TKT20010 Aineopintojen harjoitustyö: Tietorakenteet ja algoritmit Kevät 2018 (Periodi 3)

Toteutusdokumentti 3D-Ristinolla

Tekijän nimi: Kari Ojala

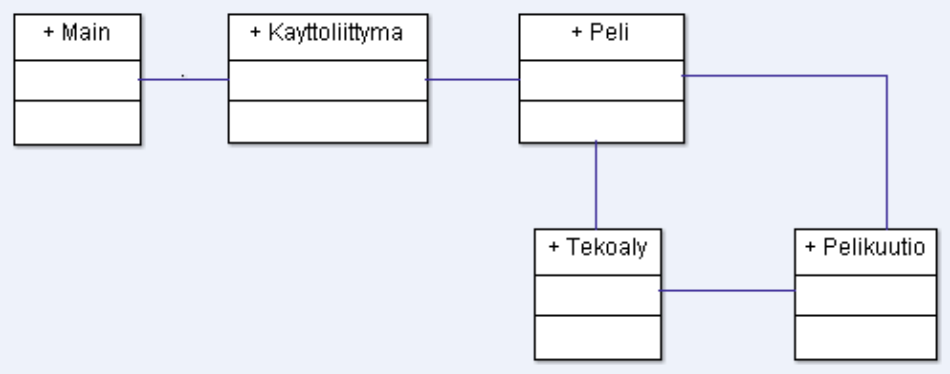
Päiväys: 9.3.2018

Toteutusdokumentti

1. Ohjelman yleisrakenne

Pääluokka Main käynnistää Käyttöliittymän. Käyttöliittymässä voidaan muokata pelin asetuksia ja tehdä valinta kahdesta tarjolla olevasta pelilajista. Käyttöliittymässä käynnistetään Peli. Luokka Peli alustaa luokan Pelikuutio, joka sisältää pelin vaatiman yksinkertaisen tietorakenteen ”pelistring”. Pelikuution metodeilla hoidetaan tärkeimmät pelitilanteisiin liittyvät kyselyt ja pelitilanteen päivitykset pelistringissä. Peli-luokka pyytää Käyttöliittymältä henkilöpelaajan uudet pelisiirrot ja Tekoäly-luokalta tekoälyn siirrot. Tekoäly-luokka etsii parhaan siirron. Pelin asetuksissa voidaan asettaa kaksi tekoälyä pelaamaan toisiaan vastaan, jolloin Luokalla Tekoaly on kaksi ilmentymää.

Peli päättyy kun jompikumpi osapuoli saa kolme merkkiä mille tahansa linjalle. 3x3x3-ristinollassa tasapeli on hyvin harvinainen. Tasapeli tarkoittaa tilannetta jossa voittajaa ei ole löytynyt ja tyhjiä ruutuja ei ole enää tarjolla. Pelin jälkeen voidaan käyttöliittymässä muuttaa pelin asetuksia ja käynnistää uusi Peli.

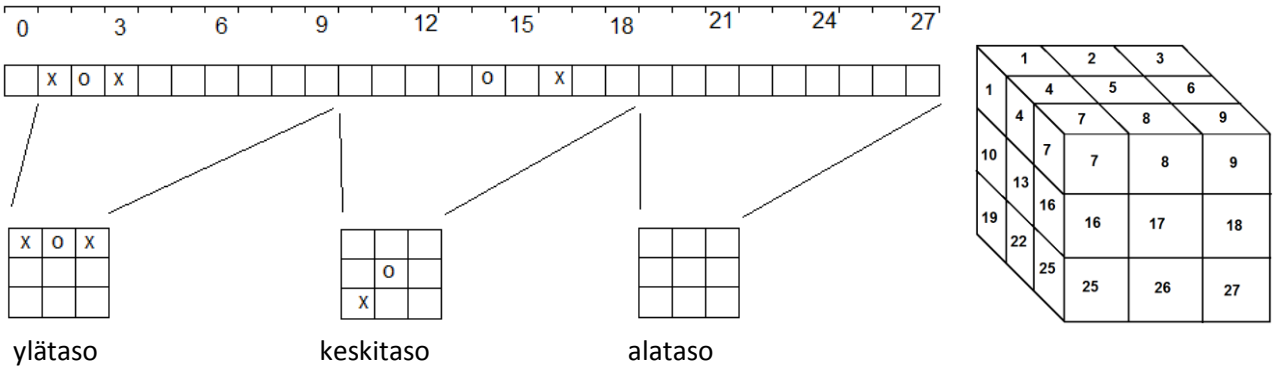


Kuva 1. Luokkarakenne

1.1. Perustietorakenne ”pelistring”

Määrittelydokumentin mukaan pelipuun perustietorakenne olisi joko 3x3x3-taulukko tai kolme 3x3-taulukkoa. Kun ohjelman laajennus 3-ulotteiseksi alkoi, näytti siltä, että näistä rakenteista tulisi käytännössä hankalia toteuttaa ja ohjelman testauksesta mutkikasta. Tilanne ratkaistiin yksinkertaisemmalla ”pelistring” tietorakenteella, jossa yksi ainoa 28 merkin pituinen jono tai merkkijonotaulukko kuvaa koko pelitilanteen. Erityisesti se mahdollistaa melko yksinkertaisen pelitietojen syötön Junit-testeissä.

Pelistring-taulukon merkki pst[0] ei ole käytössä. Esim. merkit pst[1..9] sisältävät ylimmän 3x3-ruudukon ja merkit pst[1..3] sen ylimmän rivin. Vastaavasti esim. merkki pst[14] viittaisi keskitason keskimmäiseen ruutuun ja merkki pst[16] keskitason vasemmassa alakulmassa olevaan ruutuun.



Kuva 2. Pelistring-taulukon numerointi ja jako 3x3-taulukoihin.

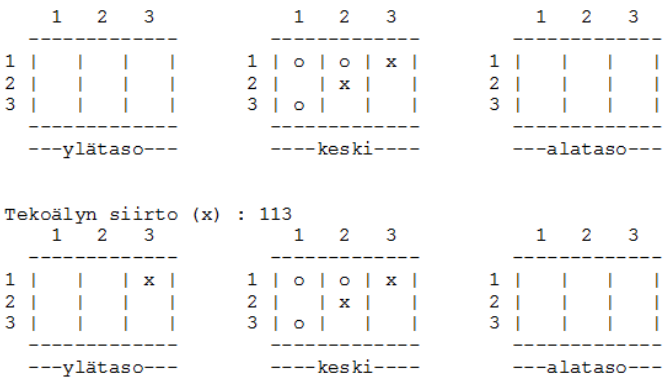
Pelistringistä voi melko yksinkertaisilla metodeilla muodostaa 2-ulotteisia 3x3-taulukoita. Pelikuutio tutkii näiden avulla mahdolliset voittolinjat. Pelitilanteen evaluointi tapahtuu kuitenkin Tekoäly-luokassa.

1.2. Heuristinen pelipuu

Lähtökohtana oli Tietorakenteet ja algoritmit kurssimonisteessa esitetty pelipuu ja Min-max-algoritmi[1]. Sen pohjalta 3x3x3-pelin aikavaativuus on luokkaa O(n!) eli pahimmillaan O(27!). Tämän takia etsittiin menetelmä arvioida pelitilanteita niin ettei koko pelipuuta käydä läpi. Aluksi harkittiin Johdatus tekoälyyn kurssilla kuvattua Alpha-beta-karsintaa[2], mutta sitä ei otettu käyttöön. Singaporen yliopiston sivuilta[3] löytyi selkeä kuvaus pelitilanteen heuristisesta arvioinnista 3x3 ristinollaa varten. Tätä sovellettiin 3x3x3 ristinollan pelitilanteen evaluointiin seuraavalla tavalla:

1. Käydään läpi kaikki mahdolliset pelitilanteet k siirtoa eteenpäin. (k on pelin ”vaativuustaso”)
2. Arvioidaan pelitilanteet laskemalla yhteen pisteet kaikilta pelilinjoilta k siirron päässä
    a. Linjalla on 3 omaa merkkiä +600 pistettä
    b. Linjalla on 2 omaa merkkiä ja tyhjä ruutu +25 pistettä
    c. Linjalla on 1 oma merkkiä ja 2 tyhjää ruutua +1 pistettä
    d. Vastaavat linjat vastustajan merkeillä: samat pisteet miinusmerkkisinä
3. Edetään pelipuussa syvemmälle kutsumalla vuorotellen metodeja uusNolla ja uusRisti. Palautetaan paras tai huonoin pisteluku kutsuvalle metodille mini-max-algoritmin mukaisesti.
4. Valitaan siirto sen mukaan, mikä tuotti parhaan pisteluvun.

Ylläkuvatusta menetelmästä löydettiin heikkous, josta esimerkki kuvassa alla.



Kuva 1. Tekoäly (x) ei estä (o)-pelaajan varmaa linjapaikkaa keskitason sarakkeessa 1.

Ilmeinen selitys tähän on se, että jos tekoäly tutkii pelitilanteet k siirron päässä, se ei huomaa jos jo yhden siirron päässä oli vastapuolen voittorivi. Vaikka tekoäly ”näkee” k siirron päässä vastapuolella olevan kolmen nollan linjan, se on mahdollisesti tuottanut myös itselleen k siirron jälkeen voittolinjan ja riittävästi pisteitä muista linjoista. Ratkaisu tähän ongelmaan oli yksinkertainen ja tehokas:

1. Tutkitaan ennen pelipuuhun menoa, voiko oma puoli voittaa yhdellä siirrolla. Jos sellainen siirto löytyy, palautetaan se kutsuvalle metodille välittömästi.
2. Sen jälkeen tutkitaan, voiko vastapuoli voittaa yhdellä siirrolla. Jos sellainen siirto löytyy, palautetaan se kutsuvalle metodille välittömästi.
3. Jos voittolinjoja ei löytynyt kohdissa 1 ja 2, evaluoidaan pelipuu k siirtoa eteenpäin.

Osoittautui että pelitilanteista jopa yli 50% on ”pakkotilanteita” kohtien 1 ja 2 tapaan. Tämä alentaa merkittävästi pelin aikavaativuutta jo muutaman ensimmäisen siirron jälkeen. Nanyang-yliopiston materiaali[3] kutsuu tätä lähestymistapaa sääntöpohjaiseksi strategiaksi. Pelissä sovellettiin kyseisen strategian kahta ensimmäistä sääntöä.

2. Saavutetut aika- ja tilavaativuudet (m.m. O-analyysit pseudokoodista)

Testidokumentissa on tarkempi kuvaus tuloksista. Tulosten mukaan heuristisen pelipuun aikavaativuus, jos peliä tutkitaan k siirtoa eteenpäin, on O(N*(N-1)*...*(N-k+1)). Tämä on murto-osa pelätystä O(N!) aikavaativuudesta. Lisäksi osoittautui, että jo k:n arvoilla 2, 3, 4 ja 5 saadaan hyvin tehokas tekoäly. Jos asetetaan k=3, tekoäly palauttaa siirtonsa käytännössä välittömästi. Jos k=5, siirron miettimisaika kasvaa kymmeniin sekunteihin. Tulokset kuitenkin osoittivat, että jo k:n arvolla 3 tekoäly antoi erittäin hyviä siirtoja.

Heuristisen pelipuun arviointi on ainoa merkittävä aikavaativuuteen vaikuttava tekijä.

Heuristisessa pelipuussa ei ole merkittäviä tilavaativuutta kasvattavia tekijöitä. Pelistring vie tilaa vain 28 merkkiä ja siitä kopioituu uusi versio aina kun edetään pelipuussa kerrosta syvemmälle. Pelipuu tuottaa kuitenkin vain yhden kopion joka tasolla, ja tasoja syntyy enintään k+1 kpl, missä k on enintään 5. Lisäksi ohjelmassa on muutamia Pelistringin kokoisia taulukoita mm. siirtovaihtoehtojen pisteytykseen.

3. Suorituskyky- ja O-analyysivertailu (mikäli työ vertailupainotteinen)

Työssä ei vertailtu mitään Javan valmista tietorakennetta omaan tietorakenteeseen. Vastaavana työnä voitaisiin pitää 3D-ristinollan pelistrategista analyysiä testidokumentissa. Peli kehitettiin asteittain, ensin 3x3 ristinolla, sitten 3-tasoinen 3x3 ristinolla ja vasta lopuksi kaksi aitoa 3x3x3 ristinollapeliä. 3-tasoinen 3x3-ristinolla oli hyödyllinen välivaihe, mutta osoittautui että sen pelistrategia poikkesi selvästi ”aidosta” 3D-ristinollasta.

4. Työn mahdolliset puutteet ja parannusehdotukset

Työssä ei ole varsinaisia puutteita ole siinä mielessä, että itse peli näyttää toimivan erinomaisesti.

Koodin laatuvaatimusdokumentin mukaan kaikki tulostaminen tulisi tapahtua käyttöliittymässä. Tästä on tehty kaksi poikkeusta: 1) Pelikuutio tulostaa itse oman tilansa eli ”pelistringin”. 2) Tekoäly tulostaa oman siirtonsa.

Työ sai alkunsa siitä, että tekijällä oli idea 4-ulotteisesta 4x4x4x4 risti-nollasta. Ensimmäisen aikavaativuus-analyysin tulos O(N!) säikäytti. Silti osoittautui, että käytännössä täydellisen hyvin 3D-ristinollaa pelaava tekoäly, jonka siirtoa ei tarvitse odottaa, oli toteutettavissa. Saadun kokemuksen valossa vaikuttaa siltä, että pelin laajentaminen 3-ulotteiseksi 4x4x4-ristinollaksi ei toisi merkittävää muutosta verrattuna 3x3x3-peliin. 4x4x4-pelissä 3x3x3-pelin ylivoimaista keskikuutiota vastaa 8 vahvaa keskikuutiota, muutoin peli toimii melko samalla tavalla. Tutkittavia linjoja tulee huomattavasti enemmän. 4x4x4-pelin tekoälyltä voitaisiin saada siirtoja nopeasti hyödyntämällä Alpha-beta-karsintaa eli etsimällä sellaisia pelihaaroja joita ei kannata käydä läpi. Tällöin pelipuun nykyinen läpikäynti yksinkertaisessa jälkijärjestyksessä pitäisi muuttaa eräänlaiseksi sisäjärjestykseksi alphan ja betan osalta. Heti kun solmun A jossakin tutkitussa alipuussa varmistuu, että solmu A ei voi tulla valituksi, voitaisiin solmun A loput alipuut jättää tutkimatta[3].

4x4x4x4-peli olisi mielenkiintoinen geometrisen ajattelun haaste, mutta käytännössä aika mekaanista uusien pelilinjojen ohjelmoimista.

5-linjaiset vähintään 3-ulotteiset pelit muodostavat jo todellisen älyllisen haasteen, koska näille ei ole vielä löydetty matemaattisesti todistettua voittavaa tai tasapelin tuottavaa strategiaa.

Ohjaajan ehdottamaa riippuvuuksien injektointia ei lähdetty toteuttamaan. Menetelmä olisi epäilemättä hyödyllinen, mutta se liittyy kurssiin Ohjelmistotuotanto, joten sitä ei tältä työltä vielä voi edellyttää.

Muita parannuskohteita.

1. Graafinen käyttöliittymä tietojen ja pelisiirtojen syöttämistä varten olisi hyvä tapa opetella GUI:n tekemistä. Tätä voi viedä pidemmällekin: pelikuutio voisi olla pyöriteltävä ja avattava kuten CAD-ohjelmissa. Oman pelimerkin voisi valita, ristin ja nollan sijasta voisi käyttää mitä tahansa valittavissa olevia kuvioita. Voittosiirtoon voisi lisätä jonkin efektin.
2. Pelissä on usein samanarvoisia siirtoja, varsinkin alussa. Pelin vaihtelevuus ja kiinnostavuus kasvaisi, jos tekoäly arpoisi siirron aina kun parhaita vaihtoehtoja olisi useita. Tämä on jossain mielessä käänteinen tavoite rinnakkaisten haarojen tutkimisen karsimiselle, mutta kenties arvokkaampi.
3. Pelitilanteen arviointia tehdään kahdessa eri metodissa ja kahdella eri tavalla, joskin hieman eri tarkoituksessa. Nämä testit voisi keskittää yhteen metodiin.
4. Luokka Tekoäly on varsin laaja. Toisaalta se hoitaa tiettyä omaa tehtäväänsä, ja sen toiminta on edelleen ymmärrettävissä ja hallittavissa. Muutamat metodit ovat melko pitkiä, mutta osa niistä on pitkiä esim. sen takia, että käydään systemaattisesti läpi suuri joukko pelilinjoja.
5. Muita parannusehdotuksia voisi kysyä pelin mahdollisilta käyttäjiltä.
6. Pelistä voisi tehdä mobiiliversion. Pikainen googlaus osoitti, että sellaisia on saatavilla.

5. Lähteet

1. Kivinen, J., Tietorakenteet ja algoritmit kurssimoniste, syksy 2017, ss. 394-400.
2. Roos, T., Johdatus tekoälyyn, 2016, ss. 8-12.
3. https://www.ntu.edu.sg/home/ehchua/programming/java/JavaGame_TicTacToe_AI.html#zz-1.3