TiraLabra – Toteutusdokumentti

Juha Vesterinen

22.10.2014

Ohjelman yleisrakenne

Pelilogiikka, tekoäly ja käyttöliittymä muodostavat kukin oman selkeän kokonaisuutensa ohjelman sisällä, joten vastaavat luokat on jaettu omiin paketteihinsa.

Pelilogiikka

Pelilogiikkana toimiva Ristinolla -luokka on hyvin yksinkertainen: pelilautana toimii 2d-taulukko, johon pelimerkit talletetaan. Peliä edistetään kutsumalla pelaaVuoro(x, y) -metodia, joka lisää vuorossa olevan pelaajan pelimerkin parametrina saatuun koordinaattiin, tarkistaa mahdollisen voiton ja siirtää vuoron vastustajalle. Voiton tarkistus tapahtuu laskemalla tehdyn siirron muodostamien suorien pituudet vaaka-, pysty-, ja diagonaalilinjoilla. Vain tehdystä siirrosta muodostuvat suorat lasketaan, koko pelilautaa ei ole tarpeen tarkistaa jokaisen siirron jälkeen. Voitontarkistus on jokseenkin triviaali laskurisilmukka ja sen aikavaativuus on O(n), jossa n on peräkkäisten merkkien lukumäärä. Tilavaativuus on O(1). Myös muut Ristinollan metodit ovat hyvin triviaaleja lisäyksiä ja tarkistuksia, ja niiden aika ja tilavaativuudet ovat O(1).

Tekoäly

Tietokoneen tekoäly perustuu Alpha-Beta karsintaan, joka on edistyneempi versio MinMax-algoritmista. Myös normaali MinMax-haku on mukana harjoitustyössä, mutta pääpaino on algoritmien samankaltaisuuden ja huomattavan suorituskykyeron takia Alpha-Beta karsinnassa. Molemmat algoritmit perustuvat eri pelitilanteiden arvojen minimointiin tai maksimointiin pelaajasta riippuen. Tässä työssä risti etsii pelipuusta aina mahdollisimman suurta arvoa ja nolla mahdollisimman pientä arvoa. Seuraavassa pseudokoodi AlphaBetaKarsinnan maxArvo() -metodista, minArvo() on täysin vastaava, mutta alphan maksimoinnin sijasta minimoidaan betaa, ja tehdään nollan siirtoja.

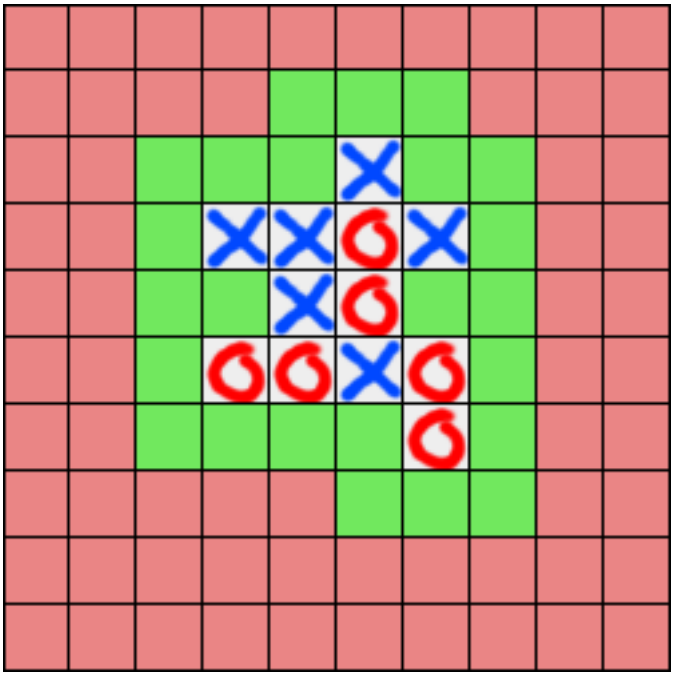
```
maxArvo(syvyys, alpha, beta, nollanSiirto)
      if (nollanSiirto voitti)
            return -OO
      else if (tasapeli)
            return 0
      else if (syvyys = 0)
            return heuristinen arvio pelitilanteesta

      do jokaiselle pelilaudan tyhjälle ruudulle
            if (ruutu on tyhjä JA aktiivinen)
```

```
                teeRistinSiirto() ja päivitäNaapuritAktiivisiksi()
                arvo = minArvo(syvyys-1, alpha, beta, ristinSiirto)
                poistaRistinSiirto() ja poistaNaapuritAktiivisista()
                if (arvo > alpha)
                      alpha = arvo
                      if (hakupuu 1. tasolla)
                            parasSiirto = ristinSiirto
                if (beta <= alpha)
                      break
    return alpha
```

MinMax-algoritmin aikavaativuus on O(b^d), jossa b tarkoittaa pelipuun haarautumisastetta ja d hakusyvyyttä. Tilavaativuus on O(d), sillä pelipuuta käydään läpi syvyyssuuntaisesti ja muistissa on siten aina vain yhden haaran pelitilanteet. Alpha-Beta karsinnan pahimman tapauksen aika- ja tilavaativuudet ovat samat kuin MinMaxilla, sillä hakupuun voi teoriassa käydä läpi sellaisessa järjestyksessä, ettei yhtään pelipuun haaraa onnistuta karsimaan. Parhaassa tapauksessa aikavaativuus on puolestaan O(b^d/2), joka vaatisi lapsisolmujen käymistä parhaassa mahdollisessa järjestyksessä. Vaativuusluokka pysyy siis samana, mutta karsinnan aiheuttama vaikutus vakiokertoimiin on huomattava. Joka tapauksessa eksponentiaalinen aikavaativuus ja pelipuun suuri haarautumisaste räjäyttää hakuajat melko pienelläkin pelipöydällä ja hakusyvyydellä hyvin nopeasti pilviin.

Pelipuun haarautumisasteen rajoittamiseksi implementoin pelialueen aktiivisia osia laskevan AktiivinenAlue -luokan. Se pitää kirjaa pelilaudalla olevien pelimerkkien välittömässä läheisyydessä olevista tyhjistä ruuduista ja merkkaa niitä lisäysten ja poistojen perusteella aktiivisiksi tai epäaktiivisiksi. Hyödynnän siis sitä empiiristä faktaa, että parhaat siirrot eivät koskaan löydy etäältä pelin ”toimintapesäkkeestä”, joten parhaan siirron hakuun ei koskaan sisällytetä epäaktiivisia ruutuja. Tämä karsii hakupuuta jokseenkin tehokkaasti, mutta aktiivinen alue toki kasvaa pelin edetessä ja haarautumisaste palaa siten lähemmäksi normaalia.



Kuva1: aktiiviset ruudut vihreällä, epäaktiiviset punaisella

Koska pelipuu kasvaa niin rajusti, täytyy haku tyypillisesti keskeyttää tietylle syvyydelle. Käytän harjoitustyössä yleisesti hakusyvyyttä 4, sillä se toimii lähes aina nopeasti ja haastaa ihmisen tarpeeksi älykkäästi. Toisin sanoen tekoäly puolustaa itseään tappiolta ja hyökkää aggressiivisesti

tilaisuuden tullen. Hakusyvyys 5 toimii useimmiten myös riittävän nopeasti, mutta vaikeissa tilanteissa se voi pysähtyä miettimään jopa minuutiksi, mikä on ihmispelaajan kannalta varsin tylsää. Hakusyvyys 3 puolestaan on liian ”tyhmä”, eikä osaa puolustaa suhteellisen triviaalejakaan uhkia vastaan.

Keskeneräisellä pelitilanteella ei ole yhtä selkeää arvoa kuin päättyneellä pelillä, joten pelitilanteelle täytyy antaa jokin heuristinen arvio. MunEvaluoija -luokka laskee molempien pelaajien suorien pituudet ja antaa kokonaisarvion pelitilanteesta. Eri pituisten suorien arvot lasketaan kaavalla 4^suoran pituus, eli pitemmät suorat ovat huomattavasti arvokkaampia kuin lyhyet. Mitä suurempi positiivinen kokonaisarvo pelitilanteella on, sitä suotuisampi pelitilanne on ristille, ja mitä pienempi negatiivinen arvo, sitä suotuisampi nollalle. Plus ääretön tarkoittaa ristin voittoa ja miinus ääretön nollan voittoa. Kyseinen evaluointimenetelmä on osoittautunut varsin toimivaksi. Evaluointi suoritetaan yksinkertaisella laskurisilmukalla, jossa käydään läpi rivit, sarakkeet ja diagonaalit. Evaluoinnin aikavaativuus on O(n*m), missä n on pelilaudan leveys ja m pelilaudan korkeus. Tilavaativuus on O(1).

Käyttöliittymä

Ohjelman käynnistyessä luodaan Käyttöliittymä, joka toimii ohjelman pääikkunana. Alussa pääikkunassa näytetään Asetusruutu, jossa pelaaja valitsee asetukset peliin (ks. käyttöohje). Asetusten valintojen perusteella luodaan Peliruutu, joka asetetaan pääikkunaan Asetusruudun tilalle. Peliruudun lisäksi ruudulla voi olla tekoälyjä varten Infopaneelit, jossa näytetään tekoälyn hakuun liittyvää metriikkaa, kuten hakuaika ja evaluoitujen pelitilanteiden lukumäärä. Kontrolleri hoitaa hiiren klikkausten tulkinnan ja tekoälyn ajamisen, ja antaa tiedot pelilogiikalle. Tekoälyn etsintä laukaistaan ajastimen avulla aina 0,5s viiveellä edellisestä siirrosta, jotta peliin syntyy parempi rytmitys ja jotta ruutu keretään piirtää jokaisen siirron jälkeen. Ilman viivästettyä laukaisua ikkuna päivitettiin vain ihmispelaajan vuoroilla. Kontrolleri hoitaa myös hiiren kursorin vaihdon pelitilanteeseen sopivaksi. Ristin vuorolla näytetään kursorina ristin merkki, nollan vuorolla nollan merkki ja tekoälyn miettiessä siirtoa järjestelmän ”wait”-kursori. Kursori myös palautetaan normaaliksi kun se viedään pois peliruudulta.

Työn puutteet

Alpha-Beta karsinnan hakupuussa lapsisolmujen läpikäymisjärjestystä ei ole optimoitu. Mikäli hakua onnistuttaisiin ohjaamaan siten, että todennäköisesti parhaat lapsisolmut evaluoitaisiin aina ensimmäisenä, karsinta tehostuisi huomattavasti ja hakupuuta voitaisiin ulottaa syvemmälle. Eräs tapa toteuttaa tällainen optimointi olisi ollut suorittaa iteratiivinen haku, eli ensin tehdään haku syvyydellä 1, sitten 2, 3, 4 jne. Hakujen välissä lapsisolmut laitettaisiin järjestykseen sen hetkisen tiedon mukaan. Näpertelin kyllä hetken aikaa Negascout-algoritmin kanssa, mutta en saanut sitä kunnolla toimimaan ja debuggaus ilman kunnollista ymmärrystä algoritmin toiminasta ei houkutellut jatkamaan.

Käyttöliittymä jäi hieman vajavaiseksi, sillä ohjelman joutuu ajamaan uudestaan joka pelikerran jälkeen. Tekoälyn hakusyvyydelle ei myöskään ole valintaa, vaan se on aina 4. Kontrolleri-luokkakin on ehkä vähän liian täyteen tungettu, kursorinvaihdokset olisi esim. voinut hoitaa omassa luokassaan.

MinMax- ja AlphaBetaKarsinta-luokat ovat lähes identtisiä ja copy pastea on paljon. Homman olisi varmasti voinut hoitaa järkevämminkin, mutta en jaksanut juurikaan panostaa MinMaxiin sen hitauden vuoksi.

Lähteet

http://en.wikipedia.org/wiki/Alpha%E2%80%93beta_pruning

http://en.wikipedia.org/wiki/Minimax

http://wiki.bethanycrane.com/minimax

Johdatus tekoälyyn -kurssimoniste, Teemu Roos, syksy 2014,
https://www.cs.helsinki.fi/courses/582216/2014/s/k/1

Heuristics and Threat-Space-Search in Connect 5, Veselin Kulev and David Wu,
http://isites.harvard.edu/fs/docs/icb.topic707165.files/pdfs/Kulev_Wu.pdf