Minimax-pelit

Aluksi

Esimerkkeinä on käytetty lähinnä shakki-, ristinolla- ja connect4-peliä, mutta minimax soveltuu mihin tahansa deterministiseen täyden informaation peliin, ja sitä voi soveltaa myös joihinkin peleihin, joissa on satunnaisuutta tai tuntemattomia tekijöitä. Myös tässä kuvattuja optimointimenetelmiä voi soveltaa kaikkiin tällaisiin peleihin, kun huomioi erot pelien luonteessa. Ristinolla tarkoittaa tässä peliä, jossa voittoon tarvitaan viiden suora, ja pelialue on riittävän laaja, jotta peli ei tilan puutteen vuoksi päädy helposti tasapeliin. Peli tunnetaan myös nimellä gomoku, jolla useimmiten tarkoitetaan 15x15 kokoisella Go-laudalla pelattavaa peliä, jossa saattaa olla eri sääntöjä kuin ristinollassa. Ristinollassa ei ole muita sääntöjä, kuin että kumpikin laittaa vuorollaan merkin laudalle, ja viisi tai useamman omaa merkkiä yhtenäiseen riviin saanut voittaa. Ellei nimenomaan halua toteuttaa gomokua (ja sitäkin on pelattu myös isommilla laudoilla), kannattaa pelilaudan olla vähän isompikin kuin 15x15, jotta tila riittää. Suurempi pelilauta ei vaikuta mitenkään pelin toteutukseen tai laskennan nopeuteen.

Minimax

Minimax on rekursiivinen raa’an voiman algoritmi, joka laskee pelitilanteita eteenpäin ja kokeilee kaikki mahdolliset siirrot joka tilanteessa. Jos tekoäly on maksimoiva osapuoli, tekee se vuoron perään kunkin lähtötilanteessa mahdollisen siirron, ja kysyy minimoivalta algoritmin osalta mikä on minimoijan kannalta paras (eli maksimoijan kannalta heikoin) tulos, jonka minimoija pystyy saavuttamaan tilanteessa, jossa kyseinen siirto on tehty. Minimoija kokeilee vastaavasti jokaisen mahdollisen jatkosiirron, ja kysyy maksimoijalta, mikä on maksimoijan saavuttama tulos, jos minimoija valitsee tämän siirron. Algoritmi kutsuu näin itseään, kunnes löytyy pelin päättävä tilanne tai laskentasyvyys on niin suuri, että on pakko lopettaa rekursio, jotta laskenta-aika ei muodostu liian pitkäksi. Tällöin palautetaan pelitilannetta kuvaava arvo, joka on positiivinen, jos tilanne on maksimoijalle edullinen, ja negatiivinen, jos tilanne on suotuisampi minimoijalle. Kultakin rekursiotasolta palautuu parhaalla siirrolla saavutettu arvo, eli suurin arvo, jos maksimoija on siellä siirtoja tekemässä, ja pienin arvo, jos on minimoijan vuoro tehdä siirto. Lopuksi algoritmi palauttaa arvon, joka saavutetaan lähtötilanteessa parhaalla siirrolla, jos kumpikin pelaa optimaalisesti.

Minimax-algoritmilla voi periaatteessa pelata täydellisesti mitä tahansa vuoropohjaista peliä, jossa algoritmilla on aina käytettävissään täysi tieto pelitilanteesta ja siitä mitä seuraa, jos tekee tietyn siirron. Lasketaan vain loppuun asti kaikki mahdolliset siirtosarjat. Käytännössä ei missään oikeassa pelissä kuitenkaan ole yleisesti mahdollista tutkia kaikkia siirtosarjoja voittoon tai tappioon asti,

koska siihen kuluisi liikaa aikaa. Algoritmin aikavaativuus on eksponentiaalinen niin, että aikafunktion kantalukuna on mahdollisten siirtojen määrä yhdessä pelitilanteessa, ja eksponenttina laskennan syvyys, siis kuinka monta siirtoa lasketaan eteenpäin. Esimerkiksi connect4-pelissä on alkuun aina 7 mahdollista siirtoa, shakissa pari kymmentä ja ristinollassa triviaalisti monta sataa, mutta niistä onneksi suurin osa voidaan jättää tutkimatta ilman vaaraa, että paras siirto jäisi löytymättä karsinnan vuoksi.

Koska minimaxissa tutkitaan miljoonia pelitilanteita, täytyy kaikki tehdä tehokkaasti. Monessa pelissä pitää esimerkiksi tehdä voittotarkistus joka siirron jälkeen. Jos sen ja muutkin yhtä usein tehtävät toimenpiteet toteuttaa oikein huonosti, voi niihin kulua aikaa kymmenkertainen määrä verrattuna hyvään toteutukseen, jopa enemmän jos toteutuksessa on jokin selvä ajatusvirhe. Suorituksen nopeuttaminen kymmenkertaisesti parantaa tekoälyn toimintaa havaittavasti. Connect4-pelissä se tarkoittaa, että voidaan laskea sallitussa ajassa 1-2 siirtoa enemmän eteenpäin, ja muissakin peleissä kymmenkertainen laskentateho mahdollistaa laskentasyvyyden nostamisen yhdellä useissa pelitilanteissa, vaikka ei kaikissa. Sekin saattaa ratkaista pelin voiton. Haluaisimme kuitenkin kasvattaa laskentasyvyyttä mahdollisimman paljon, koska tekoäly pelaa sitä paremmin, mitä pidemmälle se laskee. Kun algoritmin aikavaativuus on eksponentiaalinen suhteessa laskennan syvyyteen, voi syvyyden kasvattaminen neljällä tarkoittaa, että asiat pitäisi tehdä miljoona kertaa nopeammin. Niin paljon paremmin ei vaikkapa voittotarkistusta voi tietenkään tehdä.

Ratkaisu hurjaan tehovaatimukseen on, että useimmiten ei tehdä mitään. Alfa-beta -karsinnan ansiosta ei tarvitse tutkia siirtoja, jotka eivät voi tuottaa parempaa tulosta kuin mitä on jo löydetty. Dynaamisen ohjelmoinnin avulla voidaan muistaa mihin tietty pelitilanne johtaa, eikä sitä ehkä tarvitse laskea uudestaan. Tutkittavia siirtoja voidaan karsia, sillä usein nähdään helposti, että jotkin ja jopa useimmat siirrot johtavat varmaan tappioon. Varma voitto / tappio voidaan ehkä nähdä ennalta, jolloin ei tarvitse laskea siihen asti, että voitto tapahtuu. Todennäköiseen voittoon johtavien siirtojen tutkimiseen kannattaa käyttää enemmän aikaa kuin muiden.

Alfa-beta -karsinta

Kaikkia mahdollisia pelinkulkuja voidaan ajatella puuna. Pelipuussa solmut ovat pelitilanteita, ja niiden välillä on kaari, jos on olemassa siirto, jolla pääsee pelitilanteesta toiseen. Pelin alkutilanne on juurisolmu, ja siitä puu lähtee laajenemaan niin, että jokaisella tasolla on yhä enemmän solmuja. Minimax on tässä puussa tapahtuva syvyyshaku, jota ei kuitenkaan yleensä voida tehdä koko puun laajuudelta, vaan vain juuresta lähtien tietylle syvyydelle asti. Alfa-beta karsinta on menetelmä, jonka ansiosta voidaan jättää tutkimatta sellaiset pelipuun haarat, jotka eivät vaikuta laskennan tulokseen. Koko pelipuun sijasta tarvitsee parhaimmillaan käydä läpi vain hyvin pieni osa siitä, ja silti löydetään aina siirto, joka on paras sen tiedon perusteella, joka pystytään saamaan käytetyllä laskentasyvyydellä. Huomaa, että muiden kuin parhaan siirron arvoa ei yleensä saada selville, koska laskenta katkeaa useimmiten niiden osalta kesken. Pelipuuta ei ole tarkoitus toteuttaa konkreettisena

puurakenteena. Sen sijaan laskennan aikana pidetään kirjaa siitä, mikä on juuri kulloinenkin pelitilanne.

Alfa-beta -karsinta tehostaa minimax-algoritmia merkittävästi. Siinä maksimoija tai minimoija ilmoittaa kutsuessaan toista, että on jo löytänyt tietyn arvoisen siirron, eikä ole kiinnostunut tätä (itselleen) huonommista tuloksista. Minimoijan tapauksessa siirtovaihtoehtojen läpikäynti lopetetaan, jos löydetään siirto, jonka arvo on pienempi tai sama kuin maksimoijan ilmoittama alfa-arvo, maksimoija vastaavasti keskeyttää siirtojen tutkimisen, jos beta-arvo ylittyy. Alfa-beta -karsinta voi teoriassa kaksinkertaistaa tietyssä ajassa saavutettavan laskentasyvyyden, eli toisin sanottuna minimaxin aikavaativuus saadaan pudotettua neliöjuureen alkuperäisestä. Aikavaativuus on edelleen silti eksponentiaalinen. Käytännössä parannus on pienempi, mutta todella merkittävä. Karsinnan tehokkuus riippuu siirtojen käsittelyjärjestyksestä, parhaat siirrot pitäisi käsitellä ensin. Mahdollisimman suuri laskentasyvyys on minimaxissa tärkeää. Hyvällä heuristiikalla saadaan tekoäly pelaamaan paremmin, mutta useimmissa peleissä vain monimutkainen ja huolella opetettu neuroverkkoon perustuva pelitilannetta arvioiva heuristinen funktio pystyy merkittävästi korvaamaan laskentasyvyyttä. Jo heuristiikan käsite tarkoittaa myös sitä, että joissain tilanteissa valitaan sen perusteella siirto, joka ei ole paras, pahimmillaan ei edes hyvä. Laskentasyvyyttä kasvattamalla nähdään, toteutuuko se mitä heuristiikka ennustaa.

Pseudokoodi

Minimaxin voi toteuttaa joko kahtena erillisenä metodina tai alla olevaan tapaan yhtenä metodina, jossa on kaksi osaa: maksimoija ja minimoija. Minimax palauttaa tiedon yksittäisen pelitilanteen arvosta. Sen voisi muokata myös palauttamaan parhaan seuraavan siirron, mutta siirron valintaan kannattaa usein käyttää erillistä metodia, vaikka sinne tuleekin minimaxin kanssa toisteista koodia. Algoritmin tehostaminen vaatii toimia, jotka pitää tehdä minimaxin ulkopuolella, ja ensimmäisessä maksimoijassa saatetaan tehdä asioita, joita ei minimaxissa tarvitse tai kannata tehdä. Esimerkiksi mahdollisuus rivin tekemiseen ristinollassa tarkistetaan ehkä vain lähtötilanteessa, koska siitä eteenpäin syntyvä voittotilanne on mahdollista havaita jo ennalta, eikä tarvitse edetä niin pitkälle, että rivi syntyy. Jatkossa oletetaan, että meillä on minimaxin lisäksi metodi valitseSiirto, joka kokeilee siirtovaihtoehtoja ja kutsuu minimaxia selvittääkseen mikä niistä johtaa parhaaseen lopputulokseen. Tämä metodi palauttaa tiedon siitä, minkä siirron tekoäly tekee.

```
1     int minimax(pelitilanne, syvyys, α, β, maksimoivaPelaaja)
2           if syvyys = 0 or peli on päättynyt
3                 return pelitilanteen heuristinen arvo
4           if maksimoivaPelaaja
5                 arvo := −∞
6                 for siirto: mahdolliset seuraavat pelitilanteet
7                       arvo := max(arvo, minimax(siirto, syvyys − 1, α, β, false))
8                       α := max(α, arvo)
9                       if arvo ≥ β
10                      break
11                return arvo
12          else
```

```
13                  arvo := +∞
14                  for siirto: mahdolliset seuraavat pelitilanteet
15                        arvo := min(arvo, minimax(siirto, syvyys − 1, α, β, true))
16                        β := min(β, arvo)
17                        if arvo ≤ α
18                              break
19                  return arvo
```

Funktiota kutsutaan

minimax(arvioitava pelitilanne, laskennan syvyys, -∞, +∞, true),

jos seuraava siirto on maksimoijan, mutta kun valitseSiirto tekee maksimoijan ensimmäisen siirron, kutsutaan

minimax(arvioitava pelitilanne, laskennan syvyys, α, +∞, false).

Syvyysparametrin arvo pienenee yhdellä joka kerta, kun minimax kutsuu itseään, joten kaikkiaan lasketaan niin monta siirtoa eteenpäin kuin **laskennan syvyys** määrää, kun ensimmäinen siirto tapahtuu metodissa valitseSiirto. Siten saadaan tietää, onko voitto saavutettavissa tai tappio vääjäämätön näin monen siirron aikana, jos kumpikin pelaaja pelaa optimaalisesti.

Rivi 1:

Pelitilanne on pelitilanteen kuvaus yhdellä tai useammalla parametrilla esitettynä. Esimerkiksi ristinolla-pelissä se voisi olla kaksiulotteinen taulukko pelilaudasta, mutta luultavasti kannattaisi välittää muutakin tietoa laskennan tehostamiseksi, kuten edellisen siirron koordinaatit. Shakki-pelissä pelilauta ei kerro kaikkea tarvittavaa tietoa, vaan pitää muistaa onko sotilaita, torneja tai kuningasta liikutettu pelin aikana. Funktio saattaa käytännössä palauttaa pelitilanteen arvon lisäksi muutakin tietoa, kuten tiedon parhaasta siirrosta ja siitä onko paluuarvo todellinen vai alfa- tai beta-katkaisun takia syntynyt.

Rivit 2-3:

Syvyydestä riippumatta tarkastetaan aina ensin, onko peli päättynyt jomman kumman voittoon tai tasapeliin. Moni peli päättyy voittosiirtoon, jolloin voittotarkistukseksi riittää tutkia onko edellisen siirron tehnyt pelaaja voittanut tämän siirron seurauksena. Näin on esimerkiksi shakissa, ristinollassa, connect4:ssä ja tammessa. Jos peli ei ole päättynyt, mutta on tultu syvyyteen 0, palautetaan jonkinlainen arvio pelitilanteen potentiaalista. Rivillä 3 puhutaan heuristisesta arvosta, mutta se voi tässä tarkoittaa myös tietoa jomman kumman voitosta tai tasapelistä, mikä on varmaa tietoa eikä heuristiikkaa. Joissain peleissä, kuten othellossa, vaihtuu siirtovuoro, jos pelaaja ei voi tehdä mitään siirtoa. Tällöinkin palautetaan pelitilanteen heuristinen arvo.

Rivit 4-11:

Maksimoija kokeilee kaikki mahdolliset siirrot (kaikki sallitut siirrot tai jokin niiden osajoukko) tässä pelitilanteessa (rivi 6) ja kutsuu minimoijaa selvittääkseen kuinka hyvään pelitilanteeseen kukin siirto lopulta johtaa (rivi 7). Muuttuja **siirto** kuvaa uuden pelitilanteen, kun on tehty jokin siirto muuttujan **pelitilanne** kuvaamassa tilanteessa. Muuttujassa **α** on tieto parhaan joko rivin 6

silmukassa tai jo aiemmin tutkitussa pelipuun haarassa löydetyn siirron arvosta. Minimoijalle välitetään alfa-arvo kertomaan, että tätä huonommista arvoista ei olla kiinnostuneita, vaan laskenta tulee keskeyttää, jos minimoija löytää siirron, joka tuottaa tämän tai alemman arvon. Parhaan löydetyn siirron tuottamasta arvosta pidetään kirjaa muuttujassa **arvo**, ja myös muuttujaa **α** päivitetään, mikäli **arvo** ylittää aiemman alfa-arvon. Siirtojen tutkiminen lopetetaan, mikäli löydetään siirto, joka johtaa β-arvoa parempaan pelitilanteessen. β-arvo kertoo, että minimoija on jo aiemmin löytänyt sen arvoisen siirron, eikä siis tule tekemään muuttujan **pelitilanne** mukaista siirtoa, jos tulos on minimoijan kannalta sama tai huonompi. Siksi on turhaa tutkia löytääkö maksimoija omalta kannaltaan vielä parempia eli minimoijalle huonompia siirtoja. Rivillä 11 palautetaan arvo, joka kertoo millaiseen pelitilanteeseen joudutaan, kun valitaan tässä pelitilanteessa maksimoijalle paras siirto, ja seuraavilla siirroilla niin minimoija kuin maksimoijakin valitsevat aina itselleen parhaan siirron. Algoritmi on kuvattu abstraktilla tasolla, kuten se on tapana esittää, paitsi tässä ei käytetä pelipuun termejä, eli puhutaan pelitilanteista eikä solmuista. Käytännön toteutuksessa ei ole syytä käyttää listaa erilaisista seuraavia pelitilanteita kuvaavista olioista tms., vaan loopataan listaa seuraavista siirroista, esim. ristinollassa listaa kaikista vapaista ruuduista, joiden lähellä on ennestään oma tai vastustajan merkki. Pelitilannetta päivitetään kullakin näistä siirroista vuorollaan, ja välitetään päivitetty pelitilanne minimoijalle. Rivin 7 jälkeen pelitilanne palautetaan ennalleen, tai sitten minimax-kutsussa on välitetty muokattu klooni pelitilanteesta, jolloin palautusta ei tarvita.

Rivit 12-19:

Kuten yllä, mutta on minimoijan vuoro, ja etsitään parasta tulosta eli alinta arvoa, johon minimoija voi yltää tässä pelitilanteessa.

Pelitilanteen arvo

Maksimoivan pelaajan voitolle annetaan suurin arvo, tappiolle pienin. Tasapelin luonnollinen arvo on 0, mutta pelistä ja lähestymistavasta riippuen saatetaan tasapelitilannetta arvioida muulla tavalla. Huomaa, että myös pelitilanteen potentiaalia kuvaava heuristinen arvo voi olla 0. Jos pelitilanteen potentiaalia arvioidaan heuristisesti, on tilanteen arvo jotain maksimi- ja minimiarvojen väliltä. Eri siirtomäärillä saavutettavat voitot täytyy erottaa toisistaan. Muuten voi käydä niin, että tekoäly lykkää voittoa jopa loputtomiin, koska löytyy siirtoja, jotka myös johtavat voittoon, mutta myöhemmin. Jos tutkittavia siirtoja joudutaan karsimaan heuristisesti niin, että relevanttejakin siirtoja jää joskus tutkimatta, voi vastapelaaja saada voiton lykkäämisen takia torjuttua voiton kokonaan, koska varmaksi arvioitu voitto ei olekaan sitä, vaan sen voi tuorjua. Toki myös arvio nopeimmasta varmasta voitosta voi tällöin olla virheellinen, mutta virheen mahdollisuus kasvaa, kun siirtosarja pitenee. Ongelma korjaantuu antamalla voitolle sitä korkeampi arvo, mitä nopeammin se saavutetaan, esim. (10000 - voittoon johtavien siirtojen määrä). Vastaavasti tappioita kannattaa lykätä siinä toivossa, että vastapelaaja ei kykene niitä havaitsemaan, joten nopea tappio saa vielä pienemmän arvon kuin hidas.

Kun pelin siirtosarjoja ei ole mahdollista aina laskea voittoon tai tappioon asti, pitää laskenta lopettaa sellaiseen syvyyteen, joka saavutetaan hyväksyttävässä ajassa. Jos pelitilanne ei 0-tasolla ole kummankaan voitto eikä tasapeli, kannattaa yleensä palauttaa jokin arvio siitä, kuinka hyvä pelitilanne on, millainen potentiaali sillä on. Jos pelitilanteen arvoa on kovin vaikea arvioida merkityksellisesti, voi kuitenkin olla parempi palauttaa silloin 0, eli neutraali tilanne. 0-tasolla tapahtuva laskenta suoritetaan hyvin monta kertaa, joten jättämällä huonosti kuvaava heuristinen arvo laskematta, saatetaan säästää niin paljon aikaa, että pystytään nostamaan laskentasyvyyttä. Ristinollassa ja connect4:ssä tasapeli syntyy vain, kun pelilauta tulee täyteen, ja jos pystytään laskemaan siirtoja niin pitkälle, nähdään myös kaikki mahdolliset saavutettavissa olevat voitot. 0 tasapelin arvona tuottaa silloin oikean tuloksen, eli valitaan voitto, jos se on saavutettavissa, ja tasapeli mieluummin kuin tappio.

Siirtojen rajaaminen

Esim. shakki-pelissä voidaan suuri osa siirroista jättää tutkimatta, jos käytössä on erittäin kehittynyt heuristiikka, joka ei juuri koskaan hylkää parasta siirtoa. Tällöin voidaan saavuttaa suuri laskentasyvyys, mutta sellaiset heuristiikat ovat neuroverkkoalgoritmeja. Ristinollassa on pakko karsia siirtoja, sillä esimerkiksi jo 15x15 -kokoisessa pelilaudassa on 225 ruutua. Pelissä paras siirto tehdään usein jonkin aiemman oman tai toisen merkin viereen, mutta jos vain viereiset ruudut kokeillaan, on helppo voittaa tekoäly tähtäämällä kaksoispakotukseen, joka syntyy ruutuun, jota tekoäly ei kokeile:

```
XX                          XX X
                      →
  X                           X
  X                           X
```

Ilmeisesti paras siirto ei kuitenkaan koskaan ole yli 2 ruudun päässä aiemmista siirroista, sillä gomoku-kilpailuissa on ennen käytetty aloittajan edun tasoittamiseksi sääntöä, että aloittajan tulee tehdä toinen siirtonsa vähintään kolmen ruudun päähän aimmista siirroista. Siten riittää tutkia aiempien siirtojen naapuriruudut ja niiden naapuriruudut. Tätäkin voi sopivin menetelmin rajata niin, ettei parasta siirtoa jätetä tutkimatta ainakaan lähes koskaan. Veisi kohtuuttomasti aikaa käydä koko pelilauta läpi ja etsiä naapurit kahden ruudun etäisyydellä kaikille aiemmin tehdyille siirroille aina, kun tullaan minimaxiin ja aletaan käydä läpi mahdollisia seuraavia siirtoja. Lisäksi ristinollassa kannattaa usein tehdä siirto lähelle toisen pelaajan edellistä siirtoa tai omaa edellistä siirtoa, katso luku Siirtojen järjestäminen. Siten ristinollassa kannattaa välittää eteenpäin lista kaikista ruuduista, jotka on tarpeen tutkia. Kun kokeillaan siirtoa tässä listassa, kloonataan saatu lista, poistetaan kokeiltava siirto kloonista, koska se ei enää ole tyhjä ruutu, ja lisätään sinne kokeiltavan siirron naapurit. Jos ne ovat ennestään listalla, nostetaan ne listan kärkeen.

Iteratiivinen syveneminen

Iteratiivinen syveneminen tarkoittaa sitä, että ei suoraan lasketa peliä jollekin kiinteälle syvyydelle, vaan ensin lasketaan vain jokunen siirto eteenpäin, ja sitten yhtä siirtoa syvemmälle jne., niin kauan kuin aikaa riittää. Riippuu pelistä ja järjestämisheuristiikoista miltä syvyydeltä kannattaa aloittaa. Voi olla, että vaikka syvyydeltä 1 tai 2 saatava tieto ei vielä ole merkityksellistä, jolloin siirtojen järjestäminen sen mukaan saisi siirrot huonompaan järjestykseen, kuin missä ne ovat perusheuristiikan mukaan. Ensimmäinen hyöty, joka iteratiivisesta syvenemisestä saadaan, on että voidaan suorittaa laskentaa niin kauan kuin aikaa on käytettävissä. Laskenta tietylle syvyydelle asti kestää eri pelitilanteissa hyvin erilaisen ajan. Jos valitaan jokin kiinteä syvyys niin, että siirto tehdään varmasti aina esimerkiksi puolessa minuutissa, joudutaan käyttämään niin alhaista syvyyttä, että joidenkin siirtojen laskentaan kuluu alle sekunti, ja olisi voitu laskea pitemmällekin.

Merkittävin parannus saavutetaan järjestämällä siirrot uudelleen iteraatioiden välissä. Jo pienellä laskentasyvyydellä saadaan usein selville mitkä siirrot ovat kussakin pelitilanteessa hyviä sen pelaajan kannalta, joka silloin on siirtoa tekemässä, ja kun seuraavalla iteraatiolla kokeillaan ensin näitä siirtoja, karsii alfa-beta -katkaisu paljon pelipuun haaroja. Pelitilanteiden arvoja voidaan tallentaa välimuistiin, jolloin kaikkia niitä ei tarvitse selvittää uudestaan seuraavalla iteraatiolla. Koska algoritmin aikavaativuus on eksponentiaalinen suhteessa laskentasyvyyteen, tehdään suurin osa laskennasta joka tapauksessa 0-tasolla, joten samojen pelitilanteiden läpikäyminen uudestaan ei muutenkaan vie suhteessa niin paljon lisää aikaa kuin voisi ajatella.

Siirtojen järjestäminen

Siirrot pyritään järjestämään jo ennen ensimmäistä iteraatiota niin, että paras siirto löytyisi mahdollisimman pian. Yksi tapa on tehdä ensimmäinen iteraatio syvyydellä 1, eli laskea kunkin siirron tuottaman pelitilanteen heuristinen arvo. Tällöin siirrot käydään läpi metodissa valitseSiirto, joka kutsuu minimaxia syvyysparametrilla 0. Sitten siirrot järjestetään näiden arvojen mukaiseen järjestykseen, parhaasta huonoimpaan päin. Tätä kannattaa kokeilla eim. 2048-pelissä, mutta usein jokin muu tapa on parempi. Aloitteen saaminen pelissä ja sen säilyttäminen on edullista, ja muutenkin hyökkäävät tai pakottavat siirrot kannattaa tutkia ensin, joten ne voi nostaa siirtolistan kärkeen sekä metodin valitseSiirto alussa että minimaxissa. Usein kannattaa shakissa shakata tai syödä toisen nappuloita, ristinollassa tehdä nelonen tai avokolmonen, othellossa vallata mahdollisimman monta vastustajan nappulaa jne. Shakki-tekoälyistä löytyy paljon kirjallisuutta, jossa käsitellään myös siirtojen järjestämistä etukäteen ja laskennan aikana. Kannattaa tutustua esimerkiksi käsitteeseen killer moves, jota voi kokeilla muissakin peleissä, mutta esimerkiksi ristinollassa tai connect4:ssä sen oletukset eivät päde.

Ristinollassa on usein pakko tai edullista reagoida toisen pelaajan edelliseen siirtoon, ja toissijaisesti kannattaa jatkaa oman edellisen siirron pohjalta. Se ei aina tarkoita siirtoa ihan edellisen siirron

viereen, mutta tämä toimii hyvin perusheuristiikkana siirtojen järjestämiseen. Jokaisen pelissä varsinaisesti tapahtuvan ja laskennassa kokeiltavan siirron jälkeen kasvaa niiden lähiruutujen joukko, joihin pitää kokeilla siirtoa. Nämä uudet ruudut kannattaa lisätä listaan niin, että viimeksi lisätyt tulevat listan kärkeen. Jos ruutu on listassa jo ennestään, nostetaan se kärkeen. Tällöin listan alussa ovat aina vastustajan edellisen siirron lähiruudut. Jos tätä vielä optimoidaan, nostetaan sitten ihan kärkeen ainakin ne omat siirrot, jotka tuottavat pakotuksen. Tämän listan klooni, josta on poistettu kulloinkin tehtävä siirto, lähetetään parametrina minimax-kutsussa, jotta seuraavalla syvyydellä on käytössä valmiiksi perusjärjestetty siirtolista. Connect4-pelissä paras siirto löytyy usein keskeltä, koska silloin nappula voi kuulua mahdollisimman moneen erilaiseen neljän riviin, joten siirtojen perusjärjestys on yksinkertaisesti sarakkeet keskeltä laitoja kohti: 4, 3, 5, 2, 6, 1, 7.

Usein siirto, joka tuotti parhaan tuloksen, on hyvä tai jopa paras, myös kun lasketaan yhtä siirtoa pitemmälle. Metodissa valitseSiirto voidaan yksinkertaisesti nostaa edellisen iteraation paras siirto listan ensimmäiseksi, mutta tarvitaan välimuisti, jotta sama voidaan tehdä minimaxissa kaikilla laskennan tasoilla. Dynaamisten listojen alkuun lisääminen on hidas operaatio, joten on nopeampaa loopata listaa takaperin, jolloin kärkeen nosto tarkoittaa lisäämistä listan loppuun. Siirtolistan hyvä järjestäminen ennen iteroinnin alkua on tärkeää. Vaikka iteraatioiden välissä tehtävät nostot usein tuovat nopeasti parhaan siirron ensimmäiseksi, ei muita siirtoja voi iteraation tulosten perusteella juurikaan järjestellä. Alfa-beta -karsinnan vuoksi niille saadut arvot eivät ole aitoja. Siirtojen esijärjestykseen optimointiin voidaan myös käyttää reilusti aikaa, koska se tehdään vain kerran. Välillä käy niin, että matalalla laskentasyvyydellä parhaaksi arvioitu siirto osoittautuukin jossain vaiheessa huonoksi. Varsinkin silloin on edullista, jos seuraavat listassa olevat siirrot ovat mahdollisimman hyviä.

Transpositiotaulu

Samaan pelitilanteeseen voi päätyä useamman eri siirtosarjan kautta. Transpositiotauluun voidaan tallettaa edellisessä luvussa mainittu tieto siitä, mikä siirto on viimeksi tuottanut parhaan tuloksen tässä tilassa, sekä pelitilanteen arvo ja tietoa siitä kuinka tätä arvoa voidaan hyödyntää, kuten millä laskentasyvyydellä on oltu, kun arvo on talletettu, ja onko arvon laskennassa tapahtunut alfa- tai beta-katkaisu. Heuristiset arvot ovat vanhentuneita, kun seuraavalla iteraatiolla tullaan samaan pelitilanteeseen, koska silloin lasketaan yhtä siirtoa pitemmälle, ja saadaan parempaa tietoa siitä, mihin pelitilanteesta päädytään. Talletettujen arvojen hyödyntämisen minimaxissa tekee muutenkin vähän monimutkaiseksi se, että alfa-beta katkaisujen vuoksi ne eivät yleensä ole todellisia tilanteen arvoja, vaan joko ylä- tai alarajoja todelliselle arvolle. Jos tiedetään, että kyseessä on varmaa voittoa, tappiota tai tasapeliä kuvaava pelitilanteen todellinen arvo, voidaan se palauttaa suoraan. Myös heuristinen arvo voidaan palauttaa, jos kyseessä on todellinen arvo, ja lisäksi ollaan samassa iteraatiossa, jossa se on laskettu. Pelkästään parhaan arvon tuottaneen siirron talletuksella ja tämän tiedon käyttämisellä siirtojen järjestämiseen saadaan kuitenkin jo suuri hyöty. Jotta voidaan tallettaa paras siirto, pitää aiemmin esitetystä pseudokoodista (rivi 7) poiketen pitää kirjaa parhaan arvon lisäksi myös sen tuottaneesta siirrosta.

Metodissa valitseSiirto ensimmäiseksi havaittu nopein voittoon johtava ja sen arvo ovat aina oikeaa tietoa, kuten muutenkin se siirto on paras (kyseisellä laskentasyvyydellä saatavan tiedon perusteella), jolle paras arvo kaikista siirroista on löydetty ensimmäiseksi. Sen jälkeen tutkittujen siirtojen arvo ei välttämättä ole oikea vaan ainoastaan yläraja siirron arvolle. Tämä ei ole ongelma, kun parhaasta siirrosta pidetään kirjaa niin, että tietoa päivitetään vain, kun löytyy entistä korkeamman arvon saava siirto. Jos siirto johtaa voittoon jollain laskentasyvyydellä, ei asia muutu siitä, että laskenta jatkuu syvemälle, koska sieltä voi löytyä vain hitaammin saavutettavia eli huonompia voittoja. Jos taas paraskin siirto johtaa tappioon, ei sekään muuksi muutu seuraavalla iteraatiolla. Laskentaa ei siis ole tarpeen jatkaa, jos on jo löydetty voittoon johtava siirto tai todettu, että kaikki siirrot johtavat tappioon. Iteraatio pitää kuitenkin laskea loppuun, vaikka jokin voitto löytyisi, sillä voi löytyä toinen siirto, joka tuottaa nopeamman voiton.

Riippuu pelistä kuinka usein transpositiotaulusta löytyy käyttökelpoinen tieto pelitilanteen arvosta. Shakissa voidaan päätyä samaan pelitilanteeseen useammin kuin ristinollassa, mielekkäillä siirtosarjoilla, joita laskennassa tutkitaan. Shakissa voi sama pelitilanne syntyä myös eri pituisilla siirtosarjoilla, koska nappuloita voidaan siirtää edestakaisin. Kaikissa peleissä saadaan merkittävää hyötyä siitä, että talletetaan tieto siirrosta, joka edellisellä kerralla tuotti parhaan arvon. Tätä siirtoa kannattaa kokeilla ensimmäiseksi, koska usein se edelleen johtaa parhaaseen tulokseen, tai ainakin se yleensä on hyvä siirto, eli se nostaa alfa-arvoa tai laskee beta-arvoa riippuen siitä onko maksimoijan vai minimoijan siirtovuoro, ja tämäkin tehostaa alfa-beta karsintaa. Muodostetaan siis kokeiltavien siirtojen lista tavalliseen tapaan, mutta jos pelitilanteelle löytyy aiemmin talletettu paras siirto, nostetaan se listan kärkeen. Kun siirrot on käyty läpi, talletetaan transpositiotauluun tieto siitä, mikä siirto tällä kerralla tuotti parhaan tuloksen tässä pelitilanteessa.

Transpositiotaulussa voidaan tallettaa tietoa valitseSiirto-metodin suorituksen ajan, tai siellä saatetaan säilyttää myös vanhempaa tietoa. Aiemmin laskettu tieto vanhenee yleensä nopeasti, kun peli etenee, ja suurin osa siitä ei ole enää lainkaan tarpeellista siksi, että kun kumpikin pelaaja on tehnyt yhden siirron lisää, ei pelissä enää päädytä samoihin tilanteisiin, joita tutkittiin ennen näitä siirtoja. Jos metodi valitseSiirto luo uuden transpositiotaulun aina, kun sinne tullaan, soveltuu transpositiotauluksi usein tavallinen ohjelmointikielen valmis hajautustaulu. Taulu on tällöin olemassa vain, kunnes päätetään mikä siirto tehdään, ja valitseSiirto palauttaa siirron. Mikäli myös vanhempaa tietoa säilytetään, pitää toimia hieman eri tavoin. Vanhaa tietoa täytyy jollain tavalla poistaa tai ylikirjoittaa, jotta taulu ei kasva niin, että muisti loppuu. Yksi tähän soveltuva menetelmä on Zobrist hashing. Tutkittavien pelitilanteiden määrä voi kasvaa liian suureksi myös, jos peli ja toteutus on sellainen, että siinä voiton tunnistaminen ja pelitilanteen heuristisen arvon laskeminen on hyvin nopeaa, eikä myöskään tehdä aikaa vievää laskentaa tutkittavien siirtojen karsimiseksi. Silloin tarvitaan transpositiotaulu, jonka koko ei kasva sallittua rajaa suuremmaksi, vaikka taulun elinkaari on vain valitseSiirto-metodin suoritus.

Kun transpositiotaulua säilytetään vain yksittäisen siirron valinnan ajan, mahtuu jokaista tutkittua pelitilannetta kuvaava tieto siis usein yhtä aikaa muistiin. Käytännössä ei kohtuullisessa ajassa pystytä käymään läpi monia kymmeniä miljoonia pelitilanteita, jos kokeiltavia siirtoja järjestetään

ja karsitaan älykkäästi. Vaikka tehokkaaseen karsintaan kuluu aikaa, ja pelitilanteita ehditään siten tutkia vähemmän, ehditään silti tutkia enemmän sellaisia pelitilanteita, jotka ovat merkityksellisiä parhaan siirron löytämiseksi. Tällöin kannattaa käyttää normaalia hajautustaulua ja siinä avaimia, jotka kuvaavat pelitilanteen yksiselitteisesti. Ohjelmointikielestä riippuen pelilaudan tilasta ja mahdollisista muista pelin tilan kuvaamiseen tarvittavista tiedoista (esim. shakin ohestalyönti- ja tornitussääntöjen vaatima tieto) muodostetaan merkkijono tai kokonaisluku avaimeksi. Voi olla, että avain on parempi muodostaa pelilaudan sijaan siirtosarjasta. Kun lähtötilanne on aina sama, kuvaa tehtyjen siirtojen sarja pelitilanteen yksilöivästi, ja näin voidaan saada lyhyempi avain, jonka laskeminen on nopeampaa ja edelleen hajautusarvon laskeminen avaimesta on nopeampaa. Edelliset siirrot välitetään tällöin minimaxin parametrina. Vaikka siirtosarja kuvaa pelitilanteen yksilöivästi, on tällöin olemassa useampi siirtosarja, joka kuvaa saman pelitilanteen, eikä välimuistista saada täyttä hyötyä. Siirtosarjaa pitää ennen avaimen muodostamista muokata niin, että samaan pelitilanteeseen johtavista sarjoista syntyy sama avain. Esim. ristinollassa voidaan yksittäistä siirtoa kuvata arvolla (x*laudan koko + y)*2, mihin lisätään 1, jos kyseessä on oma siirto. Kun tällaiset arvot järjestetään suuruusjärjestykseen, saadaan sama sarja, vaikka samat siirrot olisi tehty eri järjestyksessä. Hajautustauluun talletettava arvo on olio tai monikko, joka sisältää tarvittavat tiedot.