Määrittelydokumentti **Shakkitekoäly, tira-harjoitustyö (HY CS), Markus Mustonen**

Varsinaiseen shakkipeli-logiikkakerrokseen kuuluu seuraavat osat:

1. pelimoottori
2. siirtojen generointi
3. arviointifunktio
4. hakualgoritmi (pelipuun läpikäynti)

Lisäksi toteutan yksinkertaisen graafisen käyttöliittymän, jossa nappuloita voi siirrellä hiirellä. Ohelmaa voi käyttää myös pelkkänä tekoälynä ilman käyttöliittymää niin, että syötteenä annetaan pelin kulku tähän mennessä pgn-tiedostona, joissa käytetään yleistä merkintätapa shakkipelin kulusta. Tällöin ohjelma palauttaa tekoälyn valitseman siirron, sekä tietoa pelin tilasta (esim. shakkimatti). Koko pelin kulku tarvitaan syötteenä, koska pelkkä nappuloiden sijoittelu laudalla ei riitä määrittämään pelin tilaa.

Tavoitteena on, että tekoäly voittaa viiden minuutin pelissä tekijänsä, sekä erään Timon laitokselta, jonka taitotaso on minua korkeampi (1700 chess.comissa).

Siirtojen generointi:

Siirtojen generointi on algoritmi, joka palauttaa listan kaikista mahdollisista siirroista nykyisessä pelitilanteessa vuorossa olevalle pelaajalle. Laudalla olevat nappulat läpikäydään ja jokaiselle kutsutaan siirtogenerointi funktiota riippuen niiden tyypistä. Mahdollisiin siirtoihin vaikkutaa nappulan sijainti laudalla sekä toiset laudalla olevat nappulat. Esimerkiksi keskellä lautaa olevalla hevosella voi olla kahdeksan mahdollista siirtoa. Siirtojen generointi palauttaa listan siirroista, jokaisesta lähtöruutu ja kohderuutu. Linnoitus esitetään kuninkaan siirtona. Tässä vaiheessa ei kuitenkaan otetan suorituskyvyn vuoksi huomioon sitä, että siirto on laiton, jos oman kuningas on siirron jälkeen uhattuna. Tämän tarkistaminen on pelimoottorin tehtävä.

Pelimoottori:

Pelimoottori pitää yllä tietoa pelin tilasta ja kulusta. Siihen kuuluu mm. pelilauta, pelitila, siirtovuoro ja siirron tekeminen. Vuorossa olevalta pelaajalta pyydetään siirtoa. Pelaajaa on rajapinta, joten tekoälyn voi helposti laittaa myös pelaamaan itseään vastaan. Moottori tarkistaa, että siirrot ovat laillisia, sekä muuttaa pelin tilaa, kun siirto tehdään. Laillisuustarkistuksen avulla käytöliittymä voi rajoittaa käyttäjää, ettei tämä tee laittomia siirtoja. Tarkoitus on huomioida kaikki shakin säännöt. Erityisesti nämä siirrot ovat kaikkein monimutkaisimpia:

1. sotilaan siirto aloitusruudusta kaksi ruutua eteenpäin
2. sotilaalla syönti vain viistoon
3. linnoitus
4. ohestalyönti (riippuu edellisestä siirrosta)
5. korotus.

Pelimoottorin tulisi myös tunnistaa shakkiuhkaus, mattitilanne, pattitilanne, sekä automaattinen tasapeli silloin, kun kummankaan pelaajan nappuloilla ei ole mahdollista tehdä mattia.

Lauta on 8x8 taulukko, joka koostuu Nappula-olioista. Tyhjää ruutua vastaa null. Hakupuuta läpikäydessään hakualgoritmi tekee siirtoja ja peruu niitä. Jotta siirtojen peruminen on mahdollista, täytyy tallentaa edelliset siirrot ja niihin liittyvät pelitilat. Koko laudasta ei kuitenkaan tehdä aina siirrettäessä uutta kopiota, koska se voisi laskea suorituskykyä merkittävästi. Sen sijaan jokaisesta tehdystä siirrosta tallennetaan seuraavat asiat:

1. lähtöruutu
2. kohderuutu
3. syöty nappula (tai null)
4. linnoitusmahdollisuudet tällä hetkellä (4x)
5. ohestalyöntimahdollisuus (riippuu edellisestä siirrosta)
6. oliko siirto korotussiirto (boolean)
7. oliko siirto ohestalyönti

Näiden tietojen perusteella voidaan siirtoa peruutettaessa selvitää aikaisempi pelitila täsmällisesti. Esimerkiksi jos siirto oli korotuussiirto, niin peruutettaessa siirtoa nappula korvataan sotilaalla. Jokaista tehtyä siirtoa vastaa Pelitila-olio. Shakkipeli-luokkassa on lista Pelitila-olioita, joka käsittää pelin kulun mukaan lukien nykinen tilanne.

Hakualgoritmi:

Hakualgoritmi yhdessä arviointifunktion kanssa on varsinainen tekoäly. Hakualgoritmin tehtävänä on selvittää, mikä on paras siirto tietokoneelle tämänhetkisessä pelitilanteessa. Pelitilanteen kaikista mahdollisista jatkoista muodostuu pelipuu. Yhdessä pelitilanteessa on vuorossa olevalla pelaajalla keskimäärin n. 35 siirtomahdollisuutta. Siten esimerkiksi seitsemän siirron syvyyteen ulottuvassa pelipuussa on lehtinä n. 64 miljardia asetelmaa laudalla. Hakualgoritmina toimiva yleisesti käytetty minmax-algoritmi olettaa, että molemmat pelaajat pelaavat mahdollisimman hyvin. Algoritmi käy pelipuun läpi rekursiivisesti, mutta puuta ei missään vaiheessa tallenneta. Minmax-algoritmia tehostetaan alfa-beta-karsinnalla. Sen avulla ohitetaan kokonaan sellaisia haaroja pelipuusta, jotka eivät voi vaikuttaa lopputulokseen. Valitsin alfa-beta karsinnan, koska se on helppo toteuttaa ja sillä saavutetaan huomattava etu: Voidaan hakea jopa kaksi kertaa syvemmälle pelipuussa samassa ajassa.
Minmax-algoritmi alfa-beta-karsinnalla pseudokoodina:

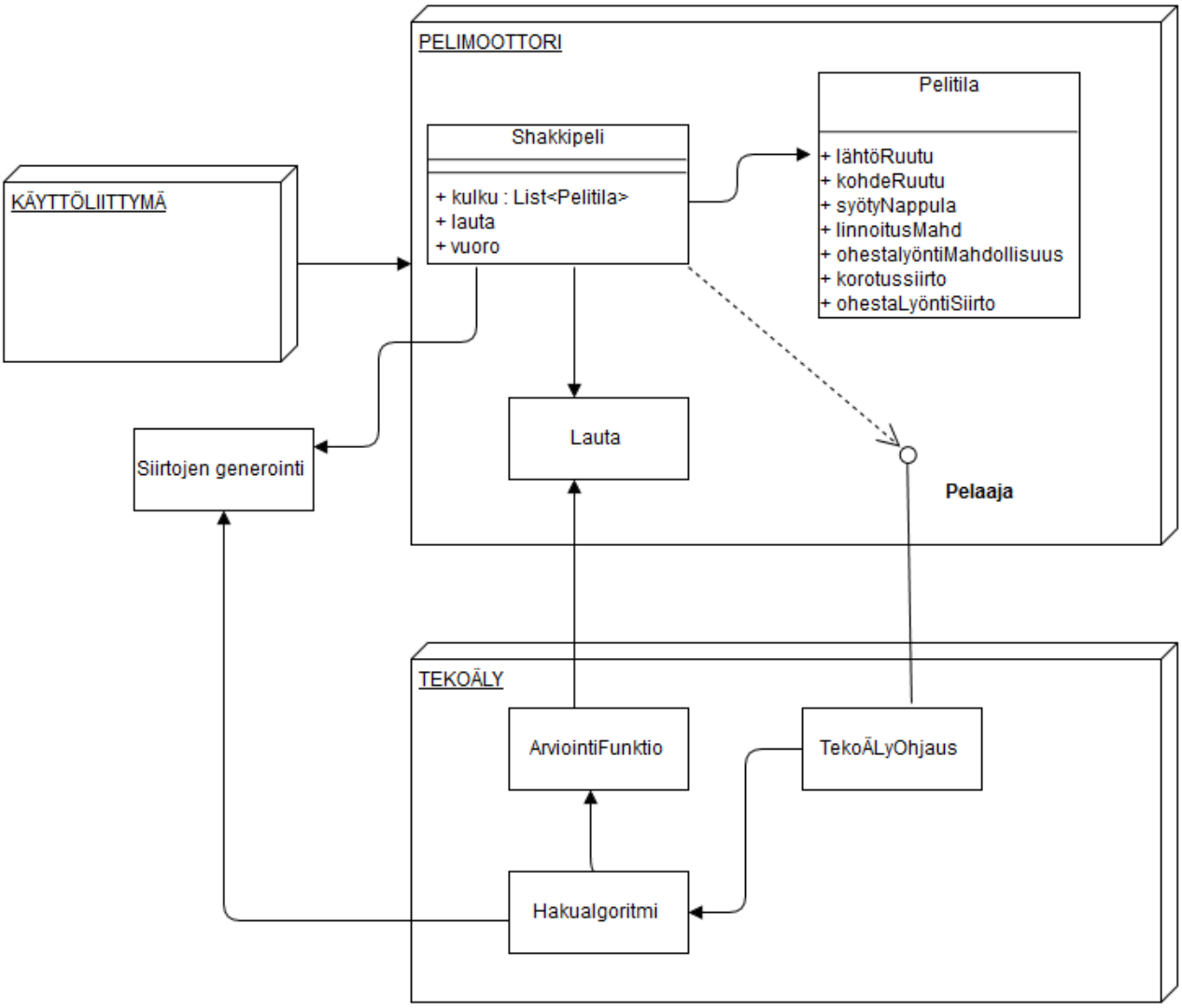
```
double haku(syvyys, alfa, beta)

          if(syvyys == 0)
                    return arviointiFunktio(lauta)

          for each laillisetSiirrot
                    teeSiirto()
                    arvo = haku(syvyys - 1, -beta, -alfa)
                    kumoaSiirto()

                    if(arvo >= beta)
                              return beta
                    if(arvo > alfa)
                              alfa = arvo

          return alfa
```

Algoritmi palauttaa siis arvion nykyiselle pelitilanteelle. Usein käytetään myös transpositiotaulukkoa, jossa tallennetaan arviot kaikista asetelmista tähän mennessä. Näitä voidaan käyttää sitten, kun päädytään samoihin asetelmiin toista kautta. En aio kuitenkaan käyttää transpositiotaulukkoa, koska se kasvattaa tilankäyttöä huomattavasti, eikä siitä saavutettava etu ole mielestäni merkittävä. AB-karsinta vähentää keskimääräistä haaroituskerrointa 35:stä kuuteen. Aikavaativuudeksi tulee siis O(6^syvyys). Lisäksi toteutuksessa täytyy ottaa huoimoon, että arviointifunktio, siirtojen generointi, sekä siirron tekeminen olisivat mahdollisimman tehokkaita suoritukseltaan, koska niitä kutsutaan hakualgoritmista. Ensimmäisessä iteraatiossa keskityn kuitenkin pelin toimintavarmuuteen, sääntöjen noudattamiseen ja testattavuuteen. Sen jälkeen optimoin hakualgoritmeja ja oheistoimintoja. Erityisesti hakualgoritmia voisi parantaa niin, että jos viimeisellä siirrolla ennen lehtiasetelmaa (syvyys == 0) on syöty vastustajan nappula, niin hakua jatketaan vielä yksi tai kaksi siirtoa syvemmälle. Tämän pitäisi parantaa tekoälyä huomattavasti, koska syövät siirrot ovat ratkaisevia.

Arviointifunktio

Arviointifunktio palauttaa liukuluvun, joka kuvaa nykyisen asetelman hyvyyttä. Jos valkoinen on voitolla, luku on positiivinen, jos taas musta on voitolla, luku on negatiivinen. Tärkeintä arvioinnissa on materiaali, eli kummalla on laskennallisesti paremmat nappulat jäljellä. Arvioinnin perusyksikkönä käytetään yhden sotilaan arvoa, joka on 1. Hevosen ja lähetin arvo on 3, tornin arvo on 5, kuningattaren yhdeksän ja kuninkaan esimerkiksi 1000. Tästä hakualgoritmi voi päätellä, että jos kuningas tulee syödyksi, niin peli on hävitty. Alkuasetelman arvo on 0. Ensimmäisessä iteraatiossa teen vain yksinkertaisen materiaalin laskevan arviointifunktion. Sen jälkeen funktiota kehitetään niin, että otetaan huomioon myös nappuloiden liikkuvuus, upseerien kehitys, sekä kuninkaan suojaus. Eri pelivaiheisiin tulee erilainen arviointifunktio. Tälläisiä vaiheta ovat alkupeli, keskipeli ja loppupeli.

Seuraava kaavio kuvaa osien välisiä riippuvuuksia:



Lähteet:

”Tekoälyn ohjelmointi shakkipeleille” – Aki Seppälä, pro-gradu, Jyväskylän yliopisto