
Jukka Kotilainen

# Hakumenetelmien vertailua myllypelissä

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tieto- ja viestintätekniikka

Insinöörityö

5.5.2022

# Tiivistelmä


| | |
|---|---|
| Tekijä: | Jukka Kotilainen |
| Otsikko: | Hakumenetelmien vertailua myllypelissä |
| Sivumäärä: | 41 sivua |
| Aika: | 5.5.2022 |
| Tutkinto: | Insinööri (AMK) |
| Tutkinto-ohjelma: | Tieto- ja viestintätekniikka |
| Ammatillinen pääaine: | Ohjelmistotuotanto |
| Ohjaajat: | Lehtori Miikka Mäki-Uuro<br>Lehtori Antti laiho |


---


Insinöörityössä vertailtiin hakumenetelmien eroavaisuuksia sekä niiden soveltuvuutta myllyssä tai muissa saman tyylisissä peleissä. Vertailtavat hakumenetelmät olivat minimax, iteratiivisesti syventyvä minimax ja Monte Carlo -puuhaku. Hakumenetelmät valittiin niiden erilaisten ominaisuuksien perusteella, eli millaisia ominaisuuksia ne mahdollistavat tehtyyn sovellukseen. Työssä käytiin myös läpi hakumenetelmien optimointia sekä menetelmien implementoinnissa esiintyneet ongelmat ja niiden ratkaisut.

Eri hakumenetelmät tarjoavat samaan ongelmaan erilaisen ratkaisun ja mahdollistavat haluttuja sivuominaisuuksia toteutettuun sovellukseen. Tästä syystä onkin siis hyvä pohtia, mitä hakumenetelmää kannattaa milloinkin käyttää ja mitä muita ominaisuuksia ne mahdollistavat.

Hakumenetelmien toiminnan yhteydessä kerrotaan pelipuun ratkomisesta. Samanlaista puurakenteiden ratkomista on myös muualla ohjelmoinnissa, eli menetelmiä voidaan myös soveltaa muuallakin.

Insinöörityössä kehitettiin myllysovellus, jossa valitut hakumenetelmät ja niiden vertailuun tarvittavat työkalut toteutettiin. Käyttöliittymälle kehitettiin myös palvelinpuolen sovellus, jolle kerätty data lähetetään ja haetaan. Palvelimelta voidaan suoraan esittää kerätty data erilaisten kuvaajien avulla, mitkä auttavat hakumenetelmiä kehittäessä, vertaillessa ja etenkin optimoidessa.

Työn tarkoituksena ei siis ollut toteuttaa mahdollisimman hyvää myllynpelausalgoritmia, vaan vertailla hakumenetelmien soveltuvuutta ja niiden ominaisuuksia keskenään. Siitä huolimatta työssä toteutettiin myllyä melko hyvin eri vaikeustasoilla pelaava algoritmi.



Avainsanat: hakumenetelmät, mylly, minimax, monte carlo -puuhaku, alfa-beta

# Abstract


| | |
|---|---|
| Author: | Jukka Kotilainen |
| Title: | Comparison of Search Methods in Nine-Men's Morris |
| Number of Pages: | 41 pages |
| Date: | 5 May 2022 |
| Degree: | Bachelor of Engineering |
| Degree Programme: | Information and Communication Technology |
| Professional Major: | Software Engineering |
| Supervisors: | Miikka Mäki-Uuro, Senior Lecturer<br>Antti Laiho, Senior Lecturer |


---


The focus of the study was to compare the differences in search methods and their suitability in nine-men's morris or other same type of games. The compared search methods were minimax, iterative deepening minimax and Monte Carlo tree search. The search methods were chosen on the basis that they would also make new features possible in the game application. The thesis also reviews the optimization of search methods and the problems observed in the implementation of them, as well as the solutions to the problems.

Different search methods offer a different solution to the same problem and enable new features to be added to the game application. Because of this it is beneficial to consider which search method to use and what features they enable.

The thesis also covers game trees and how to solve them. Similar solutions for tree structures are also used elsewhere in computer science, so the methods can be applied elsewhere, too.

During the study a game application was developed where all the search methods and the tools needed for comparing them were implemented. A server-side application was also developed where the collected data is sent to and received from. The data on the server can be visualized and analyzed with the help of different kinds of graphs, which help in comparison and the development of search methods and especially in their optimization.

The point of the study was not to implement the best possible algorithm for nine-men's morris, but to compare the suitability of the search methods and their properties. Nevertheless, an algorithm that plays it fairly well at different difficulty levels was developed.



Keywords: search methods, nine-men's morris, minimax, monte carlo tree search, alfa-beta

## Sisällys

# 1 Johdanto

Insinöörityössä vertaillaan erilaisten hakumenetelmien toimintaa ja soveltuvuutta myllyssä tai muissa samantyylisissä peleissä. Eri hakumenetelmät antavat samaan ongelmaan erilaisen ratkaisun ja samalla muita mahdollisia hyödyllisiä ominaisuuksia. Tästä syystä niihin halutaan tutustua ja tutkia, millaisiin ongelmiin eri hakumenetelmät soveltuvat parhaiten ja millaisia muita ominaisuuksia niillä saattaa olla.

Insinöörityö toteutetaan harrastus myllysovelluksen pohjalta, mihin toteutetaan eri hakumenetelmät ja niiden vertailuun tarvittavat työkalut. Ennen insinöörityön aloittamista sovelluksessa oli toteutettu vain toimiva pohja myllyn pelaamiseen ilman tekoälyvastustajaa.

Aiheessa erityisen kiinnostavaa on tutustua erilaisiin hakumenetelmien optimointitapoihin, eli tapoihin, jolla niiden toimintaa voidaan nopeuttaa ja tehdä tehokkaammaksi. Optimointitavat ovat yleispäteviä muuallakin ohjelmoinnissa, joten niitä voidaan käyttää muuallakin kuin vain hakumenetelmien parissa.

Työn tavoitteena on tutkia eri hakumenetelmien eroavaisuuksia ja niiden soveltuvuutta myllyssä tai muissa samantyylisissä peleissä. Tavoitteena ei kuitenkaan ole toteuttaa mahdollisimman hyvää algoritmia pelaamaan myllyä, vaan nimenomaan vertailla hakumenetelmien toimintaa ja soveltuvuutta keskenään. Vaikka vertailua tapahtuu ainoastaan myllyssä, niin saatua tietoa voidaan myös soveltaa muihin samantyylisiin peleihin. Pohjimmiltaan kyseessä on aina samantyylinen ongelma, eli pelipuun ratkominen.

Insinöörityön päätteeksi sovelluksella tulisi pystyä pelaamaan tietokonetta vastaan eri hakumenetelmin ja keräämään tarvittavaa dataa pelin aikana. Kerättyä dataa tulisi voida esittää erilaisten kuvaajien avulla. Kuvaajien avulla voidaan havainnollistaa hakumenetelmien toimintaa ja toimivuutta.

Luvussa 2 esitellään myllypelin säännöt, taktiikoista, sekä hieman sen historiasta. Luvuissa 3–5 syvennytään tutkimiini hakumenetelmiin sekä niiden optimointitapoihin. Luvussa 6 kerrotaan insinöörityötä varten toteutetuista käyttöliittymä- sekä palvelinpuolen sovelluksista. Luvussa 7 on hakumenetelmien vertailua keskenään ja lopuksi luvussa 8 on yhteenveto insinöörityön tuloksista ja ajatuksista.

# 2 Mylly lautapelinä

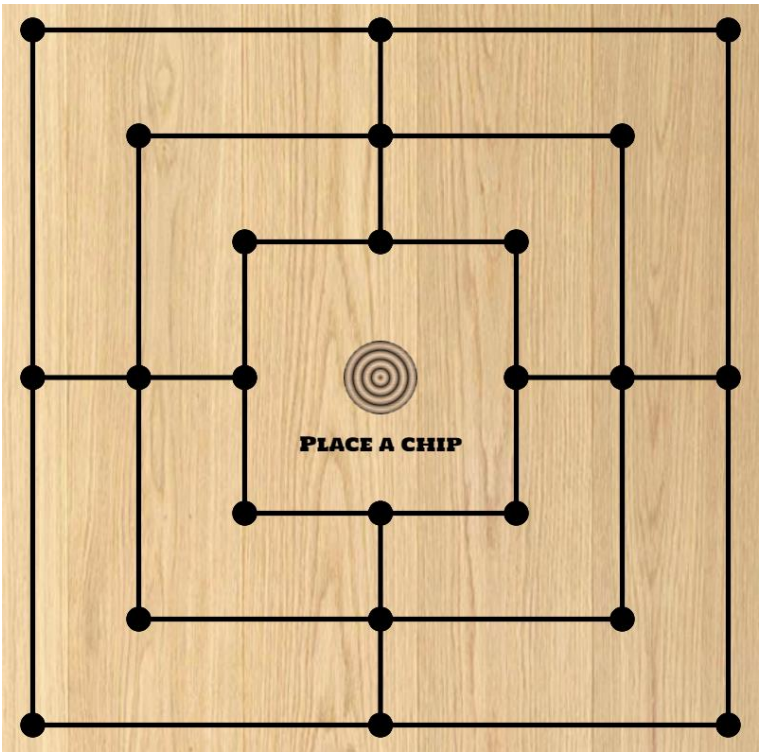
Seuraavissa aliluvuissa kerrotaan myllyn historiasta ja sen säännöistä. Sääntöjä ja pelin kulkua havainnollistetaan kuvilla myllysovelluksesta, joka tehtiin insinöörityötä varten. Tästä sovelluksesta kerrotaan tarkemmin luvussa 6. Perinteisesti myllypeliä on pelattu kuitenkin esimerkiksi paperille piirretyllä pelilaudalla ja omatekoisilla nappuloilla tai valmiilla puisella pelilaudalla ja nappuloilla.

## 2.1 Myllyn historiaa

Myllypelin tarkkaa alkuperää ei tiedetä, mutta sen uskotaan olevan yksi vanhimmista peleistä kautta aikojen. Peliä uskotaan pelanneen mm. veistäjät samalla, kun he ovat rakentaneet isoja temppeleitä, sillä kaiverruksia pelilaudasta on löydetty paljon eri rakennuksista. Vanhin löydetty myllyn pelilauta on löydetty kaiverrettuna egyptiläisen temppelin kattolaattoihin. Temppelin rakennusvuodeksi on arvioutu noin 1400 eaa. [1], mutta ei ole varmaa, onko pelilauta ollut kaiverrettuna laattaan jo ennen rakennustöitä vai oliko se kaiverrettu siihen rakennustöiden aikoihin. Pelin suosio oli huipussaan keskiajalla Englannissa, josta on löydetty pelilautoja vanhoista luostareista ja vanhojen teiden varsilta.

## 2.2 Myllyn säännöt

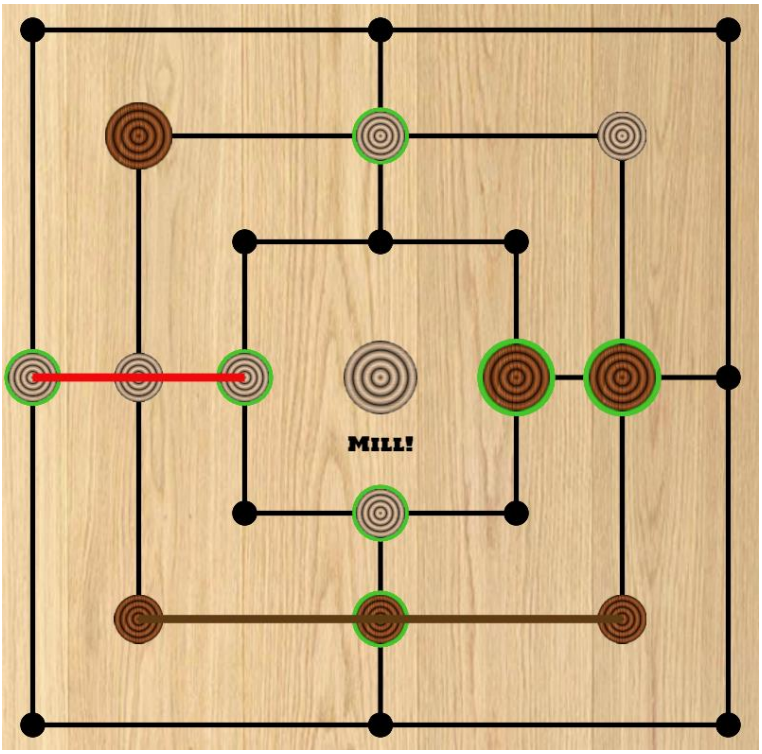
Myllyn pelilauta muodostuu kolmesta sisäkkäin olevista neliöistä, joiden välille muodostuu 24 pistettä. Pelaajalla on 9 pelinappulaa, jotka asetetaan pelilaudan pisteisiin. Kuvassa 1 on esimerkki tyhjästä myllyn pelilaudasta.




Kuva 1. Tyhjä myllyn pelilauta.

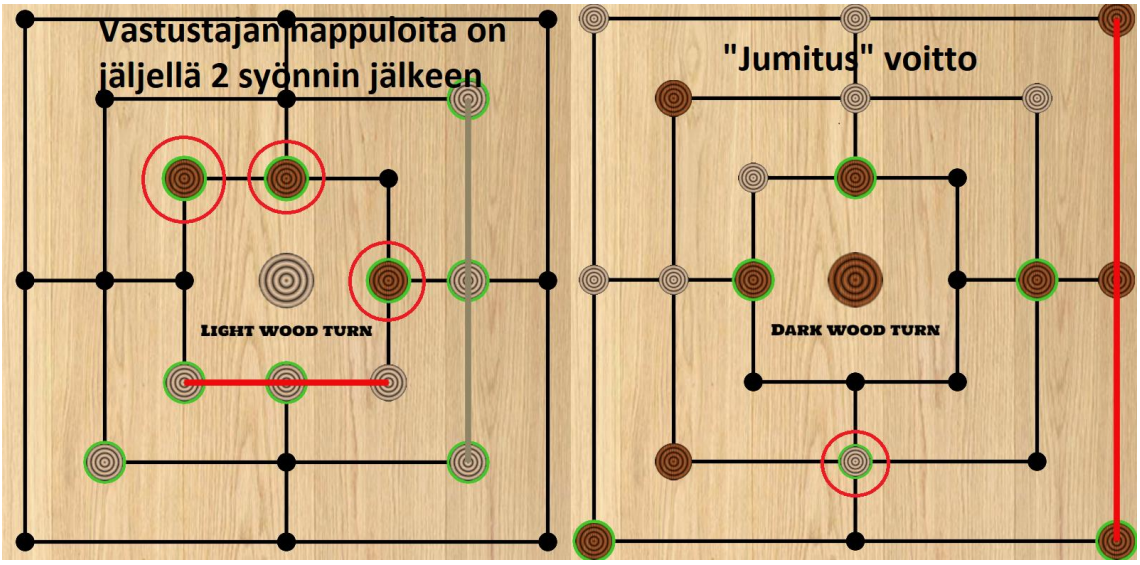
Pelin ideana on muodostaa ”myllyjä”, jolloin pelaaja voi poistaa yhden vastustajan pelinappulan laudalta. Mylly muodostuu siten, että pelaajalla on kolme omaa nappulaa vierekkäin samalla suoralla. Kuvassa 2 on kaksi esimerkkiä myllyn muodostumisesta, joista toinen on merkitty punaisella viivalla sen päällä, mikä tarkoittaa, että kyseisen mylly on juuri muodostettu ja pelaaja

on valitsemassa syötävää nappulaa. Toisessa tapauksessa mylly on merkattu pelaajan värisellä viivalla.




Kuva 2 Vaalea pelaaja on saanut myllyn ja valitsee vastustajan napeista syötävän napin. Vaalea pelaaja ei voi syödä vastustajan myllystä, joten sen on valittava syötävä nappula vastustajan kolmesta muusta nappulasta.

Peli päättyy, kun pelaaja onnistuu vähentämään vastustajan nappulat kahteen tai onnistuu ”jumittamaan” vastustajan nappulat siten, että vastustajalla ei ole enää jäljellä mahdollisia siirtoja. Kuvassa 3 ovat esimerkit voittotilanteista. Peli on myös mahdollista päättyä tasapeliin [2]. Tasapeli tapahtuu esimerkiksi silloin, jos molemmat pelaajat siirtävät nappuloitaan edestakaisin, eikä peli enää etene.




Kuva 3 Esimerkkivoittotilanteet. Vasemmassa pelissä vaalea pelaaja voittaa, kun se syö yhden vastustajan napeista. Oikeassa tapauksessa tumma pelaaja voittaa, kun se syö vaalean pelaajan ainoan liikutettavan nappulan.

Peli edistyy kolmessa eri vaiheessa [3]:

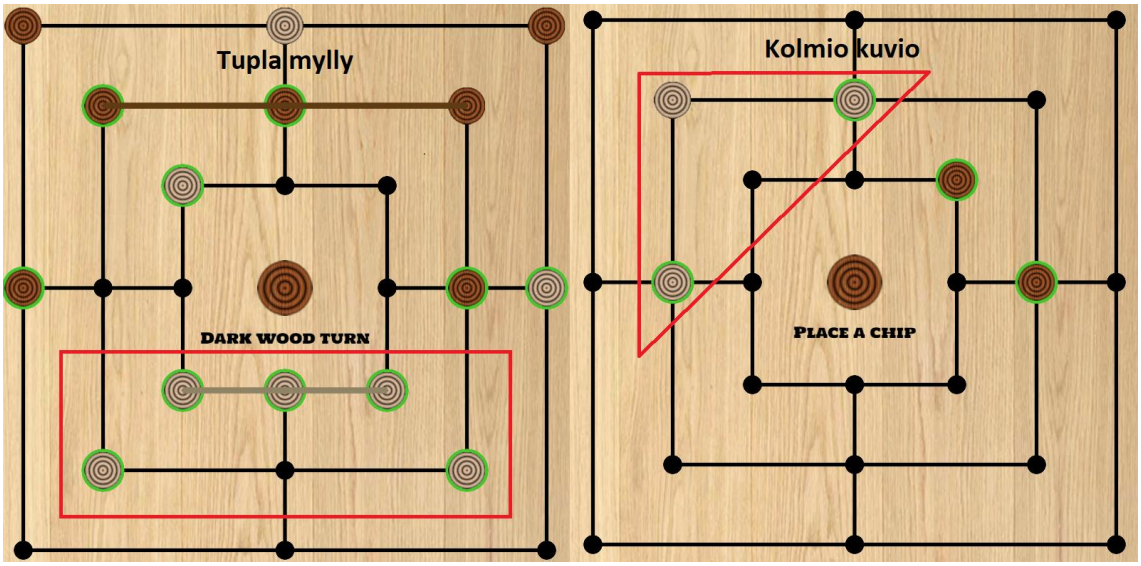
1. nappuloiden asettaminen
2. nappuloiden liikuttaminen vierekkäisiin pisteisiin
3. nappuloiden ”lennättäminen”.

Ensimmäisessä vaiheessa pelaajat asettavat pelinappulansa vuorotellen laudalle. Pelaajan tavoitteena on muodostaa myllyjä, jolloin hän voi poistaa yhden vastustajan nappuloista. Pelaaja voi poistaa vastustajan minkä tahansa nappulan, ellei se ole osa vastustajan omaa myllyä. Jos vastustajan kaikki nappulat ovat osa myllyä, saa pelaaja syödä minkä tahansa vastustajan nappulan. Ensimmäisessä vaiheessa myllyjen muodostamista tärkeämpää on kuitenkin asettaa pelinappulat huolellisesti seuraavia vaiheita varten. Tarkoituksena on saada pelinappulat mahdollisimman tasaisesti joka puolelle pelilautaa, koska jos nappulat asetetaan yhteen kasaan, niin vastustajan on helpompi ”jumittaa” ne. Pelin ensimmäinen vaihe päättyy, kun molemmat pelaajat ovat asettaneet kaikki 9 pelinappulaansa laudalle.

Toisessa vaiheessa pelaajat siirtävät nappuloitaan vuorotellen vierekkäisiin tyhjiin pisteisiin. Tavoitteena on muodostaa myllyjä ja vähentää vastustajan nappuloita tai saada vastustajan nappulat ”jumiin”, jolloin vastustaja ei voi enää liikkua ja pelaaja voittaa pelin. Pelaajan yleinen taktiikka on ”avata” mylly siirtämällä yhtä myllyn nappulaa tyhjään paikkaan, jolloin hän voi seuraavalla vuorollaan muodostaa myllyn uudelleen. Pelaaja siirtyy kolmannelle vaiheelle, kun hänellä on vain kolme pelinappulaa jäljellä.

Kolmannessa vaiheessa pelaaja voi ”lentää”. Lentäminen tarkoittaa, että pelaajan ei tarvitse siirtää nappuloitaan vain vierekkäisiin tyhjiin pisteisiin, vaan hän voi siirtää nappuloita mihin tahansa tyhjään pisteeseen pelilaudalla. Lentäminen mahdollistaa pelaajan hyvää puolustautumista. Pelaajien on hyvä miettiä tarkkaan, milloin on järkevää antaa vastustajan lentää. Yksi taktiikka on varmistaa voitto niin, että pelaajalla on kaksi myllyä auki vastustajan siirtyessä kolmannelle vaiheelle. Tällöin vastustaja ei voi estää molempia myllyjä, ja täten häviää pelin. Toinen yleinen taktiikka on välttää vastustajan pääseminen kolmanteen vaiheeseen kokonaan ”jumittamalla” vastustajan nappulat. Tämä taktiikka toimii erityisen hyvin, kun pelaajalla on paljon enemmän nappuloita jäljellä kuin vastustajalla.

Muita myllyn pelitaktiikoita ovat mm. tuplamylly ja kolmiokuvio [4]. Pelaajan on mahdollista muodostaa ns. ”tuplamylly”, jolloin pelaaja saa luotua uuden myllyn joka kierroksella siirtämällä yhtä nappulaa myllystä toiseen. Pelin ensimmäisellä ja kolmannella vaiheella on mahdollista muodostaa kolmiokuvio. Tällä kuviolla varmistetaan myllyn muodostumisen seuraavalla vuorolla, sillä vastustaja ei voi estää molempia myllymahdollisuuksia. Kuva 4 havainnollistaa näitä taktiikoita.




Kuva 4 Esimerkkitaktiikoita. Vasemmalla on tuplamylly ja oikealla on kolmiokuvio.

Myllystä on myös muita variaatioita, mutta kyseinen (Nine men’s morris) on niistä tunnetuin [5]. Variaatioissa pisteitä on eri määrä, pisteet on yhdistetty eri tavalla ja pelinappuloita on eri määrä.

# 3 Minimax-algoritmi

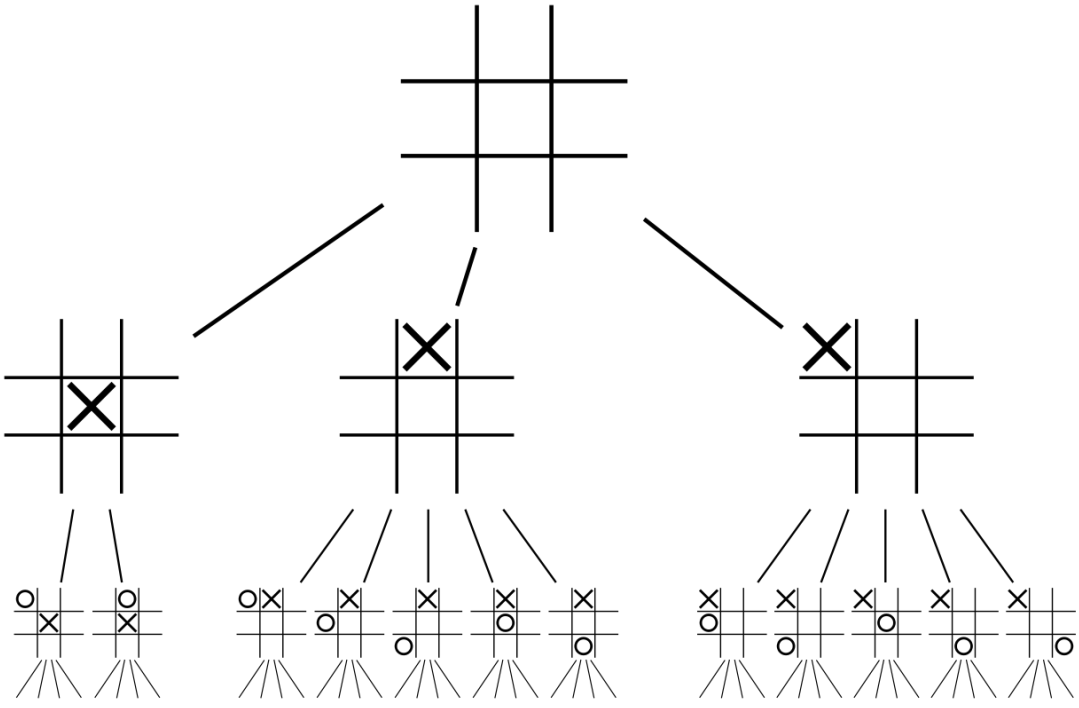
Seuraavissa aliluvuissa kerrotaan minimax-algoritmista ja pelipuista sekä niihin liittyvistä käsitteistä. Pelipuu on hyvin keskeinen osa hakumenetelmiä käsitellessä ja havainnollistan algoritmin toimintaa esimerkkipelipuukuvilla. Aliluvuissa kerrotaan myös tarkemmin minimaxin toiminnasta ja algoritmiin liittyvistä käsitteistä, kuten min- ja max-pelaajat. Luvussa 3.3 kerrotaan algoritmin optimointitavoista sekä siitä, miten nämä optimointitavat toteutettiin tehtyyn sovellukseen.

## 3.1 Minimaxin yleiskuvaus

Minimax-algoritmia käytetään usein vuoropohjaisissa kaksinpeleissä. Minimax-algoritmin tavoitteena on löytää pelaajalle optimaalinen siirto olettaen, että myös vastustaja pelaa optimaalisia siirtoja. Minimax on rekursiivinen algoritmi, eli

algoritmi noudattaa pelipuuta käydessään tiettyjä ohjeita, kunnes päädytään ratkaisuun.

Pelipuu tarkoittaa pelin kaikkien mahdollisten pelitilanteiden kuvaamista puurakenteena. Puun juurisolmu on pelin nykytilanne, ja sen lapsisolmut ovat vuorossa olevan pelaajan siirron jälkeiset pelitilanteet. Näiden lapsisolmut ovat taas vastustajapelaajan siirrot ja niin edelleen. Pelipuuta kasvatetaan ennalta määrättyyn puun syvyyteen asti, tai kunnes päädytään jommankumman pelaajan voittoon. Kuva 5 havainnollistaa pelipuun muodostumista ristinollapelissä. Kuvasta voidaan myös havaita, että pelilaudan symmetrisyyden vuoksi aluksi tarvitaan tutkia vain kolme eri siirtoa, sillä muut siirrot ovat symmetrisesti samat näiden kolmen siirtojen kanssa.



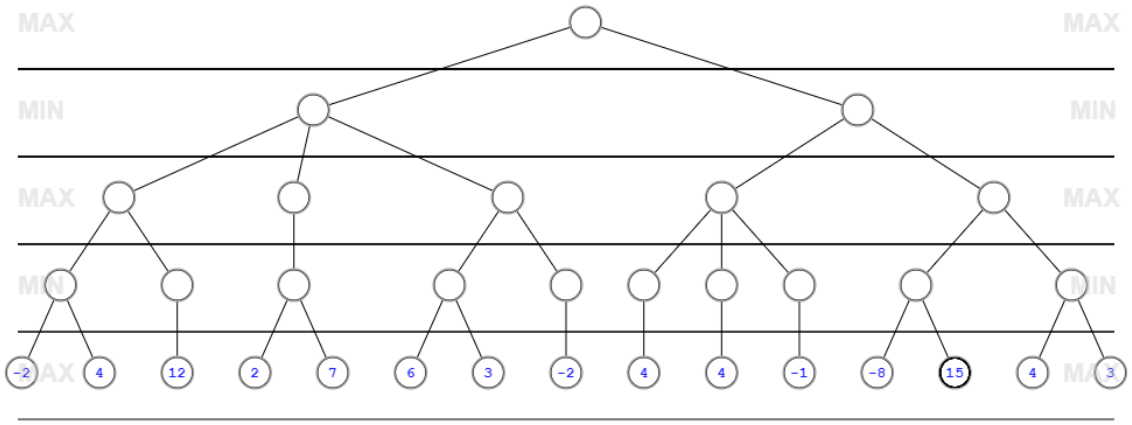
Kuva 5 Esimerkkipelipuu ristinollapelistä.

Pelin kahta pelaajaa kutsutaan min-pelaajaksi ja max-pelaajaksi, joista algoritmin nimi muodostuu. Pelin vuorossa oleva pelaaja on max-pelaaja ja hänen vastustajansa on min-pelaaja. Max-pelaajan tavoitteena on valita mahdollisimman iso pistemäärä ja vastaavasti min-pelaajan tavoitteena on

valita mahdollisimman pieni pistemäärä. Algoritmi käy siis kaikki siirtojen kombinaatiot läpi ja valitsee niistä sen siirron, mikä tuottaa parhaimman lopputuloksen max-pelaajalle.

Pelitilanteelle määritetään pistemäärä kuvaamaan, kuinka hyvä kyseinen pelitilanne on. Yksinkertaisimmillaan pisteytys on -1 (pelaaja häviää) ja 1 (pelaaja voittaa). Kyseinen pisteytys toimii vain, jos pelipuu generoidaan aina kokonaan. Pelipuuta ei kuitenkaan voida läheskään aina generoida kokonaan, sillä pelipuut kasvavat hyvin nopeasti pelin edetessä, ja puun läpikäyminen on raskas prosessi.

Pelipuu on rakenteeltaan perinteinen solmuista koostuva puurakenne. Solmuille lasketaan heuristinen arvo kyseisestä pelilaudan tilasta max-pelaajan näkökulmasta. Solmuilla voi olla lapsisolmuja ja vanhempi solmu. Kuvasta 6 nähdään, että solmulla voi olla erimäärä lapsisolmuja, joskus ei ollenkaan ja joskus useampi. Todellisuudessa myllyn pelipuussa solmuilla voi olla jopa yli 20 lapsisolmua, eli pelipuut ovat paljon suurempia kuin kuvassa oleva puu. Niitä solmuja, joilla ei ole lapsisolmuja kutsutaan lehtisolmuiksi. Puun joka toisella syvyydellä on min-pelaajan siirtovuoro ja joka toisella max-pelaajan. Puun juurella, eli syvyydessä 0 on aina max-pelaajan siirtovuoro.




Kuva 6 Esimerkkipelipuu.

Pelipuu kasvaa nopeasti mahdollisten siirtojen lukumäärän ollessa iso. Puun kasvunopeus, eli haarautumiskerroin kuvastaa, kuinka monta lapsisolmua puun solmuilla keskimäärin on. Haarautumiskerroin on hyvä indikaattori kuvaamaan,

kuinka nopeasti pelipuu kasvaa, ja sitä käytetään kuvaamaan pelien kompleksisuutta. Myllyn haarautumiskerroin on noin 10. [6.]

Myllyn alkaessa aloittavalla pelaajalla on 24 eri siirtomahdollisuutta ja seuraavalla pelaajalla on 23 jne. Tämä tarkoittaa, että pelipuun solmujen eli pelitilanteiden lukumäärä syvyydessä 2 on 24 x 23, eli 552. Syvyydessä 4 solmuja on 24 x 23 x 22 x 21, eli noin 255 000. Syvyydessä 6 solmuja on jo melkein 100 miljoonaa. Nämä arviot antavat vain suuntaa, sillä niissä ei ole otettu huomioon mahdollisten myllyjen muodostumista. Kuitenkin tästä syystä pelipuuta ei läheskään aina voida muodostaa kokonaan, vaan on valittava jokin hakusyvyys, johon asti pelipuu muodostetaan.

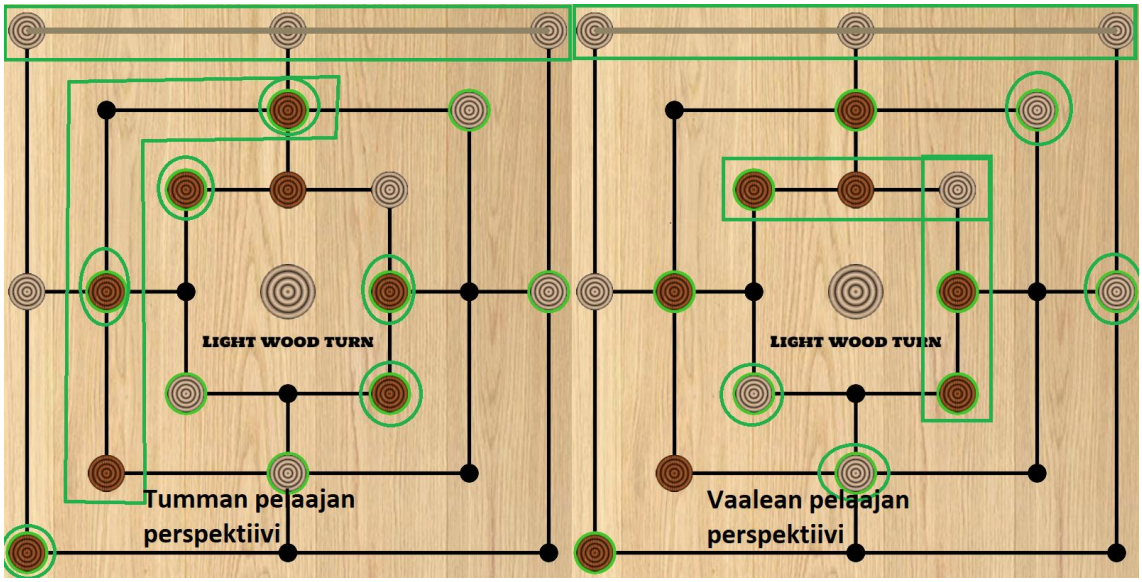
Perinteisesti minimax-algoritmi implementoidaan rekursiivisesti, minkä ansiosta pelipuu ei ole välimuistissa kokonaisuudessaan, vaan muistissa on vain sillä hetkellä käytävä puun haara. Tämä helpottaa myös algoritmin mahdollista säikeistämistä.

Pelipuuta ei läheskään aina tutkita kokonaan, vaan määritetään, kuinka syvälle, eli kuinka monen siirron päähän pelipuu tutkitaan. Pisteytystä on tästä syystä muutettava siten, että pelilaudalle voidaan laskea pistemäärä muulloinkin kuin vain pelin päättyessä. Pelilaudan pisteytystä kutsutaan evaluoinniksi ja pisteytys suoritetaan evaluointifunktiolla. Yksinkertaisimmillaan evaluointifunktio on esimerkiksi pelaajien pelinappuloiden lukumäärien erotus keskenään. Min-pelaajan nappuloiden lukumäärä vähennetään max-pelaajan nappuloiden määrästä. Samalla määritetään, että häviön pistemäärä on -∞ ja voiton ∞ pistettä.

Käytännössä näin yksinkertainen pisteytys ei kuitenkaan tuota toimivaa lopputulosta, paitsi hyvin yksinkertaisissa peleissä. Tästä syystä pelitilanteita täytyy analysoida tarkemmin ja pisteyttää ne erilaisten ominaisuuksien avulla. Insinöörityössä toteutetussa sovelluksessa evaluointifunktio antaa pisteitä mm. liikutettavien nappuloiden määrästä, myllyistä, aukinaisista myllyistä, vastusajan myllyn estämisestä ja mahdollisista tuplamyllyistä. Evaluointi on suhteellisen

raskas operaatio, mutta siihen auttaa siirtovaihtotaulun käyttö, josta kerrotaan minimaxin optimoinnissa tarkemmin.

Kuvassa 7 on esimerkkipelitilanne molempien pelaajien perspektiivistä katsottuna. Kuvan 7 pelitilanteessa pisteitä saa liikutettavista nappuloista, joita tummalla pelaajalla on 6 ja vaalealla 4. Tummalla pelaajalla on lisäksi yksi aukinainen mylly, sekä se on estänyt vastustajan myllyn aukaisemisen. Tästä syystä vaalean pelaajan mylly on merkattu molempiin perspektiiveihin. Vaalealla pelaajalla on yksi mylly, ja se on estänyt vastustajan myllyn luomisen kaksi kertaa. Pelitilanteen lopullinen pistemäärä evaluoidaan siten, että evaluoidaan pelilauta molempien perspektiiveistä ja saadut pistemäärät erotetaan keskenään.

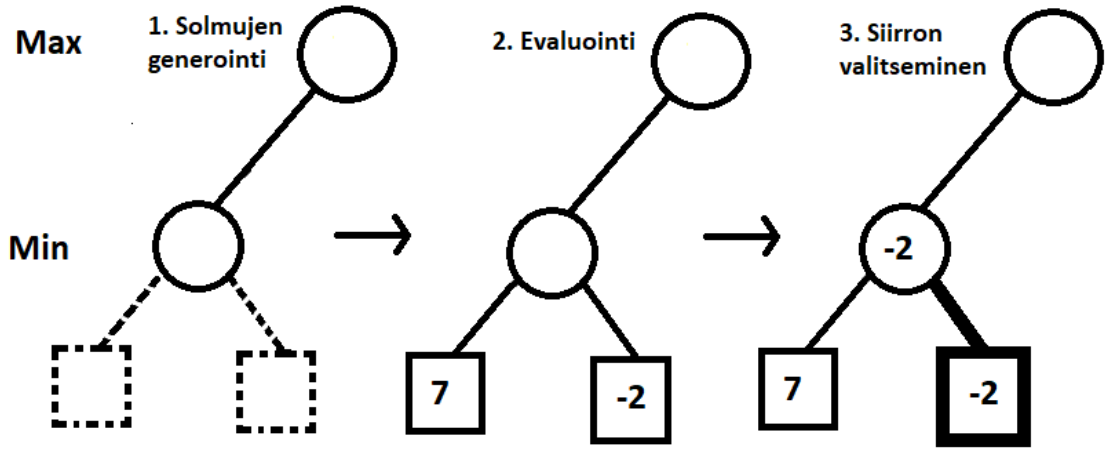



Kuva 7 Esimerkkipelitilanne evaluointifunktiolle. Vasemmalle on merkattu pelitilanne tumman pelaajan perspektiivistä ja vasemmalle vaalean pelaajan.

## 3.2 Minimaxin toimintaperiaate

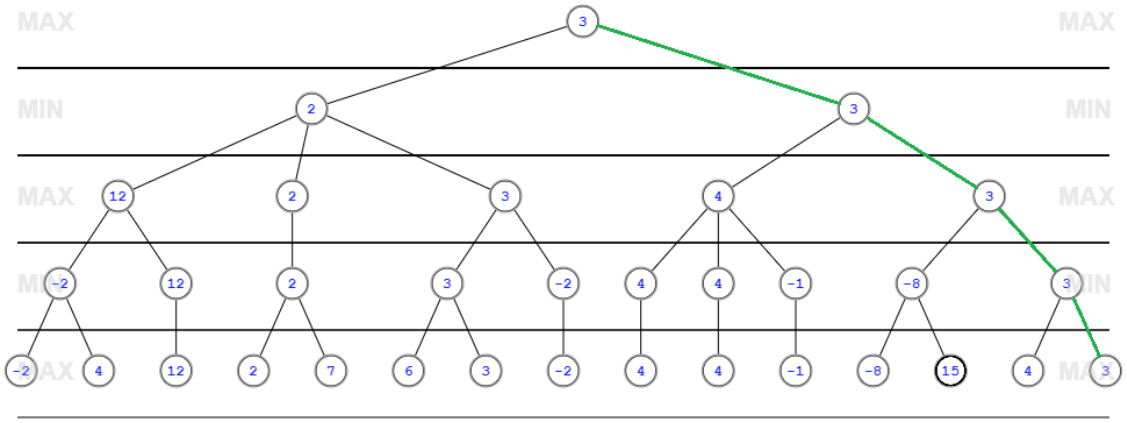
Minimax-algoritmin toimintaa läpikäydessä käytetään esimerkkinä kuvan 6 pelipuuta. Kyseinen pelipuu on hyvin yksinkertainen, ja todellisuudessa pelipuut ovat kooltaan suurempia ja solmuilla on paljon enemmän lapsisolmuja. Yksinkertaisuuden ja selkeyden vuoksi esimerkkinä käytetään vain pientä pelipuuta. Algoritmin toimintaa kertoessa on apuna käytetty lähdettä 7.

Algoritmin toiminta lähtee liikkeelle kaikkien siirtojen generoimisella vuorossa olevalle pelaajalle, eli max-pelaajalle. Näillä siirroilla saadaan luotua uudet pelitilanteet, jotka ovat siis juurisolmun lapsisolmuja. Seuraavaksi luodaan lapsisolmuja yksi kerrallaan, jotka taas tekevät saman kuin sen vanhempisolmu, eli generoivat kaikki siirrot vuorossa olevalle pelaajalle ja niiden perusteella luo uudet pelitilanteet. Tätä jatketaan niin kauan, kunnes saavutaan lehtisolmulle, eli peli on päättynyt, tai on saavutettu haluttu pelipuun syvyys. Kuvassa 8 on havainnollistettu nämä vaiheet kolmena eri vaiheena. Lehtisolmut ovat neliöt, ja niiden arvot ovat kyseisen pelitilanteen arvo, joka saadaan evaluointifunktiolla. Kyseisessä tapauksessa lehtisolmuista valitaan pienin arvo, sillä vuorossa on min-pelaaja.




Kuva 8 Minimax-algoritmin toimintavaiheet.

Juurisolmun kaikki lapsisolmut käydään läpi samalla tavalla kuin ensimmäinenkin lapsisolmu. Kun kaikki lapsisolmut ovat saaneet arvon, juurisolmun lapsisolmuista valitaan suurin luku, sillä juurisolmulla vuorossa oleva pelaaja on aina max-pelaaja. Kuvassa 9 on kuvan 6 pelipuu ratkaistuna ja kuvaan on merkattu vihreällä ”polku”, jota pitkin algoritmi pelaisi seuraavat siirrot sillä tiedolla, mitä sillä on nyt. Huomioitavaa on se, että pelipuun ratkaisu ei siis ole lehtisolmuista suurin arvo.




Kuva 9 Esimerkkipelipuu ratkaistuna minimax-algoritmilla.

Suuremmissa pelipuissa havainnollistuu selkeämmin se, että ainoastaan pelipuun lehtisolmut evaluoidaan ja näistä arvoista saadaan arvot puun muille solmuille. Tämä tarkoittaa siis sitä, että jos puuta halutaan syventää, niin haku on aloitettava alusta, eikä edellistä pelipuuta voida vain jatkaa eteenpäin. Tämä myös selittää iteratiivisesti syventämisen edun, sillä puun syventyessä evaluoitavien solmujen määrä kasvaa nopeasti, eli suuri osa evaluoinnista tehdään vasta viimeisellä iteraatiolla. Tällöin onkin etuna se, että siirrot käydään läpi paremmuusjärjestyksessä. Iteratiivisesti syventämisestä kerrotaan enemmän luvussa 4.

## 3.3 Minimaxin optimointi

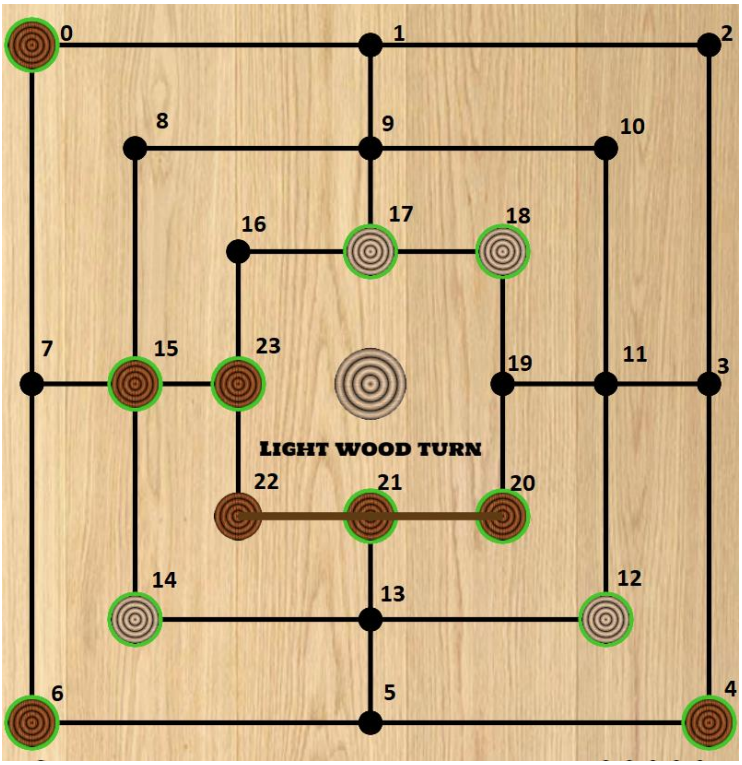
Algoritmin optimoinnin tavoitteena on nopeuttaa algoritmin toimintaa vähentämällä pelitilanteiden evaluointien määrä. Läpikäytävien pelipuunhaarojen määrää voidaan vähentää, kun voidaan päätellä, ettei pelaaja ei ikinä valitsisi kyseistä siirtoa. Pelipuuta läpikäydessä päädytään monesti samoihin pelitilanteisiin eri reittejä pitkin. Tästä syystä on järkevää kehittää tapa, joka poistaa samojen pelitilanteiden evaluoinnin useampaan kertaan. Seuraavissa aliluvuissa kerrotaan edellä mainituista optimointitaktiikoista.

### 3.3.1 Siirtovaihtotaulu

Pelipuuta rakentaessa päädytään usein samaan pelitilanteeseen eri siirtojen kautta, eli siirretään samoja nappuloita eri järjestyksessä. On turhaa evaluoida samaa pelitilannetta useampaan kertaan. Onkin siis järkevää laittaa evaluoitujen pelitilanteiden arvot muistiin, jotta ne voidaan hakea sieltä, kun sama pelitilanne esiintyy uudelleen. On kuitenkin tarkistettava, että pelitilanteet ovat aidosti samat. Pelilauta voi nimittäin näyttää samalta, mutta olla arvoltaan eri. Esimerkiksi, jos pelaajien pelivaiheet eivät ole samat. Tämä on lähinnä ongelma pelaajien siirtyessä ensimmäisestä vaiheesta toiseen. Kolmas vaihe on aina huomattavissa pelilaudasta.

Siirtovaihtotaulun käyttöönotto on suhteellisen yksinkertaista ja sillä saa huomattavia parannuksia aikaiseksi. Siirtovaihtotaulun käyttö vaatii toiminnallisuuden, jolla pelilauta voidaan kuvata esimerkiksi lyhyenä merkkijonona tai numerona, jotta pelilaudan arvolle saadaan siirtovaihtotaulukkoon avain/arvo pari [8].

Toteutetussa sovelluksessa pelilaudan muuttaminen merkkijonoksi, eli hashing, toteutettiin siten, että merkkijonon alku on molempien pelaajien pelivaiheet ja loppuosa on pelilaudan nappulat. Pelivaiheet ovat oleellista merkata, sillä sama pelilauta voi olla arvoltaan eri riippuen pelin vaiheesta. Kuvan 10 esimerkissä molempien pelaajien vaiheet ovat 2, joten merkkijonon ensimmäiset neljä merkkiä ovat ”2L” ja ”2D”. L tarkoittaa vaaleaa pelaajaa ja D tummaa pelaajaa. Loput 24 merkkiä ovat pelilaudan pisteiden ”tilat”. Nolla tarkoittaa tyhjää paikkaa, ja D tai L tarkoittavat kyseisen pelaajan nappulaa. Pelilaudan kohdat on indeksoitu kuvan 10 mukaan spiraalimaiseen tapaan. Tätä samaa järjestystä käytetään sovelluksessa muulloinkin, kuten mahdollisten myllyjen etsiessä tai pelilautaa pisteyttäessä.




Kuva 10 Esimerkkipelilauta merkkijonofunktiolle. Kyseinen pelilauta on merkkijonomuodossa ”2L2DD000D0D00000L0LD0LL0DDDD”.

Hankaluutta hashing-ratkaisussa tuotti se, että uuden ja vanhan myllyn eroa ei voinut nähdä pelilaudasta tehdystä merkkijonosta. Uusi mylly tarkoittaa sitä, että sillä myllyllä ei ole vielä ”syöty” vastustajan nappulaa, eli se on luotu edellisellä siirrolla. Tämä on ongelmallista, sillä pelilaudan arvoa laskiessa annetaan uudelle ja vanhalle myllylle eri määrä pisteitä. Vanhalle ja uudelle myllylle annetaan eri pistemäärä siksi, koska tällä tavoin kannustetaan myllyn avaamista ja uudelleen sulkemista. Toinen tapa tähän kannustamiseen olisi antaa pisteitä vastustajan nappulan syönnistä, mutta tämän heikkous on se, että algoritmille ei sinänsä kerrota, mikä aiheuttaa nappulan syönnin. Tämä tarkoittaa sitä, että pisteitä annetaan ns. yhden askeleen jäljessä.

Ratkaisuksi uuden ja vanhan myllyn erottamiseen voisi olla hyödyntää isoja ja pieniä kirjaimia myllyn nappuloita merkatessa, mutta sovelluksessa ongelma ratkaistiin eri tavalla. Ratkaisu perustui siihen, että sovellus antaa jokaiselle luodulle myllylle uniikin id:n pelaajan siirtonumeron perusteella ja myllyllä on oma muuttuja kuvaamassa, onko mylly uusi vai ei. Kyseiseen muuttujan arvoon

ei voi aina turvautua, sillä taulun arvo lasketaan aina vasta syönnin jälkeen, jolloin kyseisen myllyn muuttujat ovat päivittyneet eikä mylly enää ole uusi. Tässä tapauksessa pitää hyödyntää myllyn id:tä. Sovellus vertaa myllyä pelin alkuperäisen tilan pelaajan myllyihin ja tarkistaa, onko tällä pelaajalla samassa kohdassa mylly ja täsmääkö sen id nykypelilaudan myllyn id:een.

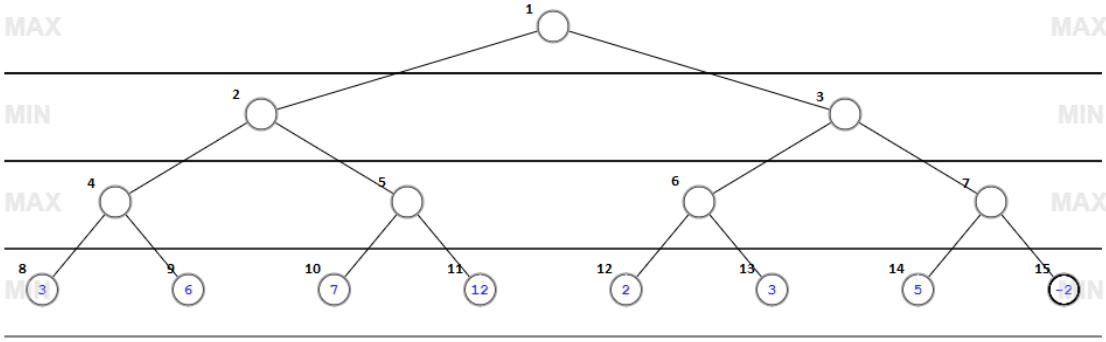
### 3.3.2 Alfa–beta-karsinta

Kaikkia pelipuun solmuja ei ole aina tarpeellista tutkia, vaan algoritmi voi päätellä joidenkin puun haarojen läpikäymisen olevan turhaa ja ne voidaan ns. ”karsia” pois. Kyseistä optimointia kutsutaan alfa-beta-karsinnaksi. Parhaimmillaan alfa-beta-algoritmin tarvitsee käydä puun solmuista vain solmujen neliöjuuren verran, eli kyseessä on todella tehokas optimointitapa [9]. Karsintaa tapahtuu eniten silloin, kun pelin siirrot käydään läpi paremmuusjärjestyksessä alkaen parhaimmasta siirrosta. Tästä syystä on siis perusteltua käyttää hieman aikaa siirtojen tutkimiseen ja järjestää uudelleen tutkittavat siirrot ennen kuin aloitetaan puun läpikäyminen. Myllyssä tyypillisiä hyviä siirtoja ovat mm. myllyn teko, avoimen myllyn teko, vastustajan myllyn estäminen ja myllyn aukaisu. Toteutetussa sovelluksessa käydään kaikki siirrot läpi ja tarkistetaan, tapahtuuko siirrosta jokin näistä ja siirretään se siirtojen alkuun.

Minimax-algoritmin toiminta pysyy pitkälti samana, ja se tuottaa saman lopputuloksen kuin alfa-beta, olettaen, että etsintäikkuna on [-∞ ∞], eli uusinen parametrien alfa ja beta alkuarvot ovat -∞ ja ∞. Erona minimax-algoritmiin on siis, että nyt pidetään muistissa uudet parametrit alfa ja beta. Alfa kuvaa max-pelaajan parasta arvoa, minkä hän voi minimissään saada kyseisellä puun syvyydellä tai ylempänä. Beta kuvaa min-pelaajan parasta arvoa, minkä hän voi minimissään saada kyseisellä puun syvyydellä tai ylempänä.

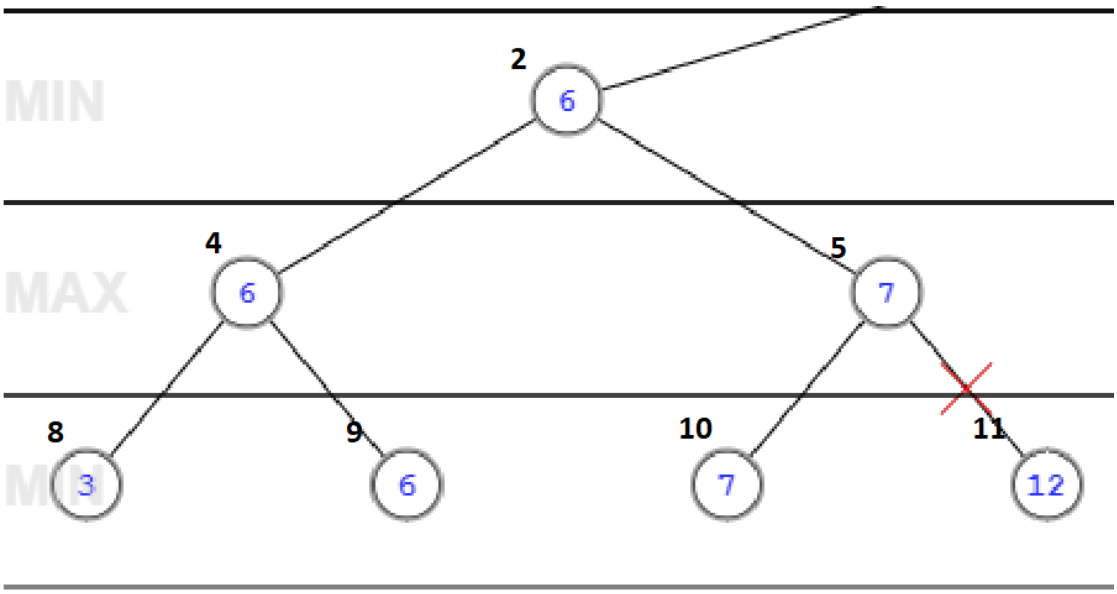
Pelipuuta käydään minimaxin tavoin läpi samalla päivittäen alfan ja betan arvoja aina, kun solmu on valitsemassa itselleen arvoa lapsisolmuista. Samalla tarkistetaan mahdollisen karsinnan mahdollisuus. Min-pelaaja suorittaa karsinnan, jos alfan arvo on pienempää tai yhtä suurta kuin beta. Max-pelaajalle

tarkistus on toisin päin, eli jos beta on pienempää tai yhtä suurta kuin alfa. Kuvassa 11 on esimerkkipelipuu, jossa alfa-beta karsintaa tulee tapahtumaan kahdesti.



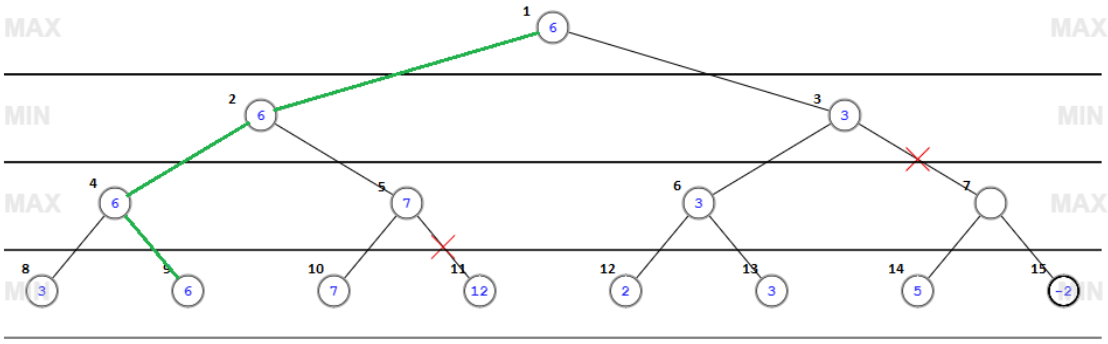

Kuva 11 Esimerkkipelipuu alfa–beta karsinnalle.

Kuvan 11 pelipuussa karsintaa tapahtuu ensimmäisen kerran, kun solmu 5 on tarkistanut ensimmäisen lapsisolmun arvon ja suorittaa sitten karsinnan tarkistuksen. Kyseisen lapsisolmun arvo on 7, joten tiedetään, että solmun 5 arvoksi tulee vähintään 7, sillä kyseessä on max-pelaajan pelivuoro. Tästä on turha käydä solmun 5 loppuja lapsisolmuja läpi, sillä solmulla 2 on jo pienempi arvoinen lapsisolmu löydetty (solmu 4). Tässä esimerkissä solmuilla on vain kaksi lapsisolmua. Muussa tapauksessa karsinnan jälkeen jatkettaisiin käymään solmun 2 lapsisolmuja läpi. Kuvassa 12 on tarkemmin kyseinen karsintatilanne ja siihen on merkattu tapahtunut karsintakohta. Kuva 12 havainnollistaa sen, että vaikka solmun 11 arvo olisi ollut mikä tahansa, tulos ei olisi muuttunut.




Kuva 12 Alfa-beta-pelipuun ensimmäinen karsintatilanne.

Kuvassa 13 on ratkaistu pelipuu alfa-beta-menetelmällä ja siitä nähdään pelipuun toinen karsintatilanne. Karsintaa tapahtuu siis solmun 3 lapsisolmuille. Kyseisessä tilanteessa solmun 3 suurin mahdollinen arvo on 3, joten voidaan päätellä, ettei pelipuun juurisolmu tule valitsemaan solmua 3, koska sillä on jo suurempiarvoinen lapsisolmu tiedossa (solmu 2). Tässäkin tilanteessa huomataan, että olipa solmujen 7,14 ja 15 arvot mitä tahansa, niin lopputulos ei olisi muuttunut.




Kuva 13 Alfa-beta-esimerkkipelipuu ratkaistuna.

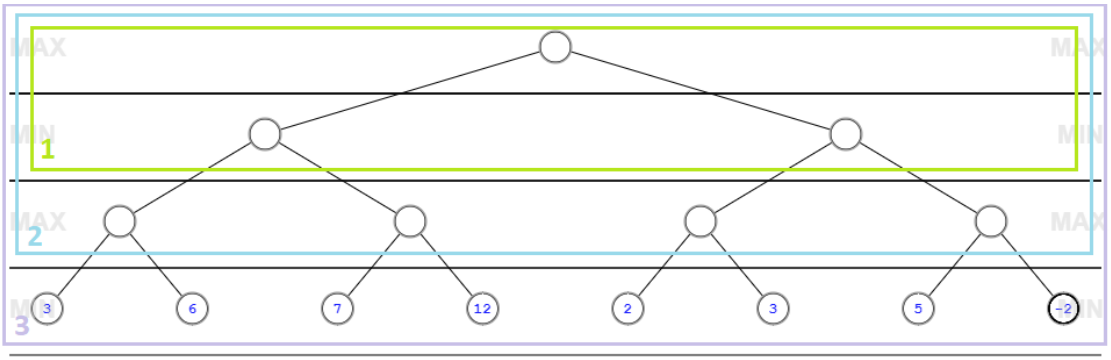
Alfa-beta-karsinta on niin yleinen optimointitapa, että usein minimax-algoritmia kutsutaan suoraan alfa-beta-algoritmiksi. Tässä työssä sitä kuitenkin pidetään vain yhtenä optimointitapana.

# 4 Iteratiivinen syventäminen

Seuraavissa aliluvuissa kerrotaan iteratiivisesti syventämisestä yleisesti, mitä se tarkoittaa ja miten se eroaa minimax-algoritmista. Iteratiivisesti ei oikeastaan ole oma algoritminsa, vaan sitä voisi paremminkin pitää minmax-algoritmin optimointitapana. Insinöörityössä iteratiivista syventämistä kuitenkin käsitellään kuin se olisi oma algoritminsa, sillä iteratiivinen syventäminen ei aina välttämättä nopeuta algoritmin toimintaa vaan päinvastoin. Se voi myös hidastaa sitä.

## 4.1 Iteratiivisesti syventämisestä yleisesti

Iteratiivisella syventämisellä tarkoitetaan pelipuun läpikäymistä vaiheittain pelipuun syvyys kerrallaan. Jokaisella iteraatiolla hyödynnetään edellisen iteraation tulosta [10]. Kuvassa 14 havainnollistetaan, kuinka jokaisella iteraatiolla pelipuun läpikäymissyvyyttä kasvatetaan yhdellä. Iterointi jatkuu niin kauan, kunnes saavutetaan haluttu hakusyvyys tai kun haluttu hakuaika on kulunut loppuun. Itse algoritmina käytetään edellä mainittua minimax-algoritmia. Iteratiivisesti syventämisen yksi ominaisuus onkin se, että käyttäjä voi määrittää algoritmille maksimiajan, jonka se haluaa algoritmin käyttävän siirron etsimiseen.




Kuva 14 Esimerkkipelipuu iteratiivisesti syventämiselle.

Ensivaikutelmana iteratiivisesti syventäminen vaikuttaisi järjenvastaiselta, sillä yhden minimax-haun sijaan käydään useampia hakuja. Sitähän se onkin, ellei algoritmi käytä edellistä hakuaan hyödyksi tehdessään seuraavaa. Iteratiivisen syventämisen hyöty saavutetaan siten, että algoritmi järjestää uudelleen tutkittavien siirtojen järjestyksen niille edellisellä haulla saatujen pisteiden perusteella. Tämä järjestys todennäköisesti muuttuu haun syventyessä, mutta alfa–beta-karsimisen todennäköisyys kasvaa tutkimalla hyvät siirrot ensin. Iteratiivisen syventymisen edun huomaa etenkin silloin, jos sitä vertaa minimax-algoritmiin, jossa käytävien siirtojen järjestys on aina satunnainen, eli niitä ei ole jo ennestään järjestetty paremmuusjärjestykseen.

## 4.2 Iteratiivisesti syventämisen toteutus

Toteutetussa sovelluksessa oli aluksi hankaluuksia ensimmäisten siirtojen uudelleenjärjestämisessä etsintäiteraatioiden välillä. Aluksi tyydyttiin vain siirtämään edellisen haun parhaimman siirron seuraavan haun ensimmäiseksi siirroksi. Tämä ei tosin paljoa auta, sillä haetut siirrot ovat aina jo uudelleen järjestetty karkeasti aikaisemmin kerrotun hyvien siirtojen etsimisen taktiikalla. Tämän toteutuksen tuloksena saatiinkin vain hitaampi minimax-algoritmi, mutta sille voidaan antaa jokin hakuaika.

Tähän toteutukseen tehtiin myöhemmin kuitenkin parannuksia ja lopulta siirtojen uudelleen järjestäminen etsintäiteraatioiden välillä saatiin toteutettua. Parannuksien jälkeen iteratiivisesti syventäminen on mahdollisesti jopa

nopeampi kuin perinteinen minimax-algoritmi. Kuitenkin vain mahdollisesti, sillä niin kuin aikaisemminkin mainittiin, haetut siirrot on jo uudelleenjärjestetty. Uudelleenjärjestäminen hyvien siirtojen etsimisen taktiikalla osoittautui olevan hyvin toteutettu, sillä iteratiivisesti syventämisen ja minimaxin nopeusero ei nopealla testauksella ollut suuri, mutta tästä kerrotaan lisää algoritmien vertailussa. On tosin todettava, että nopeat testaukset painottuivat pelin alkuvaiheeseen, jossa siirtojen väliset piste-erot ovat pienet, mikä vähentää iteratiivisen syventämisen hyötyä. Tosin jos siirtojen uudelleen järjestäminen on pois päältä, niin algoritmien välinen nopeusero selkenee huomattavasti. Tällä tavalla pystyttiinkin todentamaan iteratiivisen syventämisen toteutuksen toimivaksi.

Minimax-algoritmi ja iteratiivisesti syventäminen pitäisi teoriassa antaa aina sama lopputulos hakusyvyyksien täsmätessä. Näin se melkein onkin, sillä tulos on aina samanarvoinen siirto eli ei välttämättä sama siirto. Samanarvoisia siirtoja voi olla useampia, joten ne voivat siitä syystä olla eri siirrot. Tästä syystä algoritmit pelaavat peliä hieman eri siirroin. Tätä eroa voisi vähentää muokkaamalla evaluointifunktiota vähentämällä saman arvoisten siirtojen todennäköisyyttä, mutta tätä ei pidetty tarpeellisena toteutetussa sovelluksessa.

# 5 Monte Carlo -puuhaku

Seuraavissa aliluvuissa kerrotaan Monte Carlo -puuhakumenetelmästä yleisesti, sekä selitetään hakumenetelmän toimintaperiaate. Aliluvussa 5.1 kerrotaan myös kyseisen menetelmän hyviä ja huonoja puolia, sekä millaiseen peliin tai ongelmaan menetelmä soveltuu parhaiten.

## 5.1 MCTS:n yleisesti

Monte Carlo -puuhaku pohjautuu satunnaisuuteen, ja se soveltuu erityisen hyvin peleihin, joiden pelipuun haarautumiskerroin on suuri. Esimerkkejä haarautumiskertoimista Ristinollan ja Neljänsuoran ovat 4, Myllyn on noin 10 [6], Shakin on noin 35 [11] ja Go:n on noin 250 [6]. MCTS-algoritmia tutkiessa

tuleekin aina törmättyä Go-lautapeliin, johon MCTS-algoritmi soveltuu todella hyvin.

MCTS-algoritmin etuna minimaxiin verrattuna on se, että algoritmin ei tarvitse ”tietää” pelistä mitään, ja se voi silti löytää hyvän siirron [12]. Minimax siis vaatii pisteytysjärjestelmän eri pelitilanteille, eli evaluointifunktion. Tämä evaluointifunktio antaa annetulle pelitilanteelle arvon ja tällä tavoin algoritmin pitää tietää pelistä jotain, jotta se voi pisteyttää pelitilanteita. MCTS:n tapa selvittää pelitilanteen arvo, eli kuinka hyvä pelitilanne on pelaajalle, on täysin erilainen. Pelitilanteen arvon selvittämiseksi tarvitaan ainoastaan tapa pelata peli loppuun asti täysin satunnaisilla siirroilla. Tämä tapa perustuu siis siihen, että jos pelitilanteesta voittaa täysin satunnaisin siirroin 80 % ajasta, toisen pelitilanteen 65 % sijaan, niin kyseinen pelitilanne on todennäköisesti parempi.

Minimax-algoritmissa pelipuu generoidaan kokonaan haluttuun syvyyteen asti, mutta MCTS:ssa pelipuuta ei luoda kokonaan, vaan pelipuuta kasvatetaan vain lupaavimmista siirroista eteenpäin. Tästä syystä MCTS on erityisen hyvä peleissä, joilla on suuri haarautumiskerroin. Näissä peleissä minimax-pelipuun koko kasvaa liian nopeasti liian suureksi verrattuna MCTS:n pelipuuhun.

MCTS:n heikkoutena on huomata siirrot, jotka johtavat voittoon nopeasti, tai siirrot, jotka johtavat vastustajan voittoon nopeasti. Toteutetussa sovelluksessa tämä ilmeni etenkin silloin, kun algoritmi oli selvästi johdossa. Algoritmin ollessa selvästi johdossa melkein mikä tahansa siirto johtaa voittoon ja kaikki siirrot tuntuvat melkein yhtä hyviltä. Tämä aiheuttaa hankaluutta löytää aidosti hyvä siirto.

## 5.2 MCTS:n toimintaperiaate

Pelipuun solmussa pidetään tallessa simulointien ja voitettujen simulointien määriä. Näiden kahden arvon avulla määritetään solmun, eli pelitilanteen arvo, jota käytetään algoritmin valintavaiheessa.

MCTS on iteratiivinen algoritmi ja jokaisella iteraatiolla on neljä vaihetta [13]:

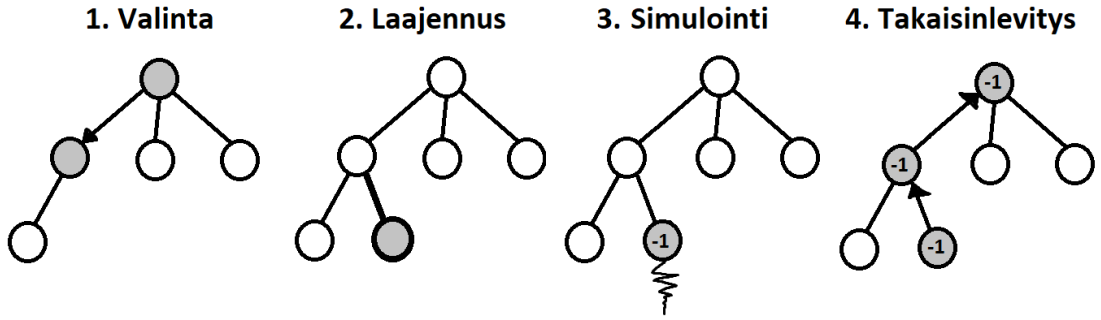
1. Valinta: Valintavaihe on algoritmin ydin. Valinta aloitetaan juurisolmusta ja puussa siirrytään eteenpäin, kunnes saavutaan solmulle, jolle ei vielä ole generoitu lapsisolmuja. Algoritmi valitsee lapsisolmun perustuen lukuarvoon, mikä lasketaan solmun simulointien ja voittojen lukumääristä. Kaavalla, jolla lukuarvo lasketaan, voidaan säädellä, millä suhteella algoritmi kasvattaa pelipuuta suhteessa, paljonko se tutkii siirtojen lupaavuutta. Kaava, jonka muunnosta yleensä käytetään, on UCT eli Upper Confidence Bound 1 applied to trees. Tästä kerrotaan tarkemmin myöhemmin.

2. Laajennus: Tutkitaan, onko solmulla vielä generoimattomia lapsisolmuja. Jos on, niin luodaan solmulle uusi lapsisolmu.

3. Simulointi: Vaihetta kutsutaan myös playout-vaiheeksi. Simuloinnissa pelataan peli loppuun asti kyseisestä pelitilanteesta lähtien satunnaisin siirroin, kunnes voittaja on selvillä.

4. Takaisinlevitys: Käytetään edellisen vaiheen eli simuloinnin tulosta ja siirrytään takaisin juurisolmulle päivittäen matkalla solmujen voitto- ja käyntikertojen määrät.

Kuvassa 15 on havainnollistettu iteraation neljä eri vaihetta. Kuvan 15 pelipuun solmuihin ei ole kuvattu solmun simulointi- tai voittomääriä. Simulointi ja takaisinlevitysvaiheessa arvo ”-1” tarkoittaa, että simuloinnin tuloksena tuli häviö. Toteutetussa sovelluksessa tämä siis tarkoittaa, että juurisolmulle palatessa voittojen lukumäärää ei päivitetä. Jos tulokseksi olisi tullut voitto, niin voittoihin olisi lisätty 1. Simulointien lukumäärää kasvatetaan aina yhdellä kummassakin tapauksessa.




Kuva 15 MCTS-algoritmin iteraation neljä eri vaihetta.

Valintavaiheessa käytetty UCT-arvo lasketaan solmulle kaavalla 1:

$$\frac{w}{n} + c * \sqrt{\frac{\ln N}{n}} \quad (1)$$

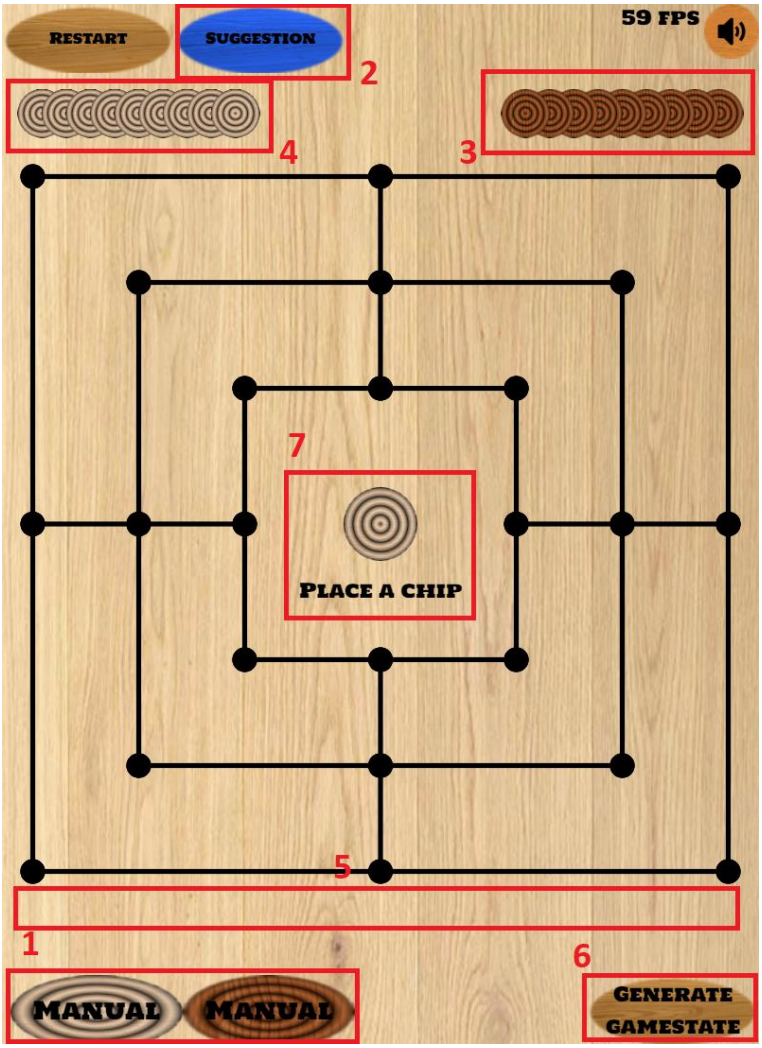
- w tarkoittaa solmun voittojen määrää.
- n tarkoittaa solmun simulointien määrää.
- N tarkoittaa solmun vanhemman simulointinen määrää.
- c on parametri, jolla säädellään pelipuun kasvattamisen ja jo luotujen solmujen lupaavuuden tutkimisen suhdetta. Teoriassa tämä arvo on $\sqrt{2}$ [14], mutta käytännössä se kannattaa säätää pelille sopivaksi.

MCTS-iteraatioita jatketaan niin kauan, kunnes algoritmille annettu laskenta-aika on kulunut. Toteutetussa sovelluksessa algoritmilla on vain staattinen iteraatioiden lukumäärä, minkä se suorittaa. Lopussa pelipuun lapsisolmuista valitaan se solmu, jolla on eniten simulointikertoja. Syy valita siirto simulointimäärien voittomäärien sijaan on aikaisemmin mainitun UCT arvon käyttö valintavaiheessa. Valintavaiheessa etsitään lupaavin solmu UCT-arvoa käyttäen siirryttäessä puussa eteenpäin. Tästä johtuen eniten simulointeja suorittanut solmu on samalla myös lupaavin siirto.

# 6 Toteutettu myllysovellus

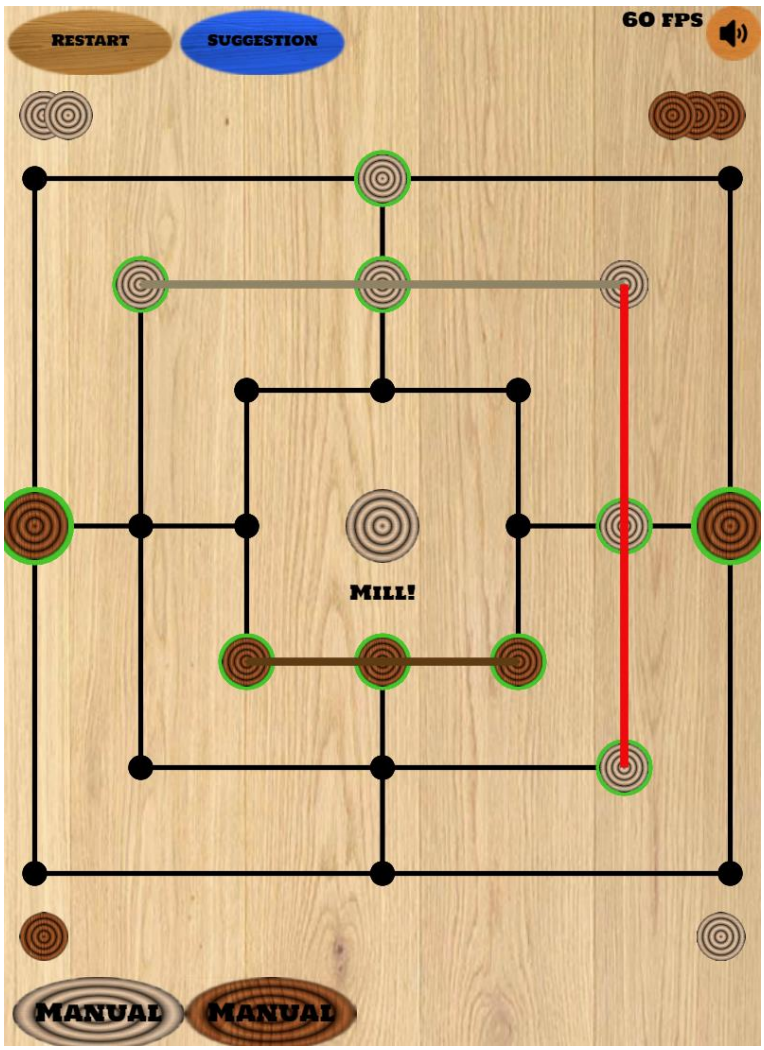
Työssä toteutettu myllysovellus toimii verkkoselaimessa ja on testattavissa osoitteessa https://jukkakot.github.io/. Sovellus on toteutettu JavaScriptillä p5.js www.p5js.org-grafiikkakirjastoa käyttäen. Myllyä on mahdollista pelata tietokonetta vastaan eri algoritmeilla tai antaa tietokoneen pelata eri algoritmeilla itseään vastaan. Molemmille pelaajille voidaan valita haluamansa vaihtoehto pelaajaksi. Vaihtoehtoina on manuaalisesti pelaaminen, täysin satunnaisesti pelaaminen, minimax-algoritmi eri hakusyvyyksillä, iteratiivinen syventäminen eri aika- tai hakusyvyysrajoituksella tai Monte Carlo -puuhaku.

Kuvassa 16 on merkattu sovelluksen käyttöliittymän oleellisimmat kohdat, mitkä on hyvä tietää. Kohdassa 1 voidaan vaihtaa pelaajien algoritmia. Sitä voidaan halutessa vaihtaa myös kesken peli. Nappulasta 2 pelaaja voi halutessaan pyytää seuraavaa siirtoehdotusta, mikä ehdottaa pelaajalle seuraavaa siirtoa jotakin algoritmia käyttäen. Kohdissa 3 ja 4 on pelaajien jäljellä olevat nappulat, ja syödyt nappulat menevät kohtaan 5. Nappulasta 6 voidaan generoida peliin satunnainen pelitilanne, mikä oli erityisen mukava ominaisuus peliä testatessa ja dataa kerätessä. Kohdassa 7 näytetään vuorossa olevan pelaajan pelinappulan väri ja kerrotaan, mitä pelaajan tulisi tehdä. Muut kohdat käyttöliittymällä ovat itsestäänselviä.




Kuva 16 Yleiskuva sovelluksesta.

Kuvasta 17 nähdään tapoja, joilla avustetaan pelitilanteen hahmottamista pelaajalle. Myllyt merkitään pelaajan värisellä viivalla, joka piirretään myllyn muodostaneiden nappuloiden päälle. Jos mylly on uusi, eli se on muodostettu edellisellä siirrolla, se merkataan punaisella viivalla. Nappulat, jotka pelaaja voi ”syödä”, animoidaan, jotta ne on helpompi erottaa. Liikutettavien nappuloiden ympärille piirretään vihreät reunukset, mikä helpottaa pelaajan miettiessä seuraavaa siirtoa ja auttaa huomaamaan tilanteet, jolloin vastustajan nappulat voitaisiin saada jumiin ja täten voittaa peli. Sovelluksessa on myös ääniefektejä nappuloita asettaessa, siirtäessä ja syödessä. Äänet voidaan halutessa asettaa pois päältä.



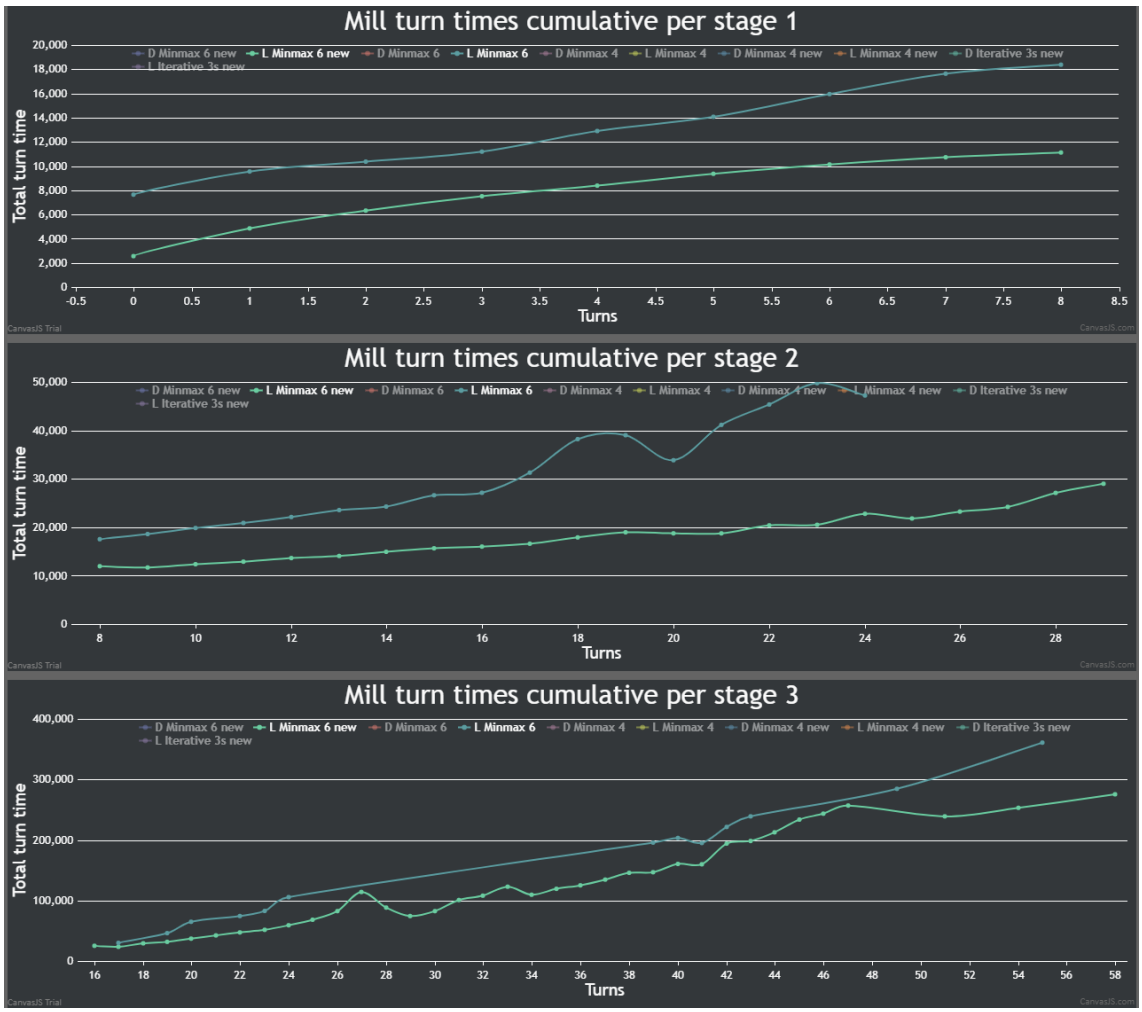

Kuva 17 Esimerkkipelitilanne sovelluksessa.

Sovelluksessa on hyödynnetty säikeitä jakamaan eri työtehtäviä omille säikeilleen. Käyttöliittymä pyörii pääsäikeellä, ja algoritmien toiminta pyörii omalla säikeellään. Tämä mahdollistaa käyttöliittymän pysymisen responsiivisena myös silloin, kun sovellus etsii seuraavaa siirtoa. Säikeistyksellä mahdollistetaan myös algoritmien vaihtaminen kesken pelin vaivattomasti ilman, että joudutaan odottamaan edellisen algoritmin tulosta. Itse algoritmien toimintaa voisi myös nopeuttaa säikeillä rinnakkaistamalla niiden toimintaa, mutta sitä ei sovelluksessa tehdä.

Pelitilastojen keräämiseksi kehitettiin sovelluksen rinnalle palvelimen sovelluksen, johon kerätty data tallennetaan. Pelidatan keräämisen helpottamiseksi sovelluksessa on automatisoitu hakumenetelmien vertailua

keskenään. Ominaisuus aloittaa uuden pelin ja lähettää palvelimelle pelin päättyessä pelidatan itsestään ja täten automatisoi pelidatan keräämisen täysin. Ennen datan keräämistä sovelluksessa tarvitsee vain valita, mitkä kaksi hakumenetelmää halutaan pelaavan vastakkain. Tällä hetkellä palvelinsovellus ei ole käytettävissä missään, vaan sitä voidaan käyttää ainoastaan lokaalisti omalla tietokoneella. Palvelimen sovelluksella generoidaan kerätystä datasta erilaisia kuvaajia pelin eri ominaisuuksista, esimerkiksi myllypelien keskimääräinen pituus, pelin voittaja / häviäjä ja siirtojen keskimääräinen kesto millisekunteina. Kerätty data ja kuvaajat mahdollistavat eri algoritmien vertailun keskenään tai vertailun algoritmeihin tehtyjen muutoksien jälkeen.

Kuvassa 18 on esimerkki minimax-algoritmin vertailusta ennen ja jälkeen evaluointifunktioon tehdyn muutoksen. Kuvassa x-akselilla on siirtonumerot, jotka ovat pelaajakohtaisia, eli molemmat pelaajat aloittavat siirrolla numero 0. Y-akselilla on siirtoon käytetty aika keskimäärin millisekunteina. Kerätty data on jaoteltu pelin kolmeen eri vaiheeseen, sillä toisen ja kolmannen vaiheen pituus ei ole aina sama, ja koska sovelluksessani on omat evaluointifunktiot pelin eri vaiheille. Pelin vaihe on määritelty siten, että se on suurin pelaajilla olevista vaiheista. Peli on siis kolmannella vaiheella, vaikka toinen pelaaja olisi vielä toisessa vaiheessa. Kuvaajista voidaan päätellä, että muutoksen jälkeen algoritmin toiminta on nopeutunut. Tästä ei kuitenkaan voi suoraan päätellä, että muutos oli hyvä, koska uudet siirrot eivät välttämättä ole parempia.




Kuva 18 Esimerkkikuvaaja kerätystä datasta jaettuna pelin kolmeen eri vaiheeseen. Y-akseli on siirtoihin käytetty kumulatiivinen aika millisekunteina ja x-akseli on siirtonumero.

Palvelinpuolen sovellus on käyttöliittymältään hyvin alkeellinen. Sovelluksen tarkoitus ei ollut tulla muiden käyttöön koskaan, vaan toimia vain tämän työn yhtenä työkaluna.

# 7 Algoritmien vertailua keskenään

Valituista vertailtavista algoritmeista minimax-algoritmi ja Iteratiivisesti syventäminen ovat hyvin saman tyyppisiä. Voisi jopa sanoa, että iteratiivisesti syventäminen on yksi tapa optimoida minimax-algoritmia. Tästä syystä näiden algoritmien vertailu keskenään on mielenkiintoista ja niiden välillä voidaan vertailla esimerkiksi hakuaikoja ja evaluoitujen pelitilanteiden määriä. Monte

Carlo -puuhaku on toiminnaltaan täysin erilainen. Tämän lisäksi kyseinen algoritmi ei osoittautunut kovin toimivaksi ratkaisuksi myllyssä. Tästä syystä algoritmien pelatessa keskenään MCTS ei pärjää minimax-algoritmille ollenkaan, vaikka minimax-algoritmin hakusyvyys olisi vain yksi. MCTS kuitenkin voittaa täysin satunnaisesti pelaavan pelaajan melkein aina. Kuitenkin vain melkein aina, sillä simuloidessa 50 peliä MCTS ja satunnaisesti pelaavan pelaajan ollessa vastakkain, MCTS hävisi kerran. Näiden 50 pelin dataa tutkiessa huomattiin, että kyseisen pelin voittohetkellä satunnaisella pelaajalla oli 8 nappulaa jäljellä ja MCTS-pelaajalla oli 2, eli voitto oli täysin murskavoitto. Tästä voitiin siis päätellä, että satunnaisella pelaajalla oli käynyt hyvä tuuri pelin alkuvaiheessa ja hän sai asetettua nappulansa erityisen hyvin ja voitti tästä syystä.

## 7.1 Vertailutavat

Algoritmien vertailua varten toteutettiin sovellukseen kaksi erilaista ominaisuutta. Algoritmit voidaan laittaa pelaamaan toisiaan vastaan ja ne voidaan laittaa myös etsimään seuraavaa siirtoa täysin satunnaisissa pelitilanteissa. Jälkimmäiseen ominaisuuteen voidaan valita niin monta algoritmia kuin käyttäjä haluaa, ja valitut algoritmit etsivät pelitilanteesta seuraavan siirron vuorotellen. Tällä ominaisuudella saadaan kerättyä algoritmien toiminnasta vertailukelpoista dataa, sillä algoritmit hakevat samasta pelitilanteesta seuraavaa siirtoa.

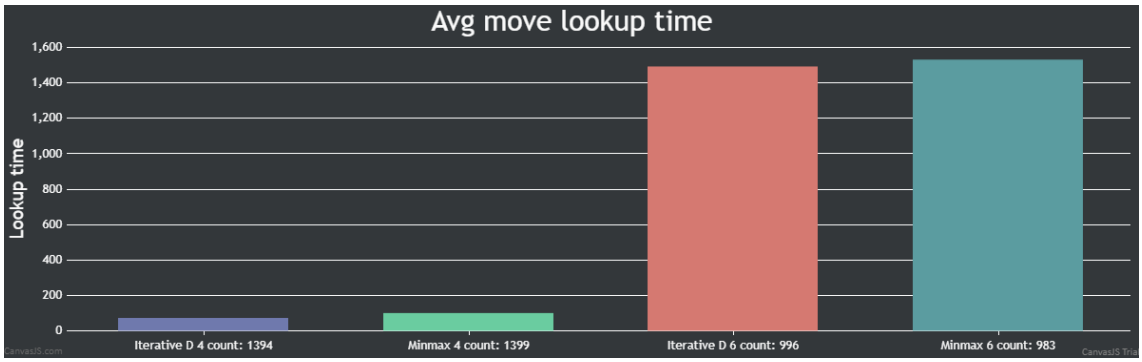
Tässä ominaisuudessa generoidaan ensin jokin satunnainen pelitilanne. Peliä pelataan satunnaisen siirtomäärän verran satunnaisin siirroin ja asetetaan pelitilanteeksi saatu peltilanne. Tämän jälkeen käydään läpi kaikki halutut algoritmit ja otetaan talteen niistä saatu tulos ja siihen kulutettu aika, jonka jälkeen generoidaan taas uusi pelitilanne ja laitetaan algoritmit hakemaan siirtoja. Saadulla datalla voidaan sitten havainnoida algoritmien nopeuksia keskenään. Siirtojen lukumäärä pelitilannetta generoidessa on jokin satunnainen luku väliltä 0 ja 150. Kyseiset luvut valittiin aikaisemmin simuloitujen satunnaisten pelien avulla. Niillä saatiin keskimääräiseksi pelin pituudeksi noin 150 siirtoa.

Vertailuissa vertaillaan algoritmien käyttämää hakuaikaa, sillä se on yksinkertaisin mittari kuvaamaan algoritmin tehokkuutta. On kuitenkin oltava tietoinen siitä, että eri aikoihin kerätty data ei välttämättä ole keskenään vertailukelpoista. Dataa kerätessä on voinut olla eri määrä taustaprosesseja pyörimässä hidastamassa algoritmin toimintaa. Dataa ei voi kerätä usealla eri laitteella ja sitten yhdistää yhdeksi dataksi myöskään. Tekemissä vertailuissa dataa kerättiin vain yhdellä samalla koneella, joten data on vertailuun tarpeeksi hyvää.

## 7.2 Vertailun tulokset

### 7.2.1 Minimax ja iteratiivisesti syventäminen

Kuvassa 19 on minmax-algoritmin ja iteratiivisesti syventämisen vertailua keskenään. Data on generoitu noin tuhannesta pelitilanteesta ja sen perusteella voidaan todeta iteratiivisesti syventämisen olevan hieman nopeampi kuin samaan syvyyteen hakeva minimax-algoritmi. Vertailua varten asetettiin iteratiivisesti syventämiselle maksimisyvyys, eli jos kyseinen syvyys saavutettiin, niin haku lopetettiin siihen. Hakusyvyydellä 4 nopeusero oli noin 30 % ja syvyydellä 6 eroa oli noin 2 %. Kuvaajassa 19 verrataan algoritmeja koko pelin ajalta, mutta jos dataa erotellaan pelin eri vaiheelle, erot ovat selkeämpiä.



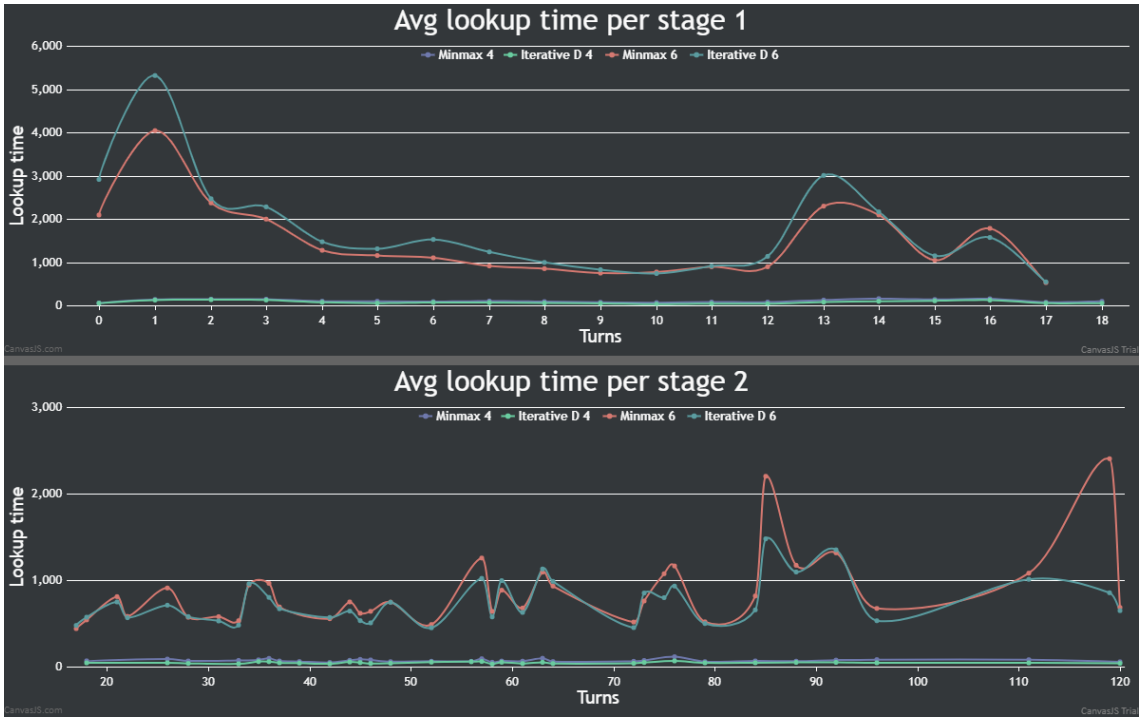

Kuva 5 Minmax-algoritmin ja iteratiivisesti syventämisen vertailua. Palkkien alapuolella lukee käytetty algoritmi ja kuinka monta hakua algoritmilla on tehty. Y-akselilla on keskimääräinen hakuaika millisekunteina.

Kuvassa 20 on sama data pilkottu pelin kolmelle eri vaiheelle. Huomattavaa on se, että iteratiivinen syventäminen ei ole nopeampi pelin jokaisella vaiheella. Syvyydellä 6 minimax-algoritmi on nopeampi ainoastaan pelin ensimmäisellä vaiheella, mutta muulloin iteratiivisesti syventäminen on noin 10 % nopeampi. Syvyydellä 4 iteratiivisesti syventäminen on aina nopeampi, ja nopeuserot heittelevät 10 % - 50 %:n välillä.


Kuva 6 Minmax-algoritmin ja iteratiivisesti syventämisen vertailua keskenään jaettuna pelin kolmeen eri vaiheeseen. Palkkien alapuolella on käytetyn algoritmin nimi ja kuinka monta hakua datassa on. Y-akselilla on hakuaika, jonka algoritmi on keskimäärin käyttänyt.

Vertailussa haluttiin myös tutkia algoritmien ns. hakuaikaprofiilia, eli kuinka paljon se käyttää aikaa keskimäärin pelin eri aikoina. Kuvassa 21 nähdään tämä. Pelin kolmatta vaihetta ei otettu mukaan, sillä sinne ei tullut tarpeeksi paljon dataa, jotta siitä voisi päätellä mitään.

Kuvaajista voidaan ainakin päätellä se, että minimax ja iteratiivisesti syventäminen seuraavat hyvin tarkkaan toisiaan, eli erot ovat pieniä. Suurimmat erot ovat pelin ensimmäisten siirtojen kohdalla, mikä luultavammin johtuu siitä, että iteratiivisen syventämisen edut ovat minimaaliset silloin, kun siirtojen väliset piste-erot ovat pienet. Tämä on erityisesti näkyvissä pelin ensimmäistä siirtoa valitessa, sillä pelilaudan symmetrisyyden takia saman arvoisia siirtoja on monesti ainakin neljä tai useampi.




Kuva 7 Algoritmien keskimääräinen hakuaika eri siirtomäärillä. Y-akselilla on hakuaika millisekunteina, ja Y-akseli on pelin siirtojen kokonaismäärä.

Pelin ensimmäisessä vaiheessa huomataan, että molempien pelaajien ensimmäiset siirrot kestävät huomattavasti kauemmin kuin muut. Tässä olisikin hyvä optimointi-idea, eli parantaa ensimmäisten siirtojen etsintää. Muuten ensimmäisen pelivaiheen kuvaaja näyttää olevan hieman laskusuunnassa, eli

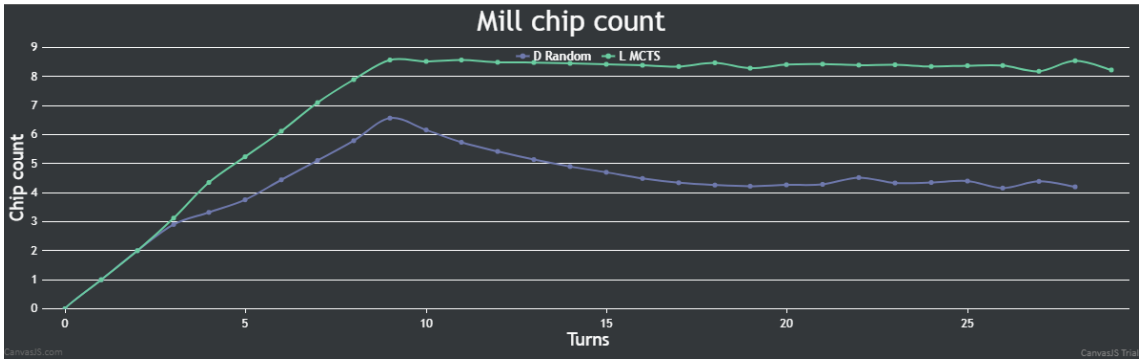
hakuaika laskee pelin edetessä. Ensimmäisen vaiheen lopussa huomataan hakuaikojen kasvavan, mikä johtuu siitä, että algoritmi siirtyy käyttämään eri evaluointifunktiota, kun pelitilanne vaihtuu toiseen vaiheeseen. Pelin toisen vaiheen hakuaikaprofiilista nähdään, että hakuaika on tasaista ja hieman nousujohteista, eli hakuaika kasvaa pelin edetessä, eli todennäköisesti pelinappuloiden vähetessä.

Edellisten havaintojen perusteella voisi todeta, että iteratiivisen syventämisen hyvä käyttötarkoitus optimoinnin lisäksi on siirron maksimihakuajan asettaminen. Jos käyttäjä haluaa sovellukseensa eri tason vastustajia pelaamaan vastaan, niin yksi tapa siihen on vaihtaa algoritmin tekemää hakusyvyyttä. Tämä kuitenkin voi tarkoittaa sitä, että algoritmin hakuaika voi kasvaa joskus aivan liian pitkäksi. Esimerkiksi toteutetussa sovelluksessa hakuajat ovat suurimmillaan pelin ensimmäisten siirtojen kohdalla ja pelin viimeisellä vaiheella, kun erilaisia siirtovaihtoehtoja on paljon. Tähän ratkaisuksi kaikki haut voidaan suorittaa iteratiivisesti syventämällä ja asettaa sille jokin aikarajoitus, jolloin algoritmin toiminta loppuu ja se palauttaa edellisen haun tuloksen. Toinen vaihtoehto eri tason vastustajien tekoon on olla asettamatta maksimihakusyvyyttä, mutta antaa algoritmille jokin maksimihakuaika, jonka algoritmi saa käyttää siirtoa hakiessaan. Tässä ratkaisussa on kuitenkin oleellista tutkia etukäteen, mitkä ovat hyviä hakuaikarajoja eri vaikeustason algoritmeille. Toteutetussa sovelluksessa toteutettiin molemmat vaihtoehdot, mutta maksimihakuajan asettaminen osoittautui mieluisammaksi vaihtoehdoksi.

### 7.2.2 Monte Carlo -puuhaku

Niin kuin aikaisemminkin mainittiin, Monte Carlo -puuhaku ei osoittautunut kovin hyväksi hakumenetelmäksi myllyssä. Tästä syystä sen vertailu muita algoritmeja vastaan ei ollut järkevää, sillä se ei pärjää niille ollenkaan. Vertailua voi kuitenkin tehdä täysin satunnaisesti pelaavaa vastaan. Kuvassa 22 on kuvattu algoritmien keskimääräistä nappuloiden määrää 50 simuloitujen pelien aikana. Kuvassa 22 sininen viiva on satunnaisesti pelaavan nappuloiden määrä ja vihreä viiva on MCTS-algoritmin. Kuvaajasta voidaan päätellä, että MCTS

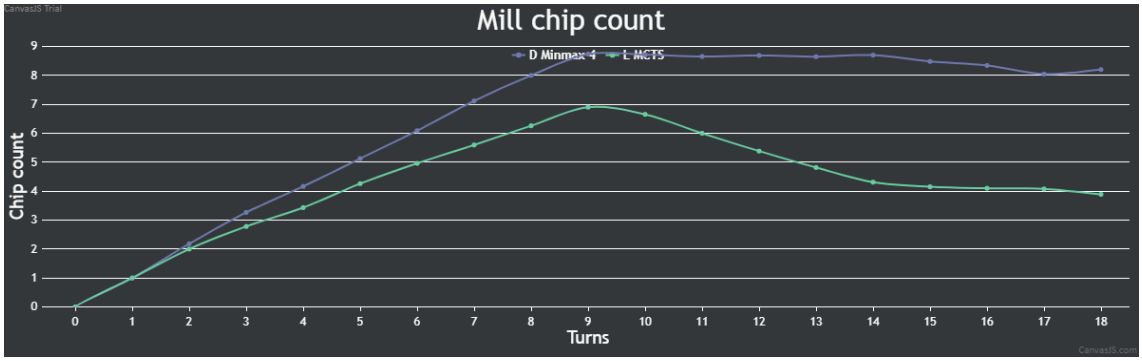
voittaa täysin satunnaisesti pelaavan pelaajan helposti. Kuvaajasta voi ehkä myös huomata MCTS-algoritmin heikkouden löytää nopeaa voittoa tai nopeaa häviötä. Tämä ilmenee siten, että noin 15 siirron jälkeen satunnaisesti pelaavan pelaajan nappuloiden määrä pysyy loppuajan melkein samana. Tämän jälkeen MCTS-algoritmi on niin paljon johdolla, että melkein mikä tahansa siirto johtaa voittoon, mikä vaikeuttaa algoritmin aidosti hyvän siirron löytöä. Tästä syystä itse voittoon johtavan siirron löytäminen voi kestää kauan.




Kuva 22 Monte Carlo -puuhaku vastakkain satunnaisesti pelaavan kanssa. Y-akseli on nappuloiden määrä ja X-akseli on siirtonumero.

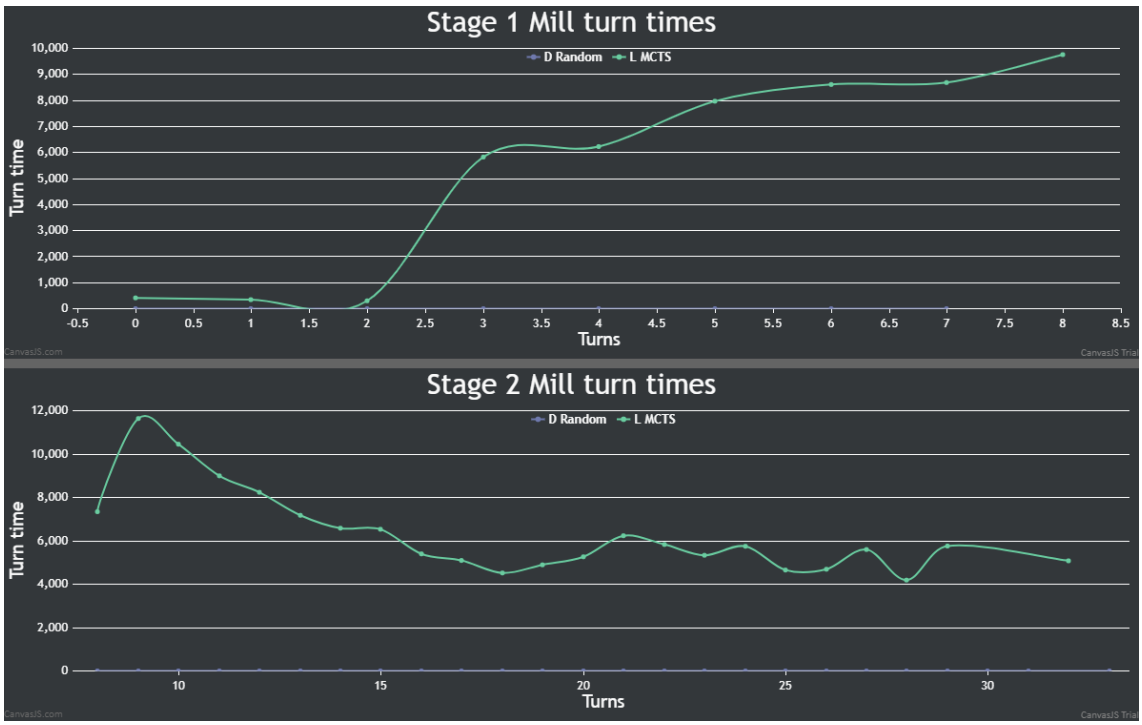
MCTS ei siis pärjää muille hakumenetelmille myllyssä ollenkaan. Tämän voi todeta kuvan 23 kuvaajasta, jossa on 50 pelin tulos MCTS pelaamassa minimax-algoritmia vastaan. Minimaxin hakusyvyys on 4. Jos tulosta vertaa kuvan 21 kuvaajaan, niin ensinnäkin keskimääräisen pelin pituus on huomattavasti lyhyempi ja MCTS-pelaaja menettää nappulansa hieman nopeammin kuin kuvan 23 häviävä pelaaja. Voidaan myös todeta se, että vasta pelin aivan loppu vaiheessa, eli noin 17 siirron kohdalla, minimax-pelaaja on keskimäärin menettänyt vasta ensimmäisen nappulansa. Molemmista kuvaajista huomaa myös sen, että keskimäärin ensimmäinen syönti tapahtuu jo

kolmannella siirrolla, eli ensimmäisellä mahdollisella hetkellä. Tämä kertoo huonosta puolustautumisesta.




Kuva 23 Monte Carlo -puuhaku ja minimax-algoritmi pelaavat vastakkain. Y-akseli on nappuloiden määrä ja X-akseli on siirtonumero.

Vertailun vuoksi tutkitaan myös MCTS-hakuaikaprofiilia. Kuvassa 24 nähdään tämä, jossa MCTS on pelannut satunnaisesti pelaavaa pelaajaa vastaan. Kuvaajasta on taas otettu pelin kolmas vaihe pois, sillä sitä ei ollut aikaisemmassakaan hakuaikaprofiilissa. Hakuaikaprofiili näyttää hyvin erilaiselta minimax-algoritmiin verrattuna. Suurimpana erona on pelin ensimmäisen vaiheen hakuajat ja sen kuvaajan muoto. Jostain syystä MCTS löytää ensimmäiset siirrot todella nopeasti, mutta sitten hakuaika nousee korkeaksi ja jatkaa kasvuaan, kunnes tullaan pelin toiselle vaiheelle, jolloin hakuaika alkaa laskea ja pysyttelee noin 5000 millisekunnin kohdalla loppuun asti.




Kuva 24 Monte Carlo -puuhaku hakuaikaprofiili myllyn eri vaiheittain.

MCTS-algoritmia olisi ollut mielenkiintoista tutkia tarkemminkin ja oikeasti todentaa sen hyvät puolet. Tämä olisi kuitenkin vaatinut algoritmien toteuttamisen johonkin toiseen peliin, jollainen on esimerkiksi Go. Tämä olisi vienyt paljon aikaa, joten tässä työtä varten todettiin riittäväksi vain todeta, että kyseinen algoritmi ei soveltunut myllyyn hyvin ja että algoritmille on kuitenkin omat käyttötarkoituksensa.

# 8 Yhteenveto

Insinöörityön tavoitteena oli tutkia eri hakumenetelmien eroavaisuuksia ja niiden soveltuvuutta myllyssä tai muussa saman tyylisessä pelissä. Tutkiessa havaittiin myös hakumenetelmien muita ominaisuuksia, joita kerrottiin niitä esitellessä.

Työssä onnistuttiin hyvin tavoitteissa ja saatiin selvitettyä hakumenetelmien soveltuvuutta pelipuun ratkaisemiseen. Samalla saatiin selvitettyä hakumenetelmien hyviä ja mahdollisia huonoja puolia, etenkin niitä

implementoidessa tehtyyn sovellukseen. Itse hakumenetelmiä tutkiessa opittiin paljon niiden toiminnasta ja itse pelipuun ratkomisesta paljon.

Algoritmien soveltuvuudesta myllyn pelaamiseen saatiin selville, että minimax-algoritmi ja alfa-beta ovat siihen erittäin hyviä etenkin, jos niitä verrataan monte carlo -puuhaku algoritmiin. Monte carlo -puuhaku ei nimittäin aina voittanut edes täysin satunnaista pelaajaa. Minimax ja alfa-beta algoritmien toteuttamisessa puolet ajasta kului algoritmin perustoiminnan toteuttamiseen, ja loput ajasta kului evaluointifunktion säätämiseen. Evaluointifunktiota voisi säätää loputtomiin ja sillä saataisiin algoritmeista vieläkin parempia. Järkevämpää olisi kuitenkin perehtyä keinoihin, jolla kyseistä säätämistä voisi automatisoida, eli evaluointifunktion säätäminen ilman suurta manuaalista työtä. Tämä olisi ollut mielenkiintoinen tutkinnan kohde työssäni myös, mutta ajan säästämiseksi päädyttiin manuaalisten evaluointifunktion pistearvojen olevan tarpeeksi hyviä.

Työssä onnistuttiin hyvin sovelluksen teossa ja sen soveltuvuudessa työtä varten. Sovellus ei peruskäyttäjälle näytä, että se olisi tehty mitään muuta varten kuin vain myllyn pelaamiseen. Sillä on kuitenkin mahdollista myös tutkia käytettävien hakumenetelmien toimintaa. Sovelluksen käyttäminen hakumenetelmien tutkimiseen vaatii kuitenkin erilaisten näppäinkomentojen käyttämistä, mitä peruskäyttäjän ei ole tarkoitus tietää.

Vaikka työn tavoitteena ei ollut luoda mahdollisimman hyvää algoritmia pelaamaan myllyä, niin tulokseksi saatiin kuitenkin toteutettua melko hyvä ratkaisu siihen. Toteutuksen heikkoutena on pelin ensimmäinen vaihe, sillä sen hyvin pelaaminen vaatisi pelin parempaa tietämystä ja taktiikoiden tuntemista.

Parantamisen varaa jäi kerätyn datan käsittelyssä. Dataa oli helppo kerätä automatisoidun sovelluksen ansiosta ja toteutuksessa otettiin talteen kaikki mahdollinen pelin data. Tämä tarkoittaa sitä, että kerätyllä datalla olisi voinut tehdä hyvinkin yksityiskohtaisia kuvaajia ja vertailuja. Esimerkkinä voitaisiin mainita, että pelin jokainen siirto on tietokannassa noin 50 riviä pitkä objekti. Vertailuissa päädyttiin kuitenkin luomaan melko yksinkertaisia kuvaajia, joista

kuitenkin sai tietoa irti. Datan pilkkominen ja käsitteleminen vaatisi enemmän perehtymistä ja aikaa, joten siihen ei käytetty kovin paljoa aikaa.

## Lähteet


1 Ancientgames. Nine Men’s Morris. Verkkoaineisto. <https://www.ancientgames.org/nine-mens-morris/> Luettu 17.10.2021.

2 Ralph Gasser. Solving Nine Men’s Morris. Games of No Chance. (1996) <http://library.msri.org/books/Book29/files/gasser.pdf/> Luettu 2.5.2022.

3 Fgbradleys. Nine Men’s Morris. Verkkodokumentti. <https://www.fgbradleys.com/rules/rules6/Nine%20Mens%20Morris%20-%20rules.pdf/> Luettu 2.5.2022.

4 Smartboxdesign. Triples Strategies. Verkkoaineisto. <http://www.smartboxdesign.com/ninemmstrategies.html/> Luettu 2.5.2022.

5 Gamescrafters. Nine Men’s Morris. Verkkoaineisto. <http://gamescrafters.berkeley.edu/games.php?game=ninemensmorris#sec-variants/> Luettu 2.5.2022.

6 Allis L. Victor. Searching for Solutions in Games and Artificial Intelligence. Opinnäytetyö. <https://project.dke.maastrichtuniversity.nl/games/files/phd/SearchingForSolutions.pdf/> Luettu 27.4.2022.

7 Javatpoint. Mini-Max Algorithm in Artificial Intelligence. Verkkoaineisto. <https://www.javatpoint.com/mini-max-algorithm-in-ai/> Luettu 2.5.2022.

8 Pascal Pons. Transposition Table. Verkkoaineisto. <http://blog.gamesolver.org/solving-connect-four/07-transposition-table//> Luettu 2.5.2022.

9 Chessprogramming. Alpha-Beta Savings. Verkkoaineisto. <https://www.chessprogramming.org/Alpha-Beta#Savings/> Luettu 2.5.2022.

10 V. Scott Gordon & Ahmed Reda. Trappy Minimax - using Iterative Deepening to Identify and Set Traps in Two-Player Games. Verkkodokumentti. <https://athena.ecs.csus.edu/~gordonvs/papers/trappy.pdf/> Luettu 2.5.2022.

11 Claude E. Shannon. Programming a Computer for Playing Chess. Verkkodokumentti. <http://archive.computerhistory.org/projects/chess/related_materials/text/2-0%20and%202-

1.Programming_a_computer_for_playing_chess.shannon/2-0%20and%202-1.Programming_a_computer_for_playing_chess.shannon.062303002.pdf/ > Luettu 2.5.2022.

12 Benjamin Wang. Monte Carlo Tree Search. Verkkoaineisto. <https://towardsdatascience.com/monte-carlo-tree-search-an-introduction-503d8c04e168/> Luettu 2.5.2022.

13 Chessprogramming. Monte-Carlo Tree Search. Verkkoaineisto. <https://www.chessprogramming.org/Monte-Carlo_Tree_Search#Four_Phases/> Luettu 2.5.2022.

14 Chessprogramming. UCT. Verkkoaineisto. <https://www.chessprogramming.org/UCT#Selection/> Luettu 2.5.2022.