta alkaen, kunnes löytyy solmu, jonka lapsisolmu ei ole vielä osa puuta. Kulkeminen voidaan suorittaa satunnaisesti tai

jotain sääntöä käyttäen. Laajennusvaiheessa valintavaiheessa löytynyt lapsisolmu lisätään osaksi puuta. Simulaatiovaiheessa peli pelataan loppuun satunnaisin siirroin tai jotain strategiaa noudattaen. Jos strategia on liian raskas laskennallisesti, simulaatioiden määrä vähenee suhteessa käytettyyn aikaan, josta seuraa heikompia peliliikkeitä. Viimeisessä eli takaisinetenemisen vaiheessa palautetaan simulaatio vaiheessa saatu tulos (joka voi olla esimerkiksi voitto = +1, tasapeli = 0 tai häviö = -1) simuloidusta pelistä laajennusvaiheessa luotuun solmuun ja kaikkiin sitä edeltäneisiin solmuihin juurisolmuun saakka. Jokaisessa solmussa ilmaistaan solmusta tehtyjen vierailujen lukumäärä ja solmun pistemäärä. Lopuksi algoritmi pelaa yhden juurisolmun lapsisolmuissa olevan siirron. Siirron valinnassa voidaan käyttää erilaisia menetelmiä. Voidaan esimerkiksi valita solmu, jonka pistemäärä jaettuna vierailujen lukumäärällä on suurin tai, solmu, jonka kautta on suoritettu eniten simulaatioita. (Chaslot, 2010)

3.5 Menetelmien vertailu

Äärellinen automaatti ja käytöspuu muistuttavat toiminnaltaan toisiaan samoin kuin minimax-algoritmi ja Monte Carlo -puuhaku muistuttavat toisiaan. Äärellisissä automaateissa ja käytöspuissa ei pyritä ennustamaan tulevaa niin kuin minimax-algoritmi ja Monte Carlo -puuhaku tekevät. Näin ollen äärelliset automaatit eivät välttämättä sovellu kovin hyvin esimerkiksi shakin pelaamiseen, sillä peliä ei pystytä ennakoimaan useiden siirtojen päähän, mutta äärellistä automaattia voi silti käyttää shakkitekoälyssä, kun määritellään kaikki shakkilaudan tilanteen ja miten niissä kannattaa toimia. Tämä on kuitenkin paljon tilaa vievä ratkaisu ja siksi ei kovin hyvä. Toki äärellisiin automaatteihin voi ujuttaa toiminnallisuutta, jolla simuloidaan pelin kulkua useiden siirtojen päähän, mutta silloin menetelmä alkaa muistuttaa jo minimax-algoritmia ja Monte Carlo -puuhakua.

Kolme kertaa kolme ruudukolla pelattavaan ristinollaan äärellistä automaattia käyttäen toteutettu tekoäly on kuitenkin järkevä ratkaisu, koska mahdollisia tiloja ei ole kovin montaa ja ristinollaan on olemassa optimaalinen voittostrategia. Äärellinen automaatti on myös helpompaa toteuttaa kolme kertaa kolme ruudukon ristinollaan kuin minimax-algoritmiin tai Monte Carlo -puuhakuun perustuva ratkaisu. Isommalla ruudukolla taas tilanne on toinen, sillä mahdollisten siirtojen määrä kasvaa suureksi, jolloin minimax-algoritmin tai Monte Carlo -puuhaun kaltainen ennustamiseen pyrkivä ratkaisu olisi parempi.

Kun vertaillaan Minimax-algoritmia ja Monte Carlo -puuhakua, yksi merkittävä ero on se, että minimax-algoritmissa on ideana laskea kaikki mahdolliset peliliikkeet usean

vuoron päähän, ellei käytetä alfa-beeta-karsintaa. Tämä on paljon laskentatehoa vaativa ratkaisu, jos pelataan peliä kuten Go, jossa paljon mahdollisia siirtoja. Siksi Monte Carlo -puuhaun kaltainen ratkaisu on tehokkaampi, sillä kaikkia mahdollisia siirtoja ei pyritä selvittämään vaan pelistä tehdään haluttu määrä simulaatioita, joissa peli pelataan loppuun eri siirroin, ja pyritään valitsemaan todennäköisin siirto, joka johtaisi voittoon.

Aina kuitenkaan ei ole hyvä asia, jos tekoäly on liian hyvä ihmispelaajaa vastaan. Jos halutaan, että ihmispelaajakin kykenee voittamaan, voidaan tekoälymenetelmän tehoa heikentää. Äärellisten automaattien ja käytöspuiden tapauksessa se onnistuu heikentämällä strategian tehokkuutta – ei tehdäkään parasta mahdollista siirtoa. Minimax-algoritmin tapauksessa voidaan valita heikomman pistemäärän omaava siirto, kun halutaan antaa vastustajalle parempi mahdollisuus voittaa. Monte Carlo -puuhaussa voidaan vähentää simuloitujen pelien määrää, jolloin päädytään todennäköisemmin heikompaan siirtoon.

4 Yhteenveto

Tekoälyä on vaikea määritellä, koska älykkyyden määritelmä on itsessään hieman epäselvä. Yleensä älykkyydellä tarkoitetaan ihmismäistä älykkyyttä. Tekoälyn yksi tutkimuskohteista on videopelitekoäly, jonka avulla pyritään simuloimaan älyllistä toimintaa videopelien eri tilanteissa. Tutkielmassa käsiteltiin neljää eri videopeleissä käytettävää tekoälymenetelmää, jotka olivat: äärellinen automaatti, käytöspuu, minimax-algoritmi ja Monte Carlo -puuhaku.

Äärellinen automaatti on matemaattinen malli, jossa järjestelmällä on ennalta määrätyt tilat, joissa se voi olla vain yhdessä kerrallaan. Äärellisien automaattien etu on se, että niitä on yksinkertaista suunnitella. Haittapuolina ovat niiden toiminnan ennustettavuus ja suunnittelun vaikeus suuressa mittakaavassa. Äärelliset automaatit olivat suositun tekoälymenetelmä videopeleissä aina vuoteen 2005 saakka. Äärellisiä automaatteja kuvataan tilakaavioilla.

Käytöspuu on myös matemaattinen malli, joka mallintaa siirtymiä äärellisen tehtäväjoukon välillä. Modulaarisuuden ansiosta käytöspuilla pystytään toteuttamaan pienien osien avulla monimutkaisia kokonaisuuksia. Käytöspuun valtteja on helppo suunniteltavuus, testaus ja virheiden korjaus. Käytöspuiden suosio alkoi vuoden 2005 tienoilla Halo 2- ja Bioshock -pelien myötä.

Minimax-algoritmia käytetään kahden pelaajan vuoropohjaisissa peleissä. Algoritmi muodostaa puun pelin mahdollisista siirroista pisteyttäen ne samalla. Siirrot vuorottelevat omien ja vastustajan siirtojen välillä ja algoritmi valitsee vuorollaan suurimman pistemäärän omaavan siirron ottaen huomioon, että vastustaja pyrkii pienimmän pistemäärän omaavaan siirtoon. Minimax-algoritmin toimintaa voidaan tehostaa alfa-beeta-karsinnalla, jolloin osa siirroista jätetään pisteyttämättä, koska niihin ei tulla oletettavasti päätymään, kun vastustaja tavoittelee mahdollisimman pientä pistemäärää ja algoritmi itse mahdollisimman suurta.

Monte Carlo -puuhaku on algoritmi, joka soveltuu yhden ja kahden pelaajan peleihin. Algoritmi perustuu Monte Carlo -simulaatioon, jossa tutkitaan peliavaruutta suorittamalla useita simulaatioita, joissa peli pelataan loppuun. Simulaatioiden tulosten perusteella Monte Carlo -puuhaku pyrkii valitsemaan parhaimman peliliikkeen.

Tulevaisuudessa videopeleissä käytettävä tekoäly siirtyy luultavasti kohti koneoppimista ja neuroverkkoja. Näitä menetelmiä on jo kokeiltu joissain peleissä kuten StarCraftissa ja shakissa. Tutkielmassa esitellyt tekoälymenetelmät todennäköisesti säilyvät etenkin harrastelijoiden ja pienten ohjelmistoyritysten käytössä vielä jatkossakin, sillä esitellyt tekoälymenetelmät helpompia toteuttaa ja ne eivät vaadi niin suurta laskentatehoa, mitä koneoppimiseen ja neuroverkkoihin tarvitaan.

Lähdeluettelo

Butterfield, A., Ngondi, G. & Kerr, A. (2016). *A dictionary of computer science (Seventh edition.).* Oxford, England: Oxford University Press.

Butterfield, A. & Szymanski, J. (2018). *A Dictionary of Electronics and Electrical Engineering (5 ed.).* Oxford, England: Oxford University Press.

Chaslot, G. M. J-B. C. (2010). *Monte-Carlo Tree Search*. Maastricht: Maastricht University.

Colledanchise, M. & Ögren, P. (2017). *Behavior Trees in Robotics and AI: An Introduction.* https://arxiv.org/abs/1709.00084

Gelly, S. & Silver, D. (2011). Monte-Carlo tree search and rapid action value estimation in computer Go. Artificial Intelligence, 175(11), 1856–1875. https://doi.org/10.1016/j.artint.2011.03.007

Heikkinen, S. (2017). *Tekoäly muuttaa maailman – pian se tekee jopa lääkärin ja juristin töitä.* Yle Uutiset. https://yle.fi/aihe/artikkeli/2017/06/04/tekoaly-muuttaa-maailman-pian-se-tekee-jopa-laakarin-ja-juristin-toita (Haettu 2.4.2020)

Kaplan, A., & Haenlein, M. (2019). Siri, Siri, in my hand: Who's the fairest in the land? On the interpretations, illustrations, and implications of artificial intelligence. *Business Horizons*, 62(1), 15–25. https://doi.org/10.1016/j.bushor.2018.08.004

McCorduck, P. (2004). *Machines who think: a personal inquiry into the history and prospects of artificial intelligence (2. ed.).* Natick (Mass.): A K Peters.

Millington, I. & Funge, J. (2009). *Artificial intelligence for games* (2nd ed.). Boca Raton, Florida: CRC Press.

Poole, D., Mackworth, A. & Goebel, R. (1998). *Computational intelligence: a logical approach*. New York: Oxford University Press.

TTP. (2020). *Tieteen termipankki.* https://tieteentermipankki.fi/w/index.php?title=Filosofia:teko%C3%A4ly&oldid=456538 (Haettu 23.2.2020)

Yannakakis, G. & Togelius, J. (2018). *Artificial intelligence and games.* Springer.