TIETO- JA SÄHKÖTEKNIIKAN TIEDEKUNTA

Janne Eskola
Eetu Haapamäki
Jani Jarkima

Shakkirobotin pelitekoäly rajallisen laskentatehon ympäristössä

Kandidaatintyö
Tietotekniikan tutkinto-ohjelma
Huhtikuu 2019

Eskola J. Haapamäki E. Jarkima J. (2019) Shakkirobotin pelitekoäly rajallisen laskentatehon ympäristössä. Oulun yliopisto, tietotekniikan tutkinto-ohjelma. Kandidaatintyö, 37 s.

TIIVISTELMÄ

Shakki pelinä on kiinnostanut tiedemiehiä jo vuosisatoja. Peliä on käytetty historiassa erilaisten matemaattisten ongelmien ja varhaisten algoritmien havainnollistamiseen. Myös nykyaikana shakista on tullut laaja mielenkiinnon aihe varsinkin erilaisten tekoälyjen kehittämisessä. Shakki on pelinä liian monimutkainen ratkaistavaksi raa'alla laskentateholla, mikä tekee siitä erinomaisen kehitysalustan tekoälyjen tehokkuuden vertailemiseen.

Tässä työssä toteutettiin tekoäly shakkia pelaavalle robotille. Robotin työlle merkittävät osat koostuivat mekaanisesta käsivarresta ja kamerasta, joka tunnisti pelilaudan tilanteen konenäön avulla. Laiteympäristönä käytettiin erittäin rajallisen laskentatehon Raspberry Pi 3 -minitietokonetta.

Laskentateho vaikuttaa suoraan sekä algoritmin pelikykyyn että ihmisvastustajan kokemaan odotusaikaan siirtojen välillä, joten tekoälyn suorituskyky oli tärkeässä asemassa työn kannalta. Suorituskykyä testattiin vaiheittaisesti, aloittaen naiivista minimax-algoritmista siirtyen alfa-beta -karsintaan ja sen erilaisiin optimisaatiotekniikoihin. Näitä eri algoritmillisia toteutuksia ja niiden suoritusaikoja vertailtiin keskenään, mikä auttoi määrittämään minimax-pohjaisten algoritmien soveltuvuutta vakuuttavan shakkitekoälyn luomiseen vähäisellä laskentateholla. Vertailuissa havaittiin selvästi perinteisen minimaxin hitaus suhteessa alfa-beta -karsintaan, varsinkin suuremmilla hakusyvyyksillä. Nopein suorituskyky ilman pelaamisen tason vähentymistä saavutettiin alfa-beta -karsintaa optimoimalla shakin avauskirjoja hyödyntämällä.

Avainsanat: Shakki, robotiikka, algoritmi, minimax, alfa-beta

Eskola J. Haapamäki E. Jarkima J. (2019) Artificial intelligence of a chess robot in a system with limited computational power. University of Oulu, Degree Programme in Computer Science and Engineering. Bachelor's Thesis, 37 p.

ABSTRACT

Chess as a game has interested scientists for centuries. In history, the game has been used to demonstrate different kinds of mathematical problems and early algorithms. Even today, chess has become a subject of great interest, especially in the development of various artificial intelligence. Chess is too complex to solve by raw computing power, which makes it excellent platform to compare the strength of artificial intelligence.

In this work, an artificial intelligence was made for a chess robot. The parts of the robot that were relevant to the work consisted of a mechanical arm and a camera that recognized the situation of the game board with the help of machine vision. Raspberry Pi 3 was used as a hardware of the artificial intelligence.

Computational power directly effects the strength of the algorithm and the time experienced by the human opponent which makes the performance of the artificial intelligence very important factor for the project. Performance was tested step by step, starting with a stripped minimax algorithm and moving to alpha-beta pruning and its various optimization techniques. These different algorithmic implementations and their execution times were compared to help determining the suitability of minimax-based algorithms to create convincing chess artificial intelligence with limited computing power. Comparisons clearly showed the slowness of the traditional minimax relative to alpha-beta pruning, especially at higher search depths. The fastest performance without any reduction in the level of gaming was achieved by optimizing alpha-beta pruning by utilizing the chess opening books.

Keywords: Robotics, algorithm, minimax, alpha-beta

SISÄLLYSLUETTELO

ALKULAUSE

Haluamme kiittää Teemu Tokolaa tämän työn ohjaamisesta sekä neuvojen antamisesta. Lisäksi haluamme kiittää kaikkia tekoälyn testaukseen osallistuneita henkilöitä.

Oulu, Toukokuu 14. 2019

Janne Eskola
Eetu Haapamäki
Jani Jarkima

LYHENTEIDEN JA MERKKIEN SELITYKSET

IRC	Internet Relay Chat.
IFR	International Federation of Robotics
HRI	Human Robot Interaction
GPS	Global Positioning System
AI	Artificial Intelligence
UCI	Universal Chess Interface
FEN	Forsyth-Edwards Notation

1. JOHDANTO

Idea automatisoiduista tai ohjailtavista mekaanisista laitteista on ollut olemassa jo esihistoriallisten kansojen myyteissä [1], mutta vasta viime vuosikymmeninä teknologia on kehittynyt tarpeeksi mahdollistamaan edistyneet, modernit robotit. Robotit ovat koneita, jotka kykenevät suorittamaan mekaanisia työtehtäviä joko autonomisesti tai ihmisten ohjattavana. Sana ”robotti” tulee tšekinkielisestä sanasta ”robota”, tarkoittaen pakkotyöläistä tai orjaa. Modernissa käsityksessä robotit toimivat sähkömekaanisen koneen ja jonkintyyppisen tietoteknisen ohjelmiston sulautettuna järjestelmänä. Robotiikka konseptina onkin melko laaja ja käsittää useita tekniikan ja tieteen aloja, mukaan lukien esimerkiksi tieto-, sähkö- ja konetekniikan sekä tietojenkäsittelytieteen alat monien muiden joukossa. Tästä johtuen robotiikan kehittyminen on ollut suurelta osin sidottuna myös näiden alojen kehittymiseen.

Vaikka robottien edistyksessä on tapahtunut suuria harppauksia, ovat täysin autonomiset yleisrobotit yhä saavuttamattomissa. Stereotyyppistä robottia ei olekaan olemassa, vaan robotit suunnitellaan ja rakennetaan käyttötarkoituksensa huomioon ottaen [2]. Teollisuudessa toimivien robottien yleistyessä ja robotiikan kehittyessä tämä teknologia on tullut myös kuluttajien ulottuviin. Robotiikan leviämisen myötä myös yhä arkipäiväisempiä työtehtäviä suorittavat robotit ovat lisääntyneet, ja ei olekaan tavatonta nähdä tätä teknologiaa tavanomaistenkin kuluttajien käytössä esimerkiksi siivous- tai ruohonleikkuurobottien muodossa [3].

Teknologian kehittyessä robotteja on kehitetty paitsi töiden automatisointiin, myös viihdekäyttöön. Eräs robotiikan osa-alue on pelirobotit, joiden kanssa ihminen voi pelata erilaisia lauta- ja videopelejä. Pelaamista varten robotti tarvitsee tavan liikuttaa pelinappuloita: joko mekaanisen käden, jota ohjata tai avustajan jolle robotti antaa ohjeita. Robotti tarvitsee myös ohjelman, joka käy läpi ja valitsee robotin liikkeet. Nykyaikana tämä yleensä toteutetaan tekoälyn, ajattelua simuloivan ohjelman, avulla.

Työn tavoite oli toteuttaa tekoäly shakkia pelaavalle robotille. Ongelman tutkiminen aloitettiin luomalla tekoälylle useita eri algoritmillisia toteutuksia, joita vertailtiin keskenään sekä pelillisesti että laskentatehollisesti.

2. ROBOTIIKKA

Robotiikka alana käsittelee robottien suunnittelemista, rakentamista ja käyttöä. Vaikka robottien määrä yhteiskunnassa nousee jatkuvasti, yleistymistä pidättelevät yhä puutteet teknologiassa. Modernitkaan robotit eivät kykene toimimaan tuntemattomassa ympäristössä tai sopeutumaan yllättäviin tilanteisiin kovin hyvin [4]. Poikkitieteellisenä tekniikan alana robotiikan leviämisen myötä myös siihen liittyvien tieteiden määrä on kasvanut. Kun robottien historiassa robotiikka keskittyi lähinnä alun perin mekaanisiin robotteihin, siitä teknologian kehittyessä laajentaen näkökulmaansa sähkö- ja tietoteknisiin puoliin, nykypäivän ongelmiin vastaten robotiikka on yhä laajemmin ottanut voimavarakseen esimerkiksi kognitiivisen neurotieteen ja biomimetiikan. Robottien liikkeeseen inspiraatiota on haettu esimerkiksi kenguruiden pomppimisesta [5], tekoälyllisiä ongelmia taas on ratkottu hakemalla algoritmeihin nostoa biologisista neuroverkoista ja aivojen toiminnasta [6].

Ratkaisemattomista ongelmista huolimatta robotit jatkavat tasaista marssiaan ihmisten elämään, sekä työpaikalla että sen ulkopuolella. IFR (International Federation of Robotics) arvioi vuonna 2017 maailmassa myytyjen teollisuusrobottien määräksi 381,000 kappaletta, tai 30 prosenttia suurempi määrä kuin vuonna 2016. Maailman teollisuusrobottien määrä kasvoi samalla aikavälillä 15 prosenttia 2,098,000:een kappaleeseen [7]. IFR pitää tämän trendin jatkumisen myös tulevaisuudessa todennäköisenä, arvioiden että vuosien 2018 ja 2021 välillä maailmassa tullaan myymään yli 2 miljoonaa uutta teollisuusrobottia, tuplaten teollisuusrobottien kokonaismäärän. Robottien yleisesti odotetaankin tulevan alati yhä integraalisemmaksi osaksi päivittäistä elämäämme.

Maallikkojen ei voida oletettavan kykenemään hallitsemaan monimutkaisia robottien ohjauslaitteita, joten hyödyn hakemiseen robotiikasta muuallakin kuin tehdaslattialla myös robottien sosiaalisen älyn kehittäminen on suuremmassa roolissa edistyksen saralla. Ihmisten ja robottien välistä vuorovaikutusta tutkii HRI (Human Robot Interaction). HRI:n tarkoitusperä on ohjata robottien ohjausjärjestelmien kehitystä ihmisille luonnollisemmaksi.

Sheridan jakaa HRI:n neljään osa-alueeseen sen mukaan minkälainen ihmisen ja robotin välinen suhde on kyseisessä tilanteessa. Ensimmäisessä osa-alueessa robotti tekee rutiininomaisia toimintoja ihmisen valvonnan alaisena. Toisessa osa-alueessa ihminen antaa etäältä komentoja robotille, joka on ihmiselle vaarallisessa ympäristössä. Kolmannessa osa-alueessa on kyse kulkuneuvoista, joissa ihminen on matkustaja ja kulkuneuvon ohjaaminen on automatisoitu. Viimeinen osa-alue on robotin ja ihmisen välinen sosiaalinen kanssakäyminen. [8]

2.1. Teollisuus

Robotteja käytetään teollisuudessa likaisiin, vaarallisiin ja tylsiin työtehtäviin. Teollisuudessa robotiikka tuottaa ja on tuottanut paljon kehitystä. Yleensä teollisuusrobotit ovat isoja, paikallaan pysyviä liukuhihnan varressa seisovia robottikäsiä, joiden luokse osat tuodaan käsiteltäviksi, kuten kuvassa 1 olevat teollisuusrobotit. Koska robotin työ on usein saman asian toistamista samassa pisteessä ja samalla tavalla, teollisuusrobotit vaativat työtä varten vain alkeellisia näkö- tai tuntosensoreita tai eivät vaadi sensoreita lainkaan. Toisaalta robotit tekevät usein työtä

ihmisten kanssa samoissa tiloissa, jolloin roboteissa on oltava sensoreita turvallisuussyistä [9, 10].

Robotit ovat olleet tärkeässä osassa niin sanottua kolmatta teollista vallankumousta, jossa tuotantoa automatisoidaan elektroniikalla ja informaatioteknologialla. Neljännessä teollisessa vallankumouksessa on kyse niin sanotuista kyberfyysisistä systeemeistä [11]. Käytännössä tämä tarkoittaa fyysisen maailman yhdistämistä virtuaalimaailmaan. Neljännessä teollisessa vallankumouksessa robotit kykenevät työskentelemään täysin autonomisesti ihmisten kanssa samoissa tiloissa ihmisten kanssa eli ihmisten ja robottien väliseen vuorovaikutukseen on panostettava tulevaisuudessa enemmän [12].

Kuva 1. Teollisuusrobotteja[1]

2.2. Robottiautot

Yleisillä teillä ja kaupunkiajossa käytössä olevat robottiautot ovat suuri haaste suunnittelijoille ja insinööreille. Itseään ajavan auton on osattava noudattaa liikennesääntöjä, lukea liikennemerkkejä ja –valoja, sekä tiemerkintöjä. Monimutkaiset ajoliikkeet kuten kaistojen vaihtaminen, peruuttaminen ja taskuparkkeeraus, mitkä sisältävät tarkkoja liikkeitä ahtaissa tiloissa vaativat laadukasta suunnittelua ja ohjelmointia. Lisäksi robottiautot ovat läheisessä toiminnassa ihmisten kanssa ja robotin on osattava varoa jalankulkijoita, pyöräilijöitä sekä muita autoilijoita jotka voivat joissain tilanteissa käyttäytyä arvaamattomasti. Robottiauton on sisällettävä useita eri tekniikoita ja sensoreita ollakseen turvallinen

[1] Bernard Agullo (2005), CC-BY: https://creativecommons.org/licenses/by-sa/2.0/deed.en

yleiseen käyttöön, kuten infrapunasensoreita esineiden havaitsemiseen, kameroita liikennemerkkien ja kappaleiden tunnistukseen ja GPS:ää paikannukseen ja nopeusrajoitusten tunnistamiseen. Ihmisten ajotyylin matkimista on hyödynnetty robottiautojen ohjelmoinnissa. Esimerkiksi Chang ym. käyttivät fuzzy behaviour-tekniikkaa robotin ohjelmoimiseen. Näin robotti opetettiin seuraamaan seiniä nurkkien kautta sekä parkkeeraamaan taskuun [13].

Autoroboteista on tehty useita erilaisia tutkimuksia keskittyen eri osa-alueisiin ja teknologioihin [13, 14, 15]. Perinteisten autojen mallisten robottien lisäksi tutkijat ovat kehittäneet nykyisiä ihmisen mallisia robotteja ajamaan autoja, esimerkiksi Paolillo ym. 2016 kehittivät robotin joka pystyy ajamaan ja ohjaamaan autoa sekä itsekseen että ihmisen kauko-ohjaamana [15]. Raportissaan he mainitsivat tällaisen robotin olevan avuksi esimerkiksi kriisitilanteissa joissa robotti joutuu auton ajamisen lisäksi suorittamaan muita avustus- tai pelastustehtäviä. Ullah ym. liittivät robottiautoon tavan kauko-ohjata autoa käden liikkeillä ja puhelimella sovelluksen avulla sekä äänentunnistuksella [14].

2.3. Lääketiede

Lääketieteessä robotiikkaa kyetään hyödyntämään sen lisäämän tarkkuuden ja virheettömyyden vuoksi. Käytännössä vaikka ihmislääkäri olisi kuinka pätevä niin tarpeeksi kehittynyt laite kykenee tekemään samat tehtävät joko yhtä hyvin tai paremmin. Zhou ym. esittelevät artikkelissaan uusimpia kohteita robotiikalle lääketieteessä. Suurin osa näistä on erilaisiin leikkauksiin kuten neurokirurgiaan suunnitellut robotit mutta mukana on myös robottiproteeseja sekä muita apuvälineitä, joilla henkilön puuttuvaa toimintakykyä pystytään korvaamaan [16].

Kirurgiassa käytettäviltä roboteilta vaaditaan äärimmäistä nanometrien tarkkuutta. Lisäksi robotit, jotka toimivat kirurgin ohjaamina tarvitsevat sensoreita potilaan turvallisuuden takaamiseksi, esimerkiksi kameroita tai painesensoreita, jotta robotin ohjaaja tietää jatkuvasti mitä robotti tekee. Lääketieteessä käytettyjen robottien on myös oltava päästöttömiä sekä helposti puhdistettavia steriiliyden takaamiseksi [16].

Lääketieteessä robotiikan automaation voidaan katsoa jakautuvan 6 eri tasolle sen mukaan kuinka itsenäisesti robotti kykenee toimimaan. Alhaisimmalla tasolla eli tasolla 0 robotit ovat täysin niiden ohjaajan hallinnassa eikä niissä ole mitään tekoälyä. Korkeimmalla tasolla eli tasolla 5 robotti on täysin itsenäinen eli kykenee hoitamaan lääketieteellisiä toimenpiteitä täysin ilman ihmisen vaikutusta. Joillain tasoilla robotti kykenee toimimaan itsenäisesti, mutta sen täytyy toimia lääkärin valvonnan alla [17].

2.4. Sotilasrobotit

Sotilaskäytössä yleisimpiä robotteja ovat miehittämättömät maa-alukset ja miehittämättömät ilma-alukset. Miehittämättömiä maa-aluksia käytetään pääsääntöisesti apuna tiloissa tai paikoissa, joihin ihminen ei pysty turvallisesti menemään. Miehittämättömiä ilma-aluksia käytettiin aikaisemmin vain tiedustelukäytössä, mutta viime aikoina niitä on alettu aseistamaan [18]. Euroopan parlamentti on ottanut miehittämättömien aseiden kehitykseen kantaa ja pyrkii kieltämään täysin autonomiset ja/tai sellaiset aseet, jotka kykenevät tekemään kriittiset päätökset kuten kohteen valinta ja tuhoaminen ilman ihmisen vuorovaikutusta [19].

2.5. Palvelu- ja viihderobotit

Palvelurobotteja on sekä tavallisia että sosiaalisesti interaktiivisia. Robotti-imurin ei välttämättä tarvitse pystyä keskustelemaan ihmisen kanssa, mutta opastavalle robotille tämä on oleellista. Sosiaalisessa kanssakäymisessä oleville roboteille pyritään yleensä suunnittelemaan ihmismäisiä ulkonäön piirteitä ja liikkumisesta pyritään tekemään mahdollisimman ihmismäistä. Kuvassa 2 on Pepper-palvelurobotti josta on selvästi erotettavissa ihmismäisiä piirteitä. Tällä pyritään saada aikaan se, että käyttäjät kohtelevat robottia kuin ihmistä, eikä koneena [20]. Esimerkki sosiaalisesti interaktiivisesta palvelurobotista on Sacarino. Sacarinon tehtäviin kuuluu hotellin vieraiden kanssa keskusteleminen, neuvominen ja erilaiset palvelut, kuten vaikka taksin tilaaminen vieraan käskystä [21].

Verrattuna sosiaalisiin palvelurobotteihin, viihderobottien persoonallisuus voi olla enemmän lemmikkimäinen tai piirrettymäisempi [22]. Aikaisemmin mainitsemallemme Sacarino-robotille on annettu myös viihderobotin ominaisuuksia. Sille oli annettu yli 5000 vitsiä ja sananlaskua, joilla se pystyy viihdyttämään hotellivieraita [21].

Kuva 2. Pepper-robotti[2]

[2] Softbank Robotics Europe (2017), CC-BY: https://creativecommons.org/licenses/by-sa/4.0/deed.en

2.6. Shakkirobotit

Viihdekäyttöön suunniteltuja robotteja on ollut vuosien saatossa monenlaisia. Eräs robotiikan osa-alue on robotit, jotka pystyvät pelaamaan lautapelejä, esimerkiksi shakkia [23, 24, 25, 26]. Shakin pelaaminen on monimutkainen prosessi, joka vaatii monta erilaista mekaanista ja digitaalista osaa sekä näiden yhteistyötä. Shakkia pelatakseen robotissa on oltava koura nappuloiden liikuttamiseksi, tapa nähdä shakkilauta ja sillä olevat nappulat sekä tekoäly, joka koordinoi robotin pelaamista.

Valmiita shakkia pelaavia robotteja on jo useita [23, 25, 26], mutta käytännön toteutus näiden robottien välillä vaihtelee. Esimerkiksi nappuloiden liikuttamisessa käytetyt kourat vaihtelevat monin tavoin kierrätysosista kootuista käsistä [25] teollisiin kouriin [26]. Kourat vaihtelevat esimerkiksi koon, nopeuden, liikeratojen sekä nivelten määrän mukaan.

Yksi olennainen merkittävä asia shakkirobotin toiminnan kannalta on shakkilaudan ja -nappuloiden tunnistaminen. Tästä aiheesta on jo useita kattavia julkaisuja [23, 24]. Esimerkiksi Larregay ym. (2018) käyttivät shakkilaudan, nappuloiden ja siirtojen tunnistamiseen videokuvaa ja digitaalista kuvankäsittelyä [23], kun taas Jiao ym. korjasivat kameran linssin vääristämän kuvan pelilaudasta nurkantunnistusalgoritmin ja referenssilaudan avulla [24].

Luonnollisesti shakkirobotin toiminnassa avainasemassa on myös tekoäly, joka toimii robotin aivoina. Tekoälyn avulla robotti kykenee ymmärtämään shakin säännöt, tekemään päätöksiä liikkeistään sekä tunnistamaan vastapelaajan siirrot. Näiden prosessien on myös pystyttävä kommunikoimaan keskenään, jotta robotti pystyy toimimaan.

3. TEKOÄLY

Tekoälyllä tarkoitetaan laitteita ja ohjelmistoja, jotka imitoivat älyllistä ajattelua. Teknisempi selitys tekoälylle on järjestelmä, joka pystyy tulkitsemaan ulkoista dataa ja käyttämään tästä kerättyä tietoa tiettyjen ongelmien ratkaisussa [27]. Nykyään käytössä olevien tekoälyjen sanotaan olevan heikkoja tekoälyjä, koska ne on suunniteltu jotain tiettyä tehtävää varten. Vahvan tekoälyn täytyisi pystyä ratkaisemaan käytännössä kaikki sille annetut tehtävät yhtä tehokkaasti kuin nykyinen tekoäly kykenee ratkaisemaan ongelman, jonka ratkaisemiseen se on suunniteltu [28]. Ihmisen etu tekoälyyn nähden on sopeutumiskyky. Ihminen kykenee reagoimaan ympäristön muutoksiin paremmin kuin kone [29].

Andreas Kaplan ja Michael Haenlein jakavat tekoälyn kolmeen eri osa-alueeseen: analyyttiseen tekoälyyn (Analytical AI), ihmispohjainen tekoäly (Human-Inspired AI) sekä inhimillistetty tekoäly (Humanized AI). Analyyttisessä tekoälyssä on kyse pelkästään kognitiivisesta älystä, eli tämänlaiset järjestelmät vain tekevät päätöksiä sille saatavissa olevasta tiedosta. Ihmispohjaisessa tekoälyssä laitteella on myös kyky ymmärtää tunneälyä. Inhimillistettyä tekoälyä ei olla vielä onnistuttu luomaan, mutta käytännössä kyseessä on vahva tekoäly, joka pystyisi toimimaan pätevästi kaikissa mahdollisissa tilanteissa. [27]

3.1. Koneoppiminen

Koneoppimisella tarkoitetaan sitä, kun ohjelmisto käyttää itse sille annettua tietoa oman toimintansa optimoinnissa [30]. Tämä tieto voi olla esimerkkidataa tai ohjelmiston omista aikaisemmista toiminnoista kerättyä dataa. Esimerkkidata voi olla vaikkapa suuri määrä kuvia, joilla pyritään optimoimaan kuvantunnistusohjelma. Koneoppiminen on tarpeellista varsinkin tilanteissa, joissa ihmisellä ei ole tarpeellista tietoa ratkaistavasta ongelmasta tai kyseinen tieto on ihmiselle liian vaikeasti ymmärrettävissä [30]. Koneoppimisella pyritään vähentämään tarvetta ohjelmoida kaikkea alusta alkaen itse, vaan ohjelman annetaan itsenäisesti luoda tarpeelliset säännöt ongelman ratkaisua varten [31].

3.1.1. Oppimisalgoritmit

Koneoppimisen kolme yleisintä oppimisalgoritmia ovat ohjattu oppiminen, ohjaamaton oppiminen ja vahvistusoppiminen. Ohjatussa oppimisessa esimerkkidata koostuu syötteistä ja tuloksista, joiden pohjalta pyritään luomaan malli joka pystyy antamaan esimerkkidataan kuulumattomalle syötteelle sopivan tuloksen. Ohjatulla oppimisella ratkaistavia ongelmia ovat regressio- ja lajitteluongelmat. Regressio-ongelmassa testiaineiston pohjalta luodaan regressiosuora, esimerkiksi $y=a+bx$, jossa y on tulos ja x syöte. Muuttujat a ja b määritellään niin että yhtälöä voidaan käyttää ennustamaan y:n arvoja eri x arvoilla. Luokitteluongelmassa pyritään ennustamaan syötteelle sopiva luokka esimerkkidatan mukaan.

Ohjaamattomassa oppimisessa mallia ei rakenneta syötteiden ja tulosten pohjalta. Esimerkkidataa ei ole luokiteltu eli se koostuu käytännössä vain tuloksista, jotka jaotellaan luokkiin ja pyritään luomaan näitä luokkia vastaava malli. Käytännön sovelluksia voi olla esimerkiksi suuren sekalaisen datamäärän lokerointi samankaltaisten kanssa.

Vahvistusoppimisessa oppiminen tapahtuu ympäristön antaman palautteen mukaan. Palaute voi olla positiivista tai negatiivista. Oppiminen tapahtuu yrityksen ja erehdyksen kautta, kun agentti pyrkii saamaan mahdollisimman paljon positiivista palautetta. Yksi vahvistusoppimisen käytännön sovelluksia ovat pelitekoälyt. Esimerkiksi AlphaGo Zero tekoäly opetettiin pelaaman Go-peliä vahvistusoppimisella [32]. [30, 33]

3.1.2. Neuroverkot ja syväoppiminen

Neuroverkkojen tarkoituksena on luoda algoritmi, jonka toiminta on lähellä ihmisen aivojen toimintaa. Neuroverkko koostuu neuroneista, jotka saavat tietyn määrän syötteitä tietyillä painoarvoilla ja laskevat aktivaatiofunktion mukaisen ulostulon [33]. Yksinkertaisimmillaan neuroverkko voi koostua vain yhdestä neuronista, mutta neuroneista voidaan koota monimutkaisempia kokonaisuuksia, jotka koostuvat kolmesta kerroksesta: syötekerroksesta, piilokerroksesta ja ulostulokerroksesta [33]. Syötekerros saa ulkopuoliset syötteet, joista lasketut ulostulot menevät seuraavaan kerrokseen syötteiksi. Piilokerros saa nimensä siitä, että se ei näy ”ulospäin” koska sille ei voi antaa suoraan syötettä eikä se anna näkyvää ulostuloa. Piilokerros laskee tuloksen syötekerrokselta saamiensa arvojen mukaan ja antaa tuloksen eteenpäin ulostulokerrokselle. Ulostulokerros antaa ulostulon saamiensa arvojen mukaan. [34]

Neuroverkot, joissa on useampi kuin yksi piilokerros kutsutaan syväksi neuroverkoksi. Syväoppimisella tarkoitetaan koneoppimisen toteuttamista käyttämällä syviä neuroverkkoja [34]. Syväoppimisen etu perinteisiin menetelmiin verrattuna on sen parempi suorituskyky sekä kyky ratkaista monimutkaisempia ongelmia [33].

3.2. Tekoäly robotiikassa

Jotta robotti kykenee toimimaan autonomisesti, tarvitaan laitteeseen tekoäly, joka kykenee tekemään päätöksiä robotin toiminnasta. Tekoälyllä on neljä yleistä käyttötarkoitusta robotiikassa. Nämä ovat robotin näkökyky, asioista kiinni ottaminen, liikkumisen hallinta ja datan ymmärtäminen [35]. Jain ym. mukaan konenäön tarkoitus on kerätä informaatiota sille annetuista kaksiulotteisista kuvista [36]. Robotiikassa konenäöllä kyetään antamaan robotille kyky havainnoida sen ympäristöä, jotta robotti kykenee toimimaan sille toivotulla tavalla.

3.3. Shakkitekoäly

Täydellinen shakkitekoäly kykenisi laskemaan kaikki mahdolliset permutaatiot kummankin pelaajan kaikille siirroille, täten pystyen ohjaamaan pelin itselleen otollisia siirtoja pitkin loppuun asti ollen käytännössä voittamaton. Shakin korkean kompleksisuuden vuoksi tämä ei ole kuitenkaan mahdollista. Claude Shannon, artikkelissaan “Programming a Computer for Playing Chess”, esitti tyypillisen shakkipelin siirtojen konservatiiviseksi alamääräksi 10^{120} [37]. Tämän luvun kokoa usein havainnollistetaan vertaamalla sitä atomien määrään havaittavassa maailmankaikkeudessa, joka on noin 10^{79} [38].

Juuri tämä kompleksisuus tekee shakista erinomaisen kentän tekoälyn kehitykselle. Kun peliä ei voi ratkaista pelkällä raa’alla laskentateholla, on tekoälyn hyödyntäminen

ja kehittäminen käytännössä ainoa vaihtoehto vakuuttavan tietokonevastustajan luomiseksi.

Vaikka shakkia pelinä ei olekaan ratkaistu, shakkitekoälyt sisältävät usein valmiiksi tallennettuja loppupelejä. Näiden avulla tekoälyt pystyvät säästämään laskentatehoa ja pelaamaan täydellisesti, kun pelinappuloita on vain muutama jäljellä. Kaikki 3-5 nappulan ja osa 6 nappulan loppupeleistä on mahdollista tallentaa vain 21 gigatavun kokoiselle muistille [39].

3.3.1. Shakkitekoälyjen historia

Ensimmäisenä shakkitekoälynä voidaan pitää Alan Turingin ja David Champernownen suunnittelema Turochamp. Turochampia ei koskaan saatu ohjelmoitua tietokoneelle käytettäväksi. Se oli vain joukko sääntöjä, joiden avulla paras siirto kyetään analysoimaan. Turochamp onnistui voittamaan Champernownen vaimon shakissa. [40, 41]

Ensimmäisten alkukantaisimpien shakkiohjelmien suurin ongelma oli niiden rajattu toiminta ja tekoälyn yleinen heikkous verrattuna ihmispelaajaan. Kaikista alkeellisimmat eivät pystyneet pelaamaan kokonaista shakkipeliä uskottavasti. Ensimmäinen oikea shakkitietokone vuonna 1951 oli Dietrich Prinzin Ferranti-tietokoneella pyörivä ohjelma, joka kykeni ratkaisemaan shakkiongelmia, joissa ratkaisu oli shakkimatti ja sen täytyi olla kahden siirron päässä. Ohjelma kävi läpi kaikki mahdolliset siirrot, joten se oli varsinkin nykystandardeilla hidas. Yhden ongelman ratkaisuun meni noin 15 minuuttia. [40, 41]

Vuonna 1962 Alan Kotokin ja John McCarthyn luoma shakkiohjelma vastasi taidoiltaan jo noin 100 peliä pelannutta aloittelijaa [42]. Nykyään jopa tavalliselle tietokoneelle saatava shakkitekoäly pystyy pelaamaan suurmestarin tasolla [43].

Vuonna 1997 IBM teki historiaa, kun sen luoma shakkitietokone Deep Blue voitti sen aikaisen hallitsevan shakin maailmanmestarin Garry Kasparovin. Deep Bluen ”älykkyys” perustui suurimmalta osalta vain pelkkään raakaan laskentatehoon. Se kykeni analysoimaan keskimäärin 100 miljoonaa tilannetta sekunnissa ja noin kolmen minuutin haulla se kykeni analysoimaan kaikki siirrot keskimäärin 12.2 siirron päähän. [44] Modernit shakkitekoälyt ovat ohittaneet ihmiset pelitaidoillaan.

Arpad Elo kehitti shakkiin Elo-järjestelmän, jolla pelaajien taitotasoja voidaan kuvata Elo-luvun avulla. Elo-arvo määräytyy pelattujen pelien, voittojen sekä vastapelaajan Elo-arvon mukaan [45]. Tällä hetkellä Elo-arvon mukaan mitattuna paras julkisesti saatavilla oleva shakkitekoäly on Stockfish, Elo-arvonaan yli 3400 [46]. Parhaan ihmispelaajan Magnus Carlsenin Elo-arvo taasen on hieman yli 2800 [47].

Aikaisemmin shakkitekoälyt arvioivat siirtojen otollisuuden asiantuntijoiden ja suurmestarien avulla luoduilla heuristiikoilla. Nykyään koneoppimista on alettu käyttämään optimaalisen heuristiikan luomisessa. DeepMindin AlphaZerolla on saatu aikaan erittäin tehokkaita tekoälyjä. AlphaZero käyttää vahvistusoppimista ja syviä neuroverkkoja. Neljän tunnin opettamisen jälkeen AlphaZero oli parempi shakissa kuin Stockfish. AlphaZeron etu perinteisiin käsin tehtyihin heuristiikkoihin on taitojen lisäksi sen monikäyttöisyys. Perinteistä heuristiikkaa voi käyttää vain siinä pelissä mihin se on alun perin luotu, mutta AlphaZerolle on kyetty opettamaan shakin lisäksi myös shogin ja go:n pelaaminen ”tyhjästä”, eli sille annettiin ainoastaan pelin säännöt pohjatietoina. DeepMind on kehitellyt myös vastaavia oppivia algoritmeja myös muillekin kuin perinteisille vuoropohjaisille peleille. Vuonna 2019 DeepMindin

kehittämä AlphaStar voitti ammattilaispelaajia reaaliaikaisessa Starcraft 2 -strategiapelissä [48]. [49]

3.3.2. Päätöspuu

Shakkipelin päätöspuu luodaan tekemällä jokaiselle mahdolliselle siirrolle ja niistä koituville siirroille omat puunhaarat. Tämän jälkeen näille solmukohdille on mahdollista määrätä heuristiikkaa seuraten siirrosta koituva saatu hyöty. Päätöspuu on tämän jälkeen mahdollista käydä läpi halutulla algoritmilla, tuottaen optimaalisen reitin puun läpi ja mahdollistaen parhaan siirron valitsemisen reitin perusteella. Reitti on mahdollista etsiä millä tahansa hakupuualgoritmilla, mutta shakissa varsinkin minimax-pohjaiset algoritmit ovat yleisesti olleet suosittuja.

3.3.3. Sijainnin heuristinen arviointi

Algoritmisissa shakkitekoälyissä heuristista arviointia käytetään muuttamaan pelin tietynhetkinen tilanne diskreetiksi arvoksi ja vertailemaan eri siirtojen otollisuutta. Koska heuristiikka toimii shakkialgoritmin ”silminä”, on sen määrittely avainasemassa tekoälyn vakuuttavuutta määrittäessä. Hyväkin tekoäly on vain yhtä hyvä kuin sille syötetty tieto, ja heuristiikan ollessa epäsopiva ei algoritmi kykene arvioimaan pelitilannetta tarpeeksi syvällisesti pystyäkseen tekemään informoituja taktisia päätöksiä. Asettaessaan pelinappuloille numeerisia arvoja laadukas heuristiikka ottaa huomioon eri nappulatyyppien lisäksi monia muita ominaisuuksia kuten nappuloiden sijainnin ja määrän, pelin vaiheen ja kuninkaan turvallisuuden.
Shakkitekoälyissä heuristiikka usein perustuu suureen määrään yksilöllisesti pieniä muutoksia, jotka muuttavat nappuloiden arvoa tiettyjen kriteerien perusteella.

3.3.3.1. Materiaali

Pelinappuloiden materiaalinen arviointi asettaa eri pelinappuloille arvot niiden tyypin mukaan. Materiaalisia arvoja käytetään alustana, jonka päälle muu heuristiikka kehitetään. Nämä muut kehittyneemmät heuristiikat muokkaavat nappuloiden arvoa joko positiiviseen tai negatiiviseen suuntaan, riippuen arviointituloksesta. Yksinkertainen ja suosittu metodi materiaaliseen arviointiin on antaa sotilaille arvo 1, lähetille ja ratsulle arvo 3, tornille arvo 5 ja kuningattarelle arvo 9. Kuninkaan arvoksi asetetaan mielivaltaisesti vain joku erittäin suuri luku, kuten 10000. Larry Kaufman, artikkelissaan The Evaluation of Material Imbalances, ehdottaa hieman otollisemmaksi materiaaliseksi arvioinniksi antaa sotilaille arvo 1, ratsulle ja lähetille arvo 3.1/4, tornille arvo 5 ja kuningattarelle arvo 9.3/4 [50].

Nämä arvioinnit kuitenkin uhraavat tarkkuutta esitystavan yksinkertaistamiseksi. Tekoälyjen materiaalinen arviointi ei olekaan yleensä näin helposti luettavassa formaatissa, vaan esimerkiksi vahva shakkitekoäly Stockfish jakaa pelin keskivaiheessa nappuloille perusarvot 128 (sotilas), 782 (ratsu), 830 (lähetti), 1289 (torni) ja 2529 (kuningatar).

3.3.3.2. Pelin vaiheet

Shakkipelit jaetaan tyypillisesti kolmeen eri pelivaiheeseen: alku-, keski- ja loppupeliin. Eri pelivaiheet eivät ole tarkasti määriteltyjä mutta yleisesti alkupeli koostuu pelilaudan kehittämisestä, päättyen ja siirtyen keskivaiheeseen kun pelaajat ovat kehittäneet kaikki tai suurimman osan nappuloistaan. Loppuvaiheessa nappuloita on vain vähän jäljellä, ja kuningas tulee aktiiviseksi osaksi peliä. Kun yleisesti tunnustettuja vaiheita on useita, tarvitsevat shakkitekoälyt myös eri vaiheisiin eri lähestymistavat. Usein tekoälyt suorittavat alkuvaiheen pelinappuloiden kehittämisen seuraten niille valmiiksi tallennettuja avauskirjastoja, ja siirtyvät analyyttisempaan heuristiikkaan vasta pelin keskivaiheessa. Loppuvaiheeseen siirryttäessä algoritmi tekee konkreettisia muutoksia pelinappuloiden arvoon. Stockfish esimerkiksi lähes kaksinkertaistaa sotilaiden arvon.

Shakkitekoälyt ovat vahvimmillaan varsinkin pelin loppuvaiheessa. Mitä vähemmän pelinappuloita on jäljellä, sitä paremmalla todennäköisyydellä tekoäly pystyy ratkaisemaan pelin joko raa'alla laskentateholla tai tutkimalla käytössä olevia tallennettuja ratkaistuja loppupelejä.

3.3.3.3. Sotilaat

Sotilaat, matalan liikkuvuuden pelinappuloina, hyötyvät heuristiikasta suuresti. Sotilaita käytetään usein peleissä vihollisnappuloiden jahtaamisen sijasta suojaamaan tiettyjä nappuloita tai hallitsemaan tiettyä aluetta pelilaudalla. Tämänlaiseen analyysiin ei päästä pelkällä raa'alla siirtojen laskemisella, vaan tarvitaan kyseisille nappuloille räätälöity heuristiikka halutun käyttäytymisen varmistamiseksi.

Sotilaat saavat arvioinnissaan lisäarvoa varsinkin keskustan tärkeiden ruutujen hallinnasta, yhtenäisen sotilasketjun ylläpitämisestä ja kuninkaan suojaamisesta. Nappulan arvoa taas vähennetään mikäli siirrosta koituu sotilaiden linjan avautuminen, sotilaan jääminen yksin ilman suojausta tai sotilaan eteneminen pelilaudalla syvemmälle kuin muut sotilaat. Jo näillä melko yksinkertaisilla mutta tärkeillä evaluoinneilla varmistetaan älykäs heuristiikka sotilaille, jossa pelinappulat eivät avaa itseään liikaa vastustajan hyökkäyksille ja suojaavat pelilaudan takalinjaa.

Muita mahdollisia arviointimenetelmiä sotilaille ovat muun muassa vapaan linjan pelilaudan toiseen päähän ja hyvien tornitusmahdollisuuksien hakeminen sekä sotilaiden peräkkäisen asettelun ja tien avaamisen vastustajan sotilaille välttäminen.

3.3.3.4. Tila ja pelinappuloiden sijainti

Shakkilaudalla ja heuristisessa arvioinnissa kaikki ruudut eivät ole samanarvoisia. Pelaaja joka hallitsee suurempaa pelialuetta ja pelilaudan keskustaa saa myös enemmän mahdollisuuksia hyökätä ja vapautta liikuttaa pelinappuloitaan, mahdollistaen pelitilanteen kehityksen itselleen yhä otollisempaan suuntaan. Hallittu tila voidaan arvioida esimerkiksi ottamalla huomioon sotilaiden sekä suojassa olevien nappuloiden syvyyden ja muuttaa näiden perusteella nappulan arvoa sen sijainnin perusteella. Yksi tapa määrittää ruuduille yksilölliset arvot on tehdä jokaiselle eri nappulatyypille oma taulukko pelilaudasta ja määrätä näiden alkioille painoarvot jotka kannustavat haluttua pelityyliä. Yksinkertaisimmassa muodossaan tämä saattaa olla vain Manhattan-etäisyys pelilaudan keskustasta. Stockfish antaa c3-f6 suorakulmiossa

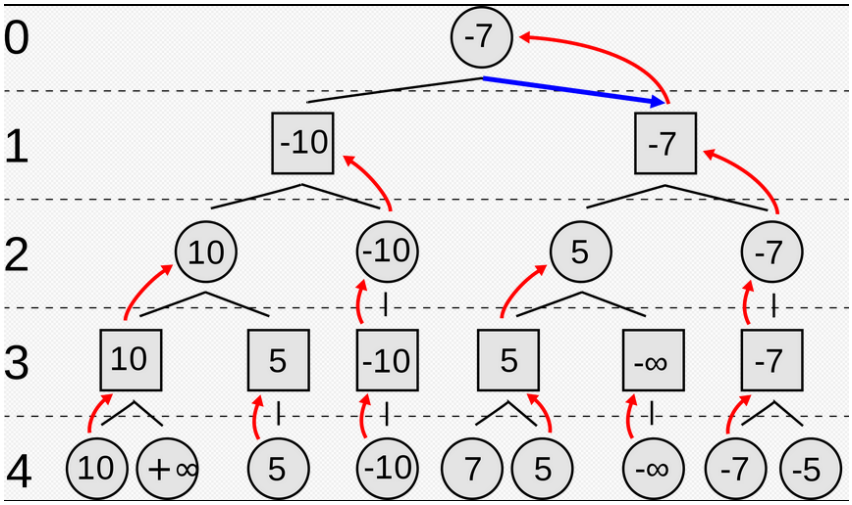
sijaitseville sotilaille painotettua lisäarvoa parantaakseen aloitustaan ja kontrolliaan pelitilanteesta.

3.3.3.5. Heuristiikka koneoppimisessa

Koneoppimisella luodut shakkitekoälyt uhmaavat odotuksia siinä mielessä, että niillä ei ole samanlaista heuristiikkaa kuin mihin on totuttu perinteisissä algoritmisissa tekoälyissä. Perinteisesti heuristiikat ovat käsin ohjelmoituja funktioita, mutta koska koneoppiva tekoäly sopeutuu sille syötettyyn dataan, sillä ei ole varsinaista omaa heuristiikkafunktiota. Tavat, joilla koneoppivat tekoälyt analysoivat pelilautaa ja pelitilannetta perustuvat neuroverkkoihin ja ovat usein ihmiselle käsittämättömiä. Tämä monimutkaisuus kuitenkin mahdollistaa paljon syvemmän analyysin luomisen kuin mihin parhaatkin ihmisohjelmoijat kykenisivät. Shakkitekoälyjen kärkirintama ja moderni kehitys keskittyykin juuri koneoppimiseen.

3.3.4. Minimax-algoritmi

Minimax-algoritmi voidaan ajatella kahden funktion, minimoinnin ja maksimoinnin, yhdistelmänä. Maksimointifunktiota käytetään tekoälyn omalla vuorolla ja minimointifunktiota vastustajan vuorolla, päämääränä tuottaa vuoron mukaan joko mahdollisimman suuri pistevoitto tai mahdollisimman pieni pistehäviö. Algoritmi käy tällä tavalla rekursiivisesti koko päätöspuun läpi haluttuun syvyyteen saakka ja valitsee pienimmistä minimointifunktion ja suurimmista maksimointifunktion tuloksista optimaalisimman reitin. Naiivi minimax käy kuitenkin läpi paljon turhia solmuja päätöspuussa ja shakissa jokaiselle siirrolle optimaalisen seuraavan siirron etsiminen nostaa etsimisaikaa eksponentiaalisesti, minkä takia sille on kehitetty useita hyödyllisiä optimisaatiotekniikoita, näistä suosituimpana alfa-beta -karsinta. Kuvassa 3 on minimax-algoritmin hakupuu, joka lopulta päätyy arvoon -7. Tasoilla 0 ja 2 algoritmi valitsee arvoista suuremman ja tasoilla 1 ja 3 arvoista pienemmän.



Kuva 3. Minimax-algoritmin päätöspuu[3]

[3] Nuno Nogueira (2006), CC-BY: https://creativecommons.org/licenses/by-sa/2.5/deed.en

3.3.5. Syvyys

Shakin kompleksisuuden vuoksi päätöspuun syvyys kasvaa eksponentiaalisesti. Claude Shannon arvioi mahdollisten eri shakkipelien lukumääräksi 10^120. Taulukossa 1 on esitetty mahdollisten erilaisten shakkipelien lukumäärä siirtojen edetessä.

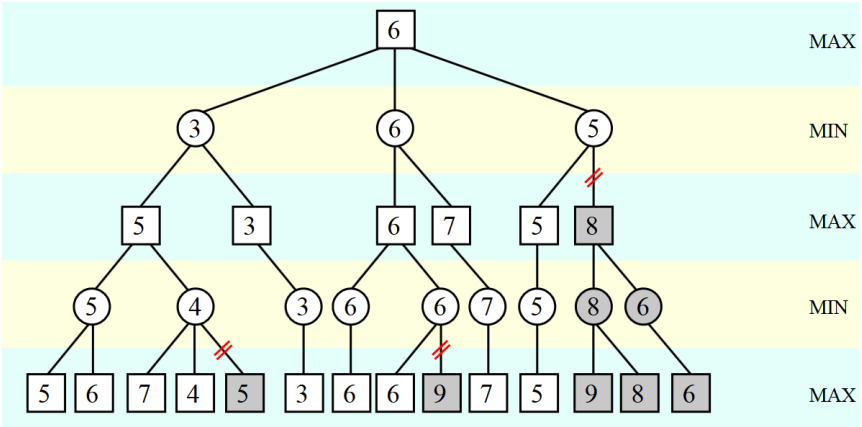
Taulukko 1. Mahdollisten eri pelien lukumäärä shakissa

Siirtojen määrä	**Mahdollisten eri pelien lukumäärä**
1	20
2	400
3	8,902
4	197,281
5	4,865,609
6	119,060,324
7	3,195,901,860
8	84,998,978,956
9	2,439,530,234,167
10	69,352,859,712,417

3.3.6. Alfa-beta -karsinta

Alfa-beta -karsinta on optimisaatiotekniikka minimax-algoritmin tuottaman päätöspuun supistamiseksi. Alfa-beta -karsinnalla on mahdollista välttää päätöspuun haarat joita minimax ei tule lopussa valitsemaan, käymättä niitä kuitenkaan kokonaan läpi.

Alfa-beta -karsinnan tehokkuus riippuu siitä, missä järjestyksessä algoritmi käy päätöspuun solmut läpi. Mikäli algoritmi vierailee puun haaroissa huonoimmassa mahdollisessa järjestyksessä, muutoksia ei välttämättä tapahdu ollenkaan. Keskimääräisesti alfa-beta -karsinta on kuitenkin aina minimax-algoritmia tehokkaampi. Kuvassa 4 minimax-algoritmi ei käy läpi harmaiksi värjättyjä alkioita, koska ne eivät voi antaa parempaa tulosta kuin jo tutkitut alkiot.



Kuva 4. Minimax-algoritmin päätöspuu alfa-beta -karsinnalla[4]

[4] Jez9999 (2007), CC-BY: https://creativecommons.org/licenses/by-sa/3.0/deed.en

3.3.7. Alfa-beta -karsinnan optimisaatiotekniikoita

Kuten alfa-beta -karsinta on minimaxille kehitetty optimisaatiotekniikka, myös alfa-beta -karsinnalle itselleen on kehitetty useita etsintää nopeuttavia tai laskentatehoa säästäviä lisäosia.

3.3.7.1. Transpoositaulukko (Transposition table)

Shakissa transpoosi tarkoittaa pelitilannetta, johon on mahdollista saapua monien eri siirtojen sarjan kautta. Transpoositaulukko toteutetaan tallentamalla algoritmin läpikäytyjen transpoosien minimax-tulos hajautustauluun, käyttäen avaimena Hash-algoritmilla luotua tiivistettä pelitilanteesta. Mikäli algoritmi löytää tulevissa etsinnöissään saman pelitilanteen voi se hakea sekä sen että sen aiemmin laskettujen alahaarojen minimax-tuloksen. Näin algoritmi kykenee parhaassa tapauksessa löytämään optimaalisen siirron lähes välittömästi ja ohittamaan laskemisen kokonaan.

Koska alfa-beta -karsinnan tehokkuus riippuu suuresti siitä missä järjestyksessä algoritmi käy päätöspuun läpi, transpoositaulukosta on myös hyötyä päätöspuun etsinnän aloittamisen suuntaamisessa. Mikäli transpoositaulukossa on aiemmin löydetty laadukkaalta vaikuttava siirto, päätöspuun tutkimisen aloittaminen siitä parantaa optimaalisen siirron aikaisin löytymisen mahdollisuutta satunnaiseen tutkimiseen verrattuna. [51]

3.3.7.2. Iteroiva syventäminen (Iterative deepening)

Iteroiva syventäminen on optimisaatiotekniikka päätöspuun etsintään käytetyn ajan hallitsemiseen. Syventämisessä jokainen päätöspuun taso käydään kerrallaan läpi ja tasolta löydetty optimaalinen reitti merkataan ylös ennen haun syventämistä seuraavaan tasoon. Syventämisprosessille asetetaan jokin maksimiaika ja tämän ajan täyttyessä tai haun päästessä haluttuun syvyyteen haku keskeytetään ja algoritmi palauttaa senhetkisen parhaan siirron. Iteroivaa syventämistä on ajanhallinnan lisäksi myös mahdollista käyttää päätöspuun haarojen tutkimisen suuntaamiseen, nopeuttaen hakua. [52]

3.3.7.3. Aspiraatioikkuna (Aspiration window)

Aspiraatioikkuna nopeuttaa hakua rajaamalla alfan ja betan raja-arvoja, samalla pienentäen myös tarvittujen laskujen joukkoa ja lyhentäen päätöspuun tutkimiseen käytettyä aikaa. Raja-arvoja ei kuitenkaan ole mahdollista asettaa tarkasti, joten rajaus pohjimmiltaan perustuu aiemman tiedon perusteella tehtyyn informoituun arvaukseen. Mikäli tämä arvaus on väärin, tarkistetaan oliko luku raja-arvon ala- vai yläpuolella, minkä jälkeen vastaavaa raja-arvoa laajennetaan tarvittuun suuntaan ja haku suoritetaan uudelleen. [53]

3.3.7.4. Tappajaheuristiikka (Killer heuristic)

Tappajaheuristiikka on päätöspuun tutkimista suuntaava optimisaatiotekniikka. Heuristiikka toimii oletuksella, että alfa-beta -karsinnan aiemmin löytämät laadukkaat

siirrot ovat tulevaisuudessakin yhä laadukkaita, vaikka ne tehtäisiin toisesta (samanlaisesta) paikasta. Ohjelma tallentaa näitä siirtoja tulevaa tarkastelua varten ja siirron ollessa laillinen aloittaa etsintänsä niistä, nostaen potentiaalisesti suosiollisten siirtojen hakuprioriteettia. [54]

3.4. Shakkitekoälyjen ongelmakentät

Shakki on tarpeeksi monimutkainen peli, että sen täydellinen matemaattinen ratkaisu on käytännössä mahdotonta. Pelin ratkaisulla tarkoitetaan sitä, että kone kykenisi voittamaan pelin aina. Shakissa on yli 10^{40} mahdollista eri tilannetta, joten kaikkien mahdollisten siirtojen läpikäyminen on mahdotonta [43]. Esimerkiksi tammi julistettiin 2007 heikosti ratkaistuksi eli tekoäly pystyy ratkaisemaan lopputuloksen, sekä strategian, jolla tähän päädytään [55]. Shakki jakautuu kolmeen eri vaiheeseen: avaukseen, keskipeliin sekä loppupeliin. Pelillisesti katsottuna shakkitekoälyt ovat yleensä parhaimmillaan keskipelissä. Varsinkin loppupelissä siirrot ovat monesti niin hienovaraisia, ettei tietokone kykene niitä ymmärtämään [43]. Kasparovin mukaan shakkitietokoneiden yhdeksi ongelmaksi pelättiin muodostuvan sen, että peleistä tulee pitkäveteisia kun siirrot ovat liki täydellisiä [56]. Kasparov on kuitenkin sitä mieltä, että Googlen oppiva tekoäly AlphaZero antaa merkkejä siitä, että näin ei ole. AlphaZero on kehittänyt omanlaisen suhteellisen aggressiivisen pelityylin, joka on ollut tehokasta [56].

4. TOTEUTUS

Työn tarkoituksena oli tuottaa vakuuttava, shakkia laillisesti pelaava tekoäly rajatun laskentatehon ympäristölle. Tähän pyrittiin iteroimalla alkeellista päätöspuun etsintäalgoritmia, pyrkimyksenä asteittaisten parannusten kautta tuottaa kilpailukykyinen shakkitekoäly. Kehitysympäristönä käytettiin Raspberry Pi 3 -minitietokonetta.

Tekoäly toteutettiin käyttäen Pythonia. Pelitilanteen seuraamista ja sääntöjä varten käytettiin avoimen lähdekoodin python-chess -kirjastoa. Kirjasto tukee Universal Chess Interface -protokollaa, joka on shakkitietokoneille suunniteltu protokolla, jolla ne saadaan toimimaan yhdessä käyttöliittymien kanssa. Pelilauta on Forsyth-Edwards Notation -standardin mukainen. Esimerkiksi shakkipelin alkutilanne FEN-notaatiolla on ”rnbqkbnr/pppppppp/8/8/8/8/PPPPPPPP/RNBQKBNR w KQkq – 0 1”. Tulkattava ohjelmointikieli kuten Python ei ole optimaalinen vaihtoehto hyvän hyötysuhteen omaavan tekoälyn tekemiseen, mutta kielen valmiit kirjastot mahdollistivat itse shakkipelin ohjelmoimisen välttämisen, mikä olisi ollut pakollista muilla ohjelmointikielillä.

4.1. Raspberry Pi 3

Raspberry Pi:n laskentateho on erittäin rajattu, joten päätöspuun hakunopeuteen pyrittiin vaikuttamaan tehostamalla hakualgoritmeja. Taulukossa 2 on esitetty Raspberry Pi 3:n tekniset tiedot.

Taulukko 2. Raspberry Pi 3 -minitietokoneen tekniset tiedot

SoC	Broadcom BCM2837
CPU	4x ARM Cortex-A53, 1.2Ghz
GPU	Broadcom VideoCore IV
RAM	1GB LPDDR2 (900 MHz)
Networking	10/100 Ethernet, 2.4Ghz 802.11n wireless
Bluetooth	Bluetooth 4.1 Classic, Bluetooth Low Energy
Storage	microSD
GPIO	40-pin header, populated
Ports	HDMI, 3.5mm analogue audio-video jack, 4x USB 2.0, Ethernet, Camera Serial Interface (CSI), Display Serial Interface (DSI)

4.2. Shakkitekoäly

Tekoälyn hakualgoritmiksi valittiin minimax johon lisättiin alfa-beta -karsinta, avauskirjojen luku, iteroiva syventäminen sekä transpoositaulukot.

Minimax-algoritmi valittiin toteutuksen aloituspisteeksi varsinkin sen peliteoriallisen merkittävyyden ja laskentatehollisen hyötysuhteen vuoksi. Algoritmi on suosittu etenkin kahden pelaajan nollasummapelejä arvioidessa, mikä tekee algoritmista

luonnollisen valinnan pohjaksi juuri shakkitekoälyä varten. Algoritmi on myös hyvin modulaarinen, mikä mahdollistaa erilaisten optimisaatiotekniikoiden nopean lisäämisen ja vaihtelun. Tämä modulaarisuus mahdollistaa myös alkuperäisen algoritmin tehokkuuden vaiheittaisen nostamisen ja näiden eri toteutusten vertailun keskenään.

Nykyaikana shakkitekoälyjen terävin kärki kohdistuu perinteisten algoritmien sijaan koneoppimisen hyödyntämiseen tekoälyjä luodessa. Työssä käytetty laskentateho oli kuitenkin liian rajallinen että koneoppimisen tarvitsema iteraatioprosessi olisi ollut varteenotettava vaihtoehto tekoälyn algoritmiselle toteutukselle. Täten koneoppiminen rajattiin pois mahdollisena ratkaisuna tutkimusongelmaan.

4.2.1. Minimax

Minimax-algoritmi toteutettiin Pythonilla. Lautatilanteen seuraamiseen käytettiin python-chess -kirjaston board-funktiota ja kaikki lailliset siirrot saatiin valmiilla funktiolla, joka antoi lautatilanteen kaikki lailliset siirrot. Lailliset siirrot saatiin listaan, joka oli aina tietyssä järjestyksessä. Listan sekoituksella saimme algoritmille hieman satunnaisuutta, koska jos algoritmi löytää kaksi yhtä hyvää siirtoa niin se valitsee niistä ensimmäiseksi löydetyn.

4.2.2. Alfa-beta -karsinta

Alfa-beta -karsintaa käyttäessä laillisten siirtojen listaa sekoittamalla on mahdollista saada nopeampia hakuaikoja, riippuen siitä sekoitetaanko siirrot vain satunnaisesti vai tietyssä järjestyksessä.

4.2.3. Avauskirjat

Tekoälyllä oli yleisesti vaikeuksia pelin avauksissa, joten siihen implementoitiin PolyGlot-avauskirjat. Ennen kuin tekoäly aloittaa tilanteen laskemisen minimax-algoritmilla, se tarkistaa onko kyseinen pelitilanne jo tallennettuna avauskirjaan. Avauskirja antaa tilanteeseen sopivat siirrot, sekä niiden painoarvot. Toinen avauskirjan tekoälylle tuoma hyöty oli sen nopeus. Avauskirjoja käyttäessä ensimmäisten siirtojen laskenta-aika oli käytännössä olematon, joten ne vähensivät huomattavasti sitä aikaa, jonka pelaaja joutuu odottamaan siirtoja koko pelin aikana.

Osassa avauskirjoja ongelmaksi muodostui se, että ne olivat liian optimaalisia. Nämä avauskirjat tarjosivat joka tilanteeseen vain yhden siirron: parhaan mahdollisen. Kyseiset avauskirjat olisivat olleet parempia jos tarkoituksenamme olisi ollut tehdä pelillisesti mahdollisimman tehokas tekoäly, koska eri siirtojen vähyyden sijaan ne sisälsivät siirtoja huomattavasti syvemmälle peliin. Tämä olisi kuitenkin tehnyt avauksista turhan kaavamaisia, joten valitsimme mieluummin avauskirjan, joka tarjosi useampia eri siirtoja yhdelle tilanteelle. Näistä siirroista valittiin satunnaisesti yksi käyttäen hyväksi avauskirjan niille antamia painoarvoja. Avauskirjaksi valittiin lopulta Brainfishin Polybook. Toinen ongelma ilmeni, kun tekoälyn vastapelaaja teki siirtoja, joita ei löytynyt avauskirjoista. Tämä ongelma toistui yleensä silloin, kun pelaaja oli täysin amatööri eikä osannut yhtään avaussiirtoa. Ongelma johtui siitä, että

avauskirjaan ei ole oleellista laittaa siirtoja, joita tapahtuu vain erittäin harvoin oikeassa pelitilanteessa. Tämä ongelma voitaisiin korjata erilaisella implementaatiolla, jossa tekoäly ymmärtäisi paremmin, milloin vaihtaa pois avauskirjasta hakualgoritmiin. Tekoälyn tulisi huomata, milloin vastapelaajan siirto on avauksen kannalta turha ja milloin siirto vaatisi sen huomiota sekä tarkempaa analyysiä.

4.2.4. Iteroiva syventäminen ja transpoositaulukot

Tekoälyyn toteutettiin iteroiva syventäminen, joka toimii tarvittaessa myös ilman aikarajoituksia. Heuristiikka ei ottanut huomioon sitä, kuinka syvältä paras arvo löydettiin. Ongelmaksi tämä muodostui tilanteissa, jossa algoritmi löytää esimerkiksi shakkimatin syvyydestä 3 ensin ja sen jälkeen syvyydestä 1. Koska siirrot johtavat samaan lopputulokseen niin algoritmi valitsee sen siirron, jonka se löysi ensin. Iteroivalla syventämisellä algoritmi tutkisi ensin 1 syvyyden kokonaisuutena ja löytäisi shakkimatin sieltä eikä sen tarvitsisi edes analysoida tilannetta yhtään syvemmälle. Iteroivalla syventämisellä siis saatiin lyhennettyä hakuaikoja pelitilanteissa, jossa algoritmi löytää siirron joka johtaa shakkimattiin.

Algoritmi oli erittäin heikko loppupeleissä, koska hakualgoritmin syvyys ei usein riittänyt löytämään sopivaa siirtoa. Ihmispelaajaa vastaan tämä oli iso ongelma, koska nappuloiden vähentyessä ihminen kykenee miettimään tilanteita syvemmälle kuin matalan tason tekoäly. Tämä pyrittiin ratkaisemaan asettamalla iteroivalle syventämiselle aikarajoitus ja vähimmäissyvyys, joiden mukaan se hakee siirrot.

Transpoositaulukot toteutettiin samalla iteroivan syventämisen kanssa, koska iteroiva syventäminen ei toimi suunnitellusti ilman sitä. Tämä johtuu siitä että iteroivassa syventämisessä käydään samat tilanteet kokoajan läpi, mutta joka kerralla yksi kerros syvemmälle. Transpoositaulukolla algoritmi sai pidettyä kirjaa tilanteista, jotka se oli jo käynyt. Täten joka kerta kun algoritmin etsintä syveni yhden tason, se pystyi ottamaan transpoositaulukosta edellisen syvyyden lasketun siirron sekä arvon. Transpoositaulukon avaimeksi valittiin pelilaudan FEN-notaatiosta muodostettu uniikki Zobrist hash-avain. Zobrist hash-funktio on lautapelitilanteita varten luotu hash-tekniikka, jolla pyritään vähentämään muistinkäyttöä käyttämällä tavallisen FEN-notaation mukaisen merkkijonon sijaan avaimena hash-funktiolla saatua lukua. Transpoositaulukkoon tallennettin tilanteen mukainen paras siirto, tämän siirron arvo, syvyys sekä tieto siitä onko tallennettu siirto ja arvo tarkka, yläraja vai alaraja. Arvo on tarkka silloin kun kaikki siirrot on käyty siltä syvyydeltä ja tämä arvo on todettu parhaaksi. Jos haussa ei ehditä löytää parasta arvoa niin se merkitään joko ylä- tai alarajaksi jotta arvoa voidaan käyttää joko alfana eli haun alarajana tai betana eli haun ylärajana, kun algoritmi tutkii tilannetta seuraavan kerran. Tällä pyritään vähentämään asioiden laskemista turhaan uudestaan tilanteessa, jossa algoritmi ei ehdi laskea kaikkia mahdollisia siirtoja tietyllä syvyydellä. Lisäksi tällä saadaan aikaan se, että algoritmin laskeminen ei mene koskaan hukkaan vaan kaikki laskemiseen käytetty aika saadaan hyödynnettyä.

4.2.5. Heuristiikka

Tekoälyn materiaalinen arviointi tehtiin taulukon 3 mukaan. Kuninkaan arvoksi asetettiin 0, koska kumpikaan kuningas ei koskaan poistu laudalta, joten sen arvioiminen tässä yhteydessä olisi täysin turhaa. Lautatilannetta arvioitaessa toisen

pelaajan nappulat arvioidaan samoille luvuilla, mutta negatiivisina. Arviointi ottaa myös huomioon lautatilanteen lailliset siirrot. Tällä pyrittiin siihen, että tekoäly pelaisi aktiivisempaa peliä ja pyrkisi etsimään uusia tilanteita samalla kun siirtojen määrä kasvaa. Shakkimatti arvioidaan +- 100000 riippuen siitä, onko kyse maksimoitavasta vai minimoitavasta pelaajasta. Arviointi ottaa lisäksi huomioon tasapelit. Jos tekoäly mielestään johtaa peliä, se pyrkii välttelemään tasapeliä ja vastaavasti tappioasemasta se yrittää saada pelin päättymään tasapeliin. Jokaiselle nappulalle luotiin matriisit, joiden avulla nappuloiden sijoittuminen laudalla voitiin arvioida. Tämä paransi tekoälyn avauksia ja yleistä sijoittumista pelilaudalla.

Taulukko 3. Tekoälyn materiaalinen arviointi

	Nappula	Arvo
	Kuningas	0
	Kuningatar	100
	Torni	50
	Lähetti	30
	Ratsu	30
	Sotilas	10

4.2.6. IRC -botti ja Lichess -botti

Aluksi tekoälyä pystyi testaamaan vain syöttämällä sille siirtoja kirjoittamalla ne komentorivin/ohjelmointiympäristön kautta tekstinä. Tekoälyn testaamista muilla pelaajilla haluttiin helpottaa, joten tekoälylle kehitettiin kaksi erilaista tapaa pelata sitä vastaan internetin välityksellä. Pelatut pelit kirjattiin talteen ja niiden avulla tutkittiin esimerkiksi siirtojen hyvyyttä ja tekoälyn pelitapaa. Tätä tietoa käytettiin tekoälyn toimivuuden parantamiseen. Lisäksi pelaajilta kysyttiin pelin jälkeen mielipiteitä ja ideoita tekoälyn parantamiseksi.

Tekoälylle toteutettiin Pythonin socket-kirjastolla kyky pelata IRC-pikaviestintäpalvelun kautta. IRC-botti toteutettiin freenode IRC-verkossa. Bottia käytettiin enimmäkseen tekoälyn toiminnan esittelemiseen projektitapaamisissa. Käyttöliittymä oli tekstipohjainen, joten siirtojen tekeminen oli kömpelöä ja laudan ruutujen sekä nappuloiden oikein havainnoinnissa oli vaikeuksia. IRC-botin kehittäminen jätettiin vähälle, koska Lichess soveltui tarkoitukseemme paljon paremmin. Suurimpina syinä olivat Lichessin graafinen käyttöliittymä sekä se, että pelitilanteet olivat saatavilla rajapinnasta helposti tilanteissa, joissa botti esimerkiksi kaatui tai käynnistettiin uudestaan kesken pelin.

Projektia varten tekoälylle luotiin oma profiili www.lichess.org-shakkipalvelimelle, jonka kautta sitä vastaan pystyi pelaamaan kuka vain. Kaikki tekoälyn tätä kautta pelaamat pelit säilyvät sivustolla tallessa. Lisäksi sivusto pitää

kirjaa siirtoihin käytetystä ajasta ja siihen on implementoitu mahdollisuus analysoida koko pelin kaikki siirrot käyttäen Stockfish 10+ -shakkitekoälyä.

Lichessin valinta botin alustaksi johtui monesta syystä. Näitä ovat esimerkiksi avoin rajapinta, jonka avulla tekoälyn pelaaminen on ylipäätänsä mahdollista, sekä toimiva mobiiliversio pelaamisen helpottamiseksi. Lichess sisältää myös keskustelutoiminnon pelaajien välillä, minkä avulla on mahdollista myös testata mahdollisia interaktiivisia toimintoja.

Varsinainen toteutus tehtiin käyttämällä Pythonille tehtyä Berserk-kirjastoa, jonka avulla mahdollistettiin tekoälyn yhdistäminen Lichessin rajapintaan. Ohjelma jakoi eri pelaajia vastaan käydyt pelit eri prosesseiksi, jotta laskenta voitiin jakaa useammalle prosessorille kerrallaan. Botti laitettiin Blanko ry:n Bugi-palvelimelle pyörimään lähes kolmeksi viikoksi ja siitä saatuja tuloksia sekä palautetta käytettiin toiminnan parantamisessa.

4.2.7. Toteutuksen ongelmakenttä

Toteutusta laadittaessa yhdeksi ratkaisevaksi ongelmatekijäksi nousi konenäön puutteellisuus. Konenäkö ei pystynyt tunnistamaan eri nappuloita toisistaan vaan ohjelman saama pelilaudan tilanne koostui värein erotelluista, mutta rooliltaan tuntemattomista, nappuloista. Tämä ongelma ratkaistiin ohjelmallisesti antamalla algoritmille pelin alussa laudan virallinen alkutilanne nappuloineen, ja tämän jälkeen aina vertaamalla konenäön antamaa pelilaudan tilannetta edelliseen pelilaudan tilanteeseen. Täten pelitilanteita vertaamalla oli mahdollista selvittää mikä nappula oli liikkunut ja määrittämään oliko nappulan siirto laillinen.

Raspberry Pi oli myös ongelmallinen toteutuksen kannalta. Taulukosta 2 nähdään että varsinkin sen prosessorin kellotaajuus on nykystandardeilla erittäin hidas, vain 1.2Ghz. Tämän hidasti hakuaikoja huomattavasti ja asetti rajoitteita algoritmin syvyydelle.

Toteutuksen luultavasti merkittävin ongelma-alue oli itse pelaaja, jota vastaan tekoäly tulisi vääjäämättä pelaamaan. Ihmisten huomiokyky on rajallinen, minkä vuoksi tekoälyn maksimaalisen laskenta-ajan pitämistä minimissä pidettiin toteutuksen kannalta erittäin tärkeänä ja sille asetettiin rajoitteet pelaajan odotusajan vähentämiseksi. Tämän lisäksi tekoälyä testatessa IRC:n ja Lichessin kautta huomioitavaksi seikaksi otettiin myös tekoälyn taito keskimääräistä ihmispelaajaa vastaan pelatessa. Liian huonoa tekoälyä vastaan pelaaminen ei ole pelaajalle vaivan arvoista, mutta kääntöpuolena liian hyväksi havaittu tekoäly voi olla ihmiselle turhauttava vastustaja.

4.3. Testaus

Testaus suoritettiin kaksiosaisesti. Ensimmäisessä osassa tekoälyn kyvykkyyttä testattiin ihmispelaajia vastaan internetin välityksellä. Toisessa osassa testattiin tekoälyn laskenta-aikoja. Tämän saavuttamiseksi toteutettiin ohjelma, joka pelautti tekoälyn eri versioita automaattisesti toisia vastaan ja tallensi siirtoihin kuluneet ajat tekstimuodossa.

4.3.1. Pelitestaus

Kuten aiemmin mainittu, tekoälyn pelaamista testattiin käyttäen IRC -bottia sekä Lichess-bottia. Suurin osa peleistä pelattiin lisäosatonta alfa-beta -karsintaa vastaan syvyydellä 3. Palautteen mukaan yleinen ongelma oli tekoälyn huonous alku- ja loppupelissä. Alkupelissä ongelmana oli lisäksi jo aikaisemmin mainittu avausten kankeus, sekä tekoälyn laskenta-ajat olivat myös liian pitkiä. Varsinkin tilanteissa joissa paras siirto oli helposti ihmisen nähtävissä tekoälyn olisi pitänyt tehdä siirto nopeammin. Positiivista palautetta sai tekoälyn kyky arvioida pelilaudan materiaalista arvoa, mikä näkyi siinä että tekoäly käytti hyväksi erilaisia haarukoita, eli siirtoja joissa uhataan useampaa nappulaa kerrallaan. Tämän lisäksi se osasi myös tehokkaasti käyttää hyväksi vastapelaajansa virheitä. Tämä oli kuitenkin varsinkin heikompien pelaajien mielestä huono asia, koska tekoälyn vaikeusastetta ei pystynyt säätämään ja tekoäly valitsi aina parhaan siirron. Testatessa tekoälyä eri syvyyksillä tulimme siihen tulokseen, että tekoäly oli pelillisesti aivan liian huono syvyyden ollessa alle 3.

Keskipelissä tekoäly oli pelillisesti parhaimmillaan. Ongelmana oli kuitenkin erittäin pitkät hakuajat ja erityisesti tilanteet, jossa tekoäly arvioi kaikki mahdolliset siirrot suunnilleen samanarvoisiksi. Hakuajat olivat näissä tapauksissa todella pitkiä, koska tekoäly kävi yleensä läpi lähes koko hakupuun. Tämänlaiset tilanteet olivat harvinaisia, mutta osoittivat kuitenkin sen, että arviointifunktiota pitäisi vielä parantaa tällaisten tilanteiden välttämiseksi.

Loppupelissä tekoälyn heikkous johti siihen, että tämä usein teki tappioon johtavan siirron. Tämä johtui tekoälyn kiinteästä syvyydestä. Loppupelissä jossa nappuloita on vähän, ihmispelaaja kykenee arvioimaan paljon syvemmälle kuin keskipelissä, jossa melkein kaikki nappulat ovat vielä laudalla. Useimmat tekoälyn tappioista johtuivat siitä, että ihmispelaaja näki shakkimatin 4-5 siirron päästä ja tekoäly ei kyennyt reagoimaan tähän, koska se ei pystynyt näkemään tilannetta niin syvälle. Kuvassa 5 hyvä esimerkki tällaisesta tilanteesta. Tekoäly pelaa tilanteessa valkoisella. Jos valkoinen pelaaja ei reagoi, musta pelaaja tekee shakkimatin kahdella siirrolla (qe8 b4 re8). Tekoäly ei kuitenkaan nähnyt niin pitkälle ja teki siirron qd8 joka siis johti tappioon. Iteroivalla syventämisellä tekoäly saataisiin ainakin teoriassa tutkimaan tilannetta syvemmälle. Varsinkin jos pelilaudalla on pakotettuja siirtoja, eli pelaajalla on ainoastaan 1-2 laillista siirtoa, tilanteita olisi mahdollista analysoida huomattavasti syvemmälle.

Kuva 5. Esimerkkipelitilanne.

4.3.2. Laskentatehollinen testaus

Tekoälyn laskentatehoa testattiin pelauttamalla sen eri versioita tiettyä tekoälyä vastaan. Aluksi tekoälyt asetettiin pelaamaan itseään vastaan eri syvyyksillä, mutta tällä metodilla saaduissa testaustuloksissa oli paljon siirtoja joita ei syntyisi pelatessa ihmisvastustajaa vastaan. Tuotettu data ei siis ollut tutkimuksen aiheen huomioon ottaen vertailukelpoista ja testaus päätettiin suorittaa uudestaan sopivammalla testausasetelmalla.

Testausta uudelleen suoritettaessa algoritmeille päätettiin antaa staattinen vastustaja, jota vastaan kaikkia tekoälyjä pelattiin. Vertailukohteeksi valittiin syvyydellä 2 pelaava alfa-beta -karsinta. Algoritmi kykeni pelaamaan riittävän hyvin shakkia ilman liian suuria odotusaikoja siirtojen välillä, millä varmistettiin tarpeeksi pitkä pelien kesto riittävän datan keräämiseksi nostamatta kuitenkaan testausaikoja tarpeettoman suuriksi. Tällä metodilla samalla syvyydellä pelaavat algoritmit olivat myös erinomaisesti vertailtavissa, koska tekoälyt käyttivät samaa heuristiikkaa. Analysoidut pelit pelattiin Raspberry Pi 3-minitietokoneella. Testaamisen automatisoinniksi tekoälyn ohjelmaan lisättiin tapa aloittaa ja pelata haluttuja pelejä automaattisesti sekä tallentaa pelien aikana tapahtuneet siirrot ja lopputulokset talteen pelikohtaisesti.

4.3.2.1. Minimax

Tavallinen minimax-algoritmi soveltuu erittäin huonosti shakkitekoälyn hakualgoritmiksi sen hitauden takia, mutta sen tehokkuus testattiin silti siksi että saataisiin hyvä vertailukohta muihin paranneltuihin algoritmeihin. Taulukossa 4 on tilastoja minimax-algoritmin peleistä eri syvyyksillä. Jo syvyydellä 3 hakuajat ovat aivan liian pitkiä ja syvyydellä 4 testaus keskeytettiin, koska siirroissa meni niin kauan, että yhden pelin pelaamisessa testiä varten olisi kulunut monta päivää.

Taulukko 4. Minimax-algoritmin laskenta-ajat

Syvyys	Keskiarvo	Mediaani	Keskihajonta
1	2.025s	2.075s	0.915s
2	49.683s	56.570s	37.272s
3	3453.478s	3466.857s	1341.391s

4.3.2.2. Alfa-beta -karsinta ilman lisäosia

Taulukossa 5 on tilastot tekoälystä peleistä alfa-beta -karsinnan kanssa. Tuloksista nähdään, että hakusyvyyden noustessa myös ero tavalliseen minimax-algoritmiin kasvaa. Esimerkiksi 3. Syvyydellä minimax ilman alfa-beta -karsintaa on yli 25 kertaa hitaampi kuin sen kanssa. Algoritmi ei kuitenkaan ollut tarpeeksi nopea 4. syvyydelle, vaan testaus täytyi jälleen keskeyttää kuten tavallisen minimaxin tapauksessa.

Taulukko 5. Alfa-beta -karsinnan laskenta-ajat

Syvyys	Keskiarvo	Mediaani	Keskihajonta
1	1.057s	0.732s	0.953s
2	26.232s	29.223s	18.912s
3	130.689s	50.324s	172.614s

4.3.2.3. Alfa-beta -karsinta lisäosilla

Taulukossa 5 on tilastot, kun algoritmiin lisättiin alfa-beta -karsinnan lisäksi avauskirjat, transpoositaulukot sekä iteroiva syventäminen. Koska testin tarkoituksena oli laskenta-aikojen selvittäminen, iteroivan syventämisen aikarajoitusta ei käytetty.

Taulukko 6. Alfa-beta -karsinnan laskenta-ajat lisäosilla

Syvyys	Keskiarvo	Mediaani	Keskihajonta
1	1.764s	1.666s	1.110s
2	26.877s	28.739s	23.136s
3	86.841s	60.405s	82.125s
4	634.741s	218.080s	1162.508s

5. POHDINTA

Työssä onnistuttiin luomaan shakkirobotille tekoäly, joka kykeni pelaamaan shakkia riittävän vakuuttavasti ihmispelaajaa vastaan. Tekoälyn laskenta-ajat pysyivät myös matalammilla algoritmin syvyyksillä kohtuullisen lyhyinä, ja tekoälylle suoritetuissa testeissä ihmispelaajan odottamisaika ei ollut turhauttavan suuri.

Jatkokehityksen kannalta tekoälyn parannusalueet ovat jakautuneet kahteen osioon: algoritmin pelikyvyn parantamiseen ja algoritmin laskentatehokkuuden nostamiseen. Algoritmin laskentatehon nostaminen mahdollistaa myös algoritmin syvyyden nostamisen, vaikuttaen kaksiosaisesti sekä pelaajan kokemaan odottamisaikaan siirtojen välillä, mutta myös tekoälyn tarjoamaan vastukseen. Mikäli syvyys tällöin nousisi kuitenkin erittäin suureksi, tulisi tekoälylle myös toteuttaa sovelias vaikeuden säätö heikompia pelaajia varten.

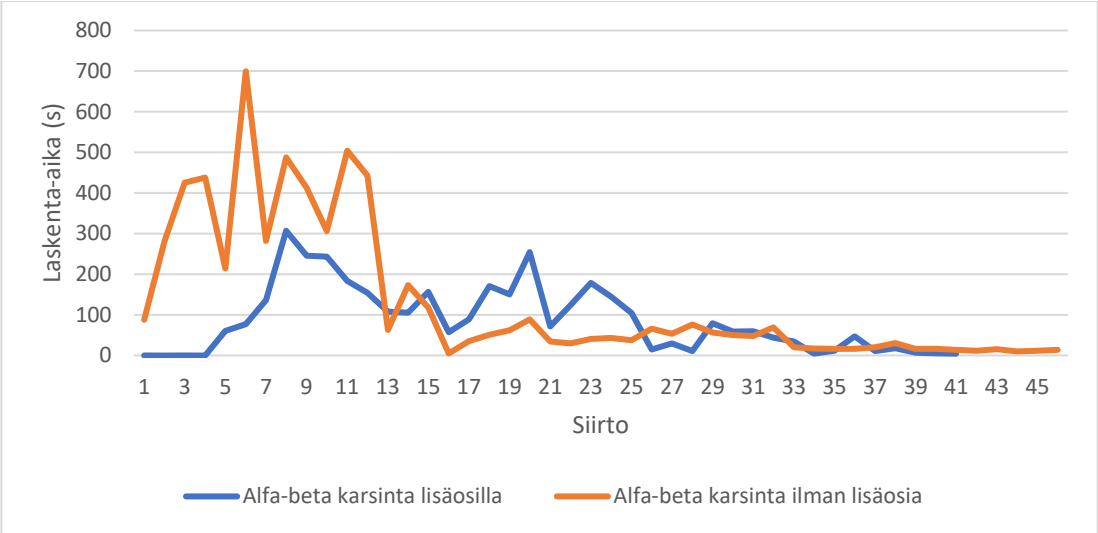
5.1. Tekoälyn suorituskyky ihmispelaajia vastaan

Testeissä ihmispelaajia vastaan tekoälyn laskenta-aika ei noussut kriittiseksi ongelmaksi, mutta tästä huolimatta huomattavassa osassa palautetta mainittiin algoritmin siirtoihin käyttämä aika tekoälyn haittapuolena. Algoritmia luodessa tähdättiin noin alle minuutin keskimääräiseen odotusaikaan, mutta palautteen perusteella tämä on kuitenkin ihmispelaajalle liikaa. Alle puolen minuutin laskenta-aika olisi jo huomattavasti siedettävämpi, mutta Raspberry Pi:n laskentateholla ja algoritmin nykyisellä kehitystasolla tämä on kuitenkin saavuttamattomissa vaarantamatta tekoälyn pelikykyä.

Kuitenkin juurikin pelikyvyllisesti algoritmi oli kuitenkin varsinkin aloittelijoille sopiva jo 3. syvyydellä lisäosia käytettäessä. Osassa testeistä saatua palautetta tekoälylle toivottiin vaikeustason säätöä, sillä nykyisessä tilassaan tekoälyn osaamistaso vaihteli huomattavasti. Vaikkakin tekoäly kykeni etenkin pelin alkupuolella toimimaan vakuuttavana vastustajana, monesti algoritmi kuitenkin loppupelissä ajoi itsensä umpikujaan kehnolla siirrolla. Tähän olisi mahdollista vaikuttaa esimerkiksi lopetuskirjaston käyttämisellä tai laskentasyvyyden nostamisella.

5.2. Avauskirjojen vaikutus tekoälyn laskenta-aikaan

Kuvassa 6 on kaavio, joka kuvaa alfa-beta -karsintaa lisäosilla ja tavallisen alfa-beta -karsinnan laskenta-aikaa siirroittain. Kaaviosta on mahdollista huomata avauskirjojen kauaskantoinen vaikutus siirtojen laskenta-aikaan. Vaikka esimerkkipelissä alfa-beta -karsinta lisäosilla valitsee vain 4 ensimmäistä siirtoa avauskirjoista, pysyy se silti huomattavasti nopeampana aina siirtoon 13 asti. Taulukoista 5 ja 6 nähdään myös tavallisen alfa-beta -karsinnan keskihajonnan olevan yli kaksinkertainen alfa-beta -karsinnan lisäosilla vastaavan arvon 3 syvyydellä. Tämä tarkoittaa, että ilman lisäosia algoritmin laskenta-aika vaihtelee merkittävästi siirtojen välillä. Kuvasta voidaankin havaita, että lisäosaton algoritmi on laskenut nopeimmat siirtonsa lähes välittömästi, mutta miettinyt hitaimpia siirtojaan yli 11 minuuttia. Lisäosilla tämä laskenta-ajan vaihtelu on ollut paljon tasaisempaa.



Kuva 6. Alfa-beta-algoritmien siirtoaikojen vertailu

5.3. Jatkokehitys

Jatkokehitys voisi keskittyä esimerkiksi laskenta-ajan alentamiseen. Laskenta-aikaa saisi nopeutettua huomattavasti toteuttamalla tekoäly käännetyllä ohjelmointikielellä kuten C++. Tekoälyn toimintaa voi myös nopeuttaa paremmalla laitteistolla tai siirtämällä algoritmien suorittamisen pilvipalveluun ja käyttämällä robotin omaa laitteistoa vain pelisiirtojen toteuttamiseen.

Yksi mielenkiintoinen jatkokehityksen osa-alue olisi myös algoritmien toteuttaminen ja vertailu käyttäen alfa-beta -karsinnan sijasta jotain muuta hakualgoritmia, kuten esimerkiksi alfa-betaa mahdollisesti tehokkaampaa NegaScout–algoritmia. Laskentatehon salliessa koneoppimisen käyttö tekoälyn muodostamisessa voisi myös tuoda uusia näkökulmia tutkimusaiheeseen.

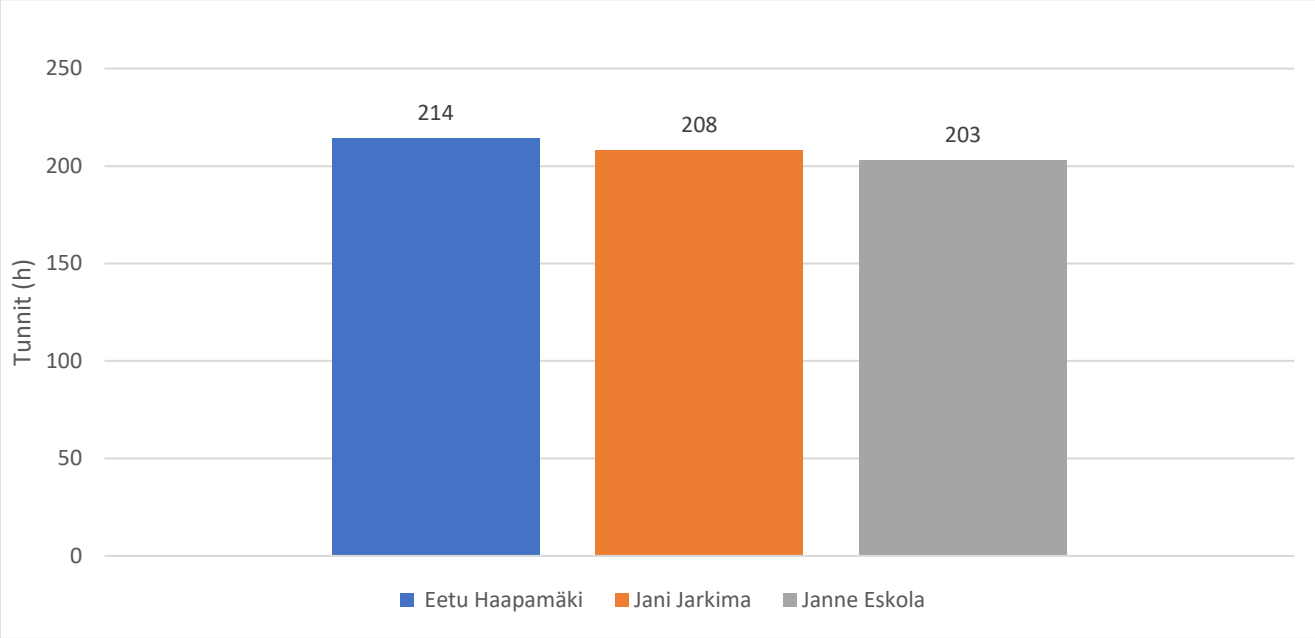
HRI:n kannalta olennaisia ominaisuuksia voisi olla esimerkiksi vaikeusasteen säätäminen tai puhesyntetisaation käyttäminen pelaajan viihdyttämiseen shakkirobotin siirtojen aikana. Pelaaja voisi joko valita itselleen sopivan vaikeusasteen ennen pelin aloittamista tai tekoäly voisi arvioida ihmispelaajan vahvuuden tämän tekemien siirtojen perusteella ja muuttaa pelitasoaan tämän mukaisesti. Shakkirobotille on myös mahdollista antaa puhesyntetisaation avulla kyky puhua. Robotti voisi esimerkiksi kommentoida ihmispelaajan pelaamista tai kertoa mielenkiintoisia faktoja shakin historiasta ja shakkiteoriasta samalla kun pelitekoäly laskee robotin seuraavaa siirtoa. Tämä voisi lisätä ihmispelaajan viihtyvyyttä sekä vähentää pelaajan turhautumista pitkien laskenta-aikojen aikana, mahdollistaen laskenta-aikojen nostamisen.

6. AJANKÄYTTÖ

Työskentely tapahtui osittain ryhmässä ja osittain itsenäisesti. Itsenäisen työn ohjaukseksi ryhmä tapasi viikoittain, jolloin käytiin läpi kunkin ryhmän jäsenen edellisviikolla tekemä työ sekä suunniteltiin työnjako seuraavaan tapaamiseen asti. Taulukossa 7 ja kuvassa 7 on esitetty ryhmän jäsenten työhön käyttämät tunnit.

Taulukko 7. Projektin ajankäyttö

Nimi	Tunnit
Eetu Haapamäki	214
Jani Jarkima	208
Janne Eskola	203



Kuva 7 Projektiin käytetyt tunnit

7. YHTEENVETO

Tietotekniikan ja robotiikan kehittyessä mahdollisuus robotiikan käyttämiseen jokapäiväisiin tarkoituksiin lisääntyy jatkuvasti. Teknologian kehittyessä myös robottien käyttöalueet laajenevat, tuoden robotit yhä oleellisemmaksi osaksi ihmisten elämää. Yksi näistä käyttöalueista on pelirobotit, jotka pelaavat esimerkiksi lautapelejä ihmisen kanssa.

Tässä työssä toteutettiin tekoäly shakkirobotille rajatun laskentatehon ympäristössä. Työssä vertailtiin alfa-beta -karsinta -algoritmin suorituskykyä tavalliseen minimax-algoritmiin. Alfa-beta-algoritmista oli vertailussa sekä tavallinen versio, että kehittyneempi versio johon oli toteutettu iteroiva syventäminen, transpoositaulukot sekä shakkiavauskirjat. Tekoälyä testattiin vaiheittain ihmispelaajia vastaan ja saatua palautetta käytettiin tekoälyn kehittämiseen sekä tutkimuskysymysten suuntaamiseen. Tekoälylle toteutettiin myös IRC- sekä Lichess-botit joiden avulla sitä vastaan pystyttiin pelaamaan myös internetin välityksellä.

Kehitetty tekoäly oli tutkimuksen erittäin rajallisellakin laskentateholla riittävän vakuuttava ihmispelaajaa vastaan ja sen laskentateho pysyi hyväksyttävissä rajoissa. Huomattiin kuitenkin, että näissä rajoissa pysyminen ei ollut riittävä ihmispelaajalle ja parannusvaraa oli yhä.

8. LÄHDELUETTELO

[1] A. Mayor, Gods and Robots: Myths, Machines, and Ancient Dreams of Technology, 2018.

[2] G. Legnani ja I. Fassi, Robotics: State of the Art and Future Trends, 2012, p. 10.

[3] O. Kinnander, ”Rise of the Lawn-cutting Machines,” Bloomberg, 26 10 2012. [Online]. Available: https://www.bloomberg.com/news/articles/2012-10-25/rise-of-the-lawn-cutting-machines. [Haettu 14 5 2019].

[4] J. Selkäinaho, Adaptive autonomous navigation of mobile robots in unknown environments, 2002.

[5] E. Ackerman, ”Festo's Newest Robot Is a Hopping Bionic Kangaroo,” IEEE Spectrum, 2 4 2014. [Online]. Available: https://spectrum.ieee.org/automaton/robotics/robotics-hardware/festo-newest-robot-is-a-hopping-bionic-kangaroo. [Haettu 14 5 2019].

[6] M. Elshaw, G. Palm ja S. Wermter, Biomimetic Neural Learning for Intelligent Robots: Intelligent Systems, Cognitive Robotics, and Neuroscience, 2005.

[7] A. Bauer, ”World Robotics 2018,” International Federation of Robotics, 2018. [Online]. Available: https://ifr.org/downloads/press2018/Foreword_WR_2018_Industrial_Robots.pdf. [Haettu 14 5 2019].

[8] T. B. Sheridan, ”Human–Robot Interaction: Status and Challenges.,” *The Journal of the Human Factors and Ergonomics Society,* osa/vuosik. 58, nro 4, pp. 525-532, 2016.

[9] N. Nikolakis, V. Maratos ja S. Makris, ”A cyber physical system (CPS) approach for safe human-robot collaboration in a shared workplace,” *Robotics and Computer-Integrated Manufacturing,* osa/vuosik. 56, pp. 233-243, 2019.

[10] A. Levratti, A. De Vuono, C. Fantuzzi ja C. Secchi, ”TIREBOT: A novel tire workshop assistant robot,” tekijä: *2016 IEEE International Conference on Advanced Intelligent Mechatronics (AIM)*, Banff, 2016.

[11] F. Othman, M. A. K. Bahrin, N. H. N. Azli ja M. F. Talib, ”Industry 4.0: A review on industrial automation and robotic,” *Jurnal Teknologi,* osa/vuosik. 78, nro 6-13, pp. 137-143, 2016.

[12] J. Bloem, M. van Doom, S. Duivestein, D. Excoffier, R. Maas ja E. van Ommeren, ”VINT research report 3: The Fourth Industrial Revolution,” 2014.

[13] T.-H. S. Li ja S.-J. Chang, ”Autonomous fuzzy parking control of a car-like mobile robot,” *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Human,* osa/vuosik. 33, nro 4, pp. 451-465, 2003.

[14] S. Ullah, Z. Mumtaz, S. Liu, M. Abubaqr, A. Mahboob ja H. A. Madni, ”An Automated Robot-Car Control System with Hand-Gestures and Mobile Application Using Arduino,” Preprints, 2019.

[15] A. Paolillo, P. Gergondet, A. Cherubini, M. Vendittelli ja A. Kheddar, ”Autonomous car driving by a humanoid robot,” *Journal of Field Robotics,* osa/vuosik. 35, pp. 169-186, 2018.

[16] J. Zhou, X. Ma, Z. Xu ja Z. Qi, ”Overview of Medical Robot Technology Development,” tekijä: *37th Chinese Control Conference*, 2018.

[17] G.-Z. Yang, J. Cambias, K. Cleary, E. Daimler, J. Drake, P. E. Dupont, N. Hata, P. Kazanzides, S. Martel, R. V. Patel, V. J. Santos ja R. H. Taylor, ”Medical robotics—Regulatory, ethical, and legal considerations for increasing levels of autonomy,” *Science Robotics,* osa/vuosik. 2, nro 4, 2017.

[18] M. Barnes ja F. Jentsch, Human-Robot Interactions in Future Military Operations (Human Factors in Defence), 2010.

[19] *European Parliament resolution on autonomous weapon systems,* 2018.

[20] T. Fond, I. Nourbakhsh ja K. Deutenkahn, ”A Survey of Socially Interactive Robots,” *Robotics and Autonomous Systems,* osa/vuosik. 42, nro 3-4, pp. 143-166, 2003.

[21] R. Pinillos, S. Marcos, R. Feliz, E. Zalama ja J. Gómez-García-Bermejo, ”Long-term assessment of a service robot in a hotel environment,” *Robotics and Autonomous Systems,* osa/vuosik. 79, pp. 40-57, 2016.

[22] A. Agah, J.-J. Cabibihan, A. M. Howard ja M. A. H. H. Salichs, ”Social Robotics: 8th International Conference,” 2016.

[23] G. Larregay, F. Pinna, L. Avila ja D. Morán, ”Design and Implementation of a Computer Vision System for an Autonomous Chess-Playing Robot,” *Journal of Computer Science and Technology, 18(01), e01,* 25 April 2018.

[24] J. Jiao, C. Huang, H. Liu ja G. Zhang, ”A Chinese Chessboard Calibration Method in Chess-Playing Robot by Machine Vision Sensing,” tekijä: *Journal of Physics: Conference Series, Volume 1026, conference 1*, Chongqing, 2018.

[25] H. M. Luqman ja M. Zaffar, ”Chess Brain and Autonomous Chess Playing Robotic System,” tekijä: *2016 International Conference on Autonomous Robot Systems and Competitions (ICARSC)*, Bracanga, 2016.

[26] D. Lukač, ”Playing chess with the assistance of an industrial robot,” tekijä: *2018 3rd International Conference on Control and Robotics Engineering (ICCRE)*, Nagoya, 2018.

[27] A. Kaplan ja M. Haenlein, ”Siri, Siri, in my hand: Who’s the fairest in the land? On the interpretations, illustrations, and implications of artificial intelligence,” *Business Horizons,* osa/vuosik. 62, nro 1, pp. 15-25, 2019.

[28] H. Lu, Y. Li, M. Chen ja H. Kim, ”Brain Intelligence: Go beyond Artificial Intelligence,” *Mobile Networks and Applications,* osa/vuosik. 23, nro 2, pp. 368-375, 2018.

[29] W. Ertel, Introduction to Artificial Intelligence (2. painos)., 2017.

[30] E. Alpaydin, Introduction to Machine Learning., 2010.

[31] P. Lison, An introduction to machine learning., 2012.

[32] M. Campbell, ”Mastering Board Games.,” *Science,* osa/vuosik. 362, nro 6419, p. 1118, 2018.

[33] G. Bonaccorso, Machine Learning Algorithms, 2017.

[34] M. Nielsen, Neural Networks and Deep Learning, 2018.

[35] R. O. M. Team, ”Applying Artificial Intelligence and Machine Learning in Robotics,” [Online]. Available: https://www.robotics.org/blog-

article.cfm/Applying-Artificial-Intelligence-and-Machine-Learning-in-Robotics/103. [Haettu 13 4 2019].

[36] R. Jain, R. Karturi ja B. G. Schunck, Machine Vision, 1995.

[37] C. Shannon, ”Programming a Computer for Playing Chess,” *Philosophical Magazine,* osa/vuosik. 41, pp. 256-275, 1950.

[38] ”Mass, Size and Density of the Universe,” National Solar Observatory, [Online]. Available: https://people.cs.umass.edu/~immerman/stanford/universe.html. [Haettu 14 5 2019].

[39] R. Djurhuus, ”Chess Algorithms: Theory and Practice,” 3 10 2012. [Online]. Available: https://www.uio.no/studier/emner/matnat/ifi/INF4130/h12/undervisningsmateriale/chess-algorithms-theory-and-practice-ver2012.pdf. [Haettu 14 5 2019].

[40] B. J. Copeland, The Essential Turing: Seminal Writings in Computing, Logic, Philosophy, Artificial Intelligence, and Artificial Life: Plus The Secrets of Enigma., 2004.

[41] High Tech History, ”A history of computer chess – from the “Mechanical Turk” to “Deep Blue”,” 2011. [Online]. Available: https://hightechhistory.wordpress.com/2011/04/21/a-history-of-computer-chess-–-from-the-mechanical-turk-to-“deep-blue”/. [Haettu 13 4 2019].

[42] A. Kotok, ”A Chess Playing Program for the IBM 7090 Computer,” 1962.

[43] G. Kasparov, ”The Chess Master and the Computer,” tekijä: *Chess Metaphors: Artificial Intelligence and the Human Mind (The MIT Press)* , 2010, p. 205.

[44] M. Campbell, A. J. J. Hoane ja F.-h. Hsu, ”Deep Blue,” *Artificial Intelligence,* Osa %1/%21-2, nro 143, pp. 57-83, 2002.

[45] M. E. Glickman ja A. C. Jones, ”Rating the Chess Rating System,” 1999.

[46] C. (. C. R. lists). [Online]. Available: http://ccrl.chessdom.com/ccrl/. [Haettu 17 4 2019].

[47] F. (. I. d. Échecs). [Online]. Available: https://ratings.fide.com/top.phtml?list=men. [Haettu 17 4 2019].

[48] K. Arulkumaran, A. Cully ja J. Togelius, ”AlphaStar: An Evolutionary Computation Perspective,” 2019.

[49] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. Lillicrap, K. Simonyan ja D. Hassabis, ”Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm,” 2017.

[50] L. Kaufman, ”The Evaluation of Material Imbalances,” *Chess Life,* Maaliskuu 1999.

[51] R. Greenblatt, D. Eastlake ja S. Crocker, ”The Greenblatt chess program,” tekijä: *AFIPS '67 (Fall) Proceedings of the November 14-16, 1967, fall joint computer conference*, Anaheim, 1967.

[52] T. Marsland, ”Computer Chess and Search,” tekijä: *Encyclopedia of Artificial Intelligence*, 2 toim., S. Shapiro, Toim., 1992.

[53] R. Shams, H. Kaindl ja H. Horacek, ”Using Aspiration Windows for Minimax Algorithms,” tekijä: *Twelfth International Joint Conference on Artificial Intelligence*, Sydney, 1991.

[54] B. Huberman, A Program to Play Chess End Games, 1968.

[55] J. Schaeffer, N. Burch, Y. Björnsson, A. Kishimoto, M. Müller, R. Lake, P. Lu ja S. Sutphen, ”Checkers Is Solved,” *Science,* nro 317, pp. 1518-1522, 2007.

[56] G. Kasparov, ”Chess, a Drosophila of reasoning.,” *Science,* osa/vuosik. 362, nro 6419, p. 1087, 2018.