Shakkipeli

Toteutusdokumentti

Tuukka Paukkunen
tuukka.paukkunen@cs.helsinki.fi

Aineopintojen harjoitustyö: Tietorakenteet ja algoritmit
6.9.2015
Helsingin yliopisto, tietojenkäsittelytieteen laitos

Sisällysluettelo

1 Ohjelman yleisrakenne

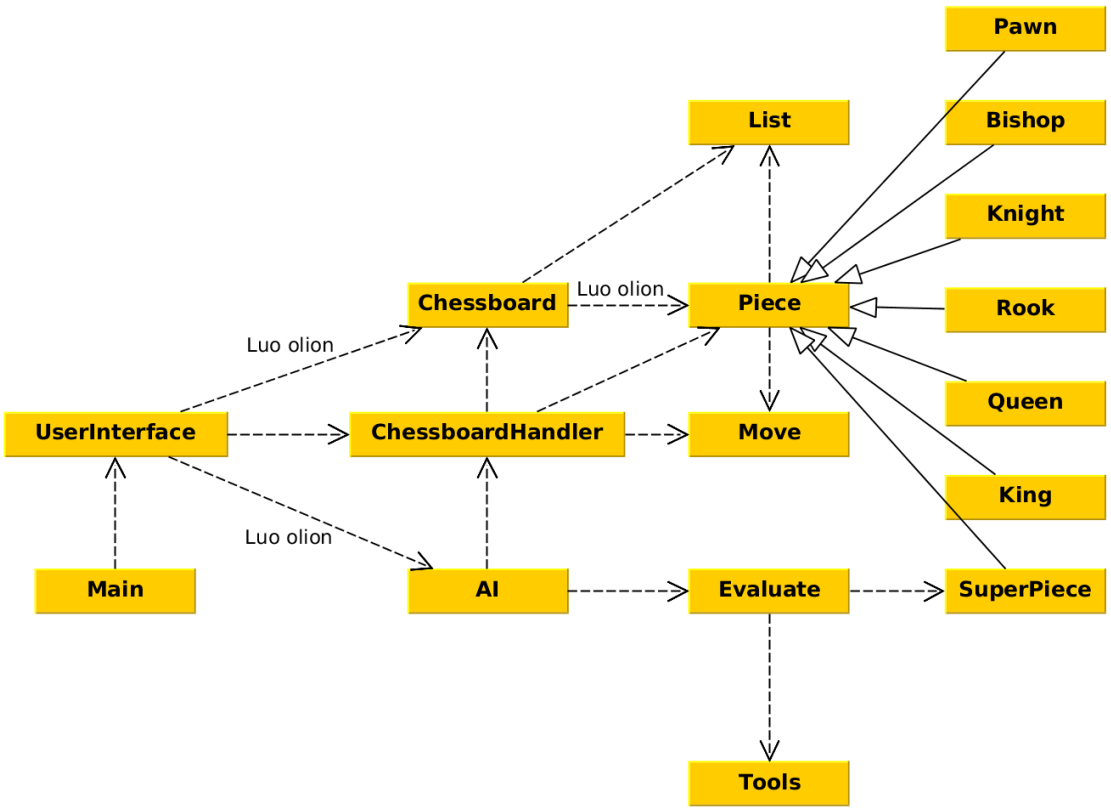
Sovellus koostuu seuraavista luokista:

Pakkaus	**Luokka**	**Staattinen**	**Luokan käyttötarkoitus**
Main	Main	Kyllä	Luokasta käynnistetään sovellus. Luokka kutsuu UserInterface-luokkaa, joka käynnistää sovelluksen tarvitsemat oliot.
Chessboard	Chessboard Handler	Kyllä	Hallinnoi shakkilautaa, eli Chessboard-oliota shakin sääntöjen puitteissa. Tarkistaa mm. Piece-olioiden avulla, että tehtävät siirrot ovat shakin sääntöjen mukaiset, sekä sisältää metodit shakki- ja shakkimatti-tilanteiden tarkistamiseen.
Chessboard	Chessboard	Ei	Shakkilauta-olio. Sisältää pelitilanteen nappuloineen. Sisältää kaksiulotteisen Piece-taulukon, valkoiset ja mustat nappulat sisältävät ArrayListit.
Chessboard.pieces	Piece	Ei	Nappuloiden abstrakti parent-luokka.
Chessboard.pieces	Pawn	Ei	Piece-luokasta periytyvä Pawn-luokka, joka sisältää tiedon, miten sotilas käyttäytyy pelilaudalla shakin sääntöjen mukaisesti.
Chessboard.pieces	Knight	Ei	Piece-luokasta periytyvä Knight-luokka, joka sisältää tiedon, miten hevonen käyttäytyy pelilaudalla shakin sääntöjen mukaisesti.
Chessboard.pieces	Bishop	Ei	Piece-luokasta periytyvä Bishop-luokka, joka sisältää tiedon, miten lähetti käyttäytyy pelilaudalla shakin sääntöjen mukaisesti.
Chessboard.pieces	Rook	Ei	Piece-luokasta periytyvä Rook-luokka, joka sisältää tiedon, miten torni käyttäytyy pelilaudalla shakin sääntöjen mukaisesti.
Chessboard.pieces	Queen	Ei	Piece-luokasta periytyvä Queen-luokka, joka sisältää tiedon, miten kuningatar käyttäytyy pelilaudalla shakin sääntöjen mukaisesti.
Chessboard.pieces	King	Ei	Piece-luokasta periytyvä King-luokka, joka sisältää tiedon, miten kuningas käyttäytyy pelilaudalla shakin sääntöjen mukaisesti.
Chessboard.pieces	SuperPiece	Ei	Piece-luokasta periytyvä SuperPiece-luokka.

Pakkaus	Luokka	Staattinen	Luokan käyttötarkoitus
			Tätä luokkaa ei käytetä varsinaisena pelinappulana. Tämä nappula sisältää kaikkien nappuloiden hyökkäyspatternit samassa nappulassa, joten tätä nappulaa käytetään pelitilanteen arvioinnissa sen selvittämiseen, suojaako jokin oma nappula toista nappulaa, tai onko jokin vastustajan nappula hyökkäämässä.
AI	AI	Ei	Luokka sisältää minimax-metodin alpha-beta -karsinnalla, joka palauttaa tietokoneen seuraavan siirron.
AI	Evaluate	Kyllä	Arvioi pelilaudan pelitilanteen hyödyllisyyden tietokoneen näkökulmasta.
AI	Tools	Kyllä	Sisältää Evaluate-luokan käyttämiä sekalaisia työvälineitä, mm. satunnaislukugeneraattori.
AI	Move	Ei	Yhden siirtokomennon sisältävä seuokka. Sisältää alku- ja loppukoordinaatit sekä mahdollisesti syödyn nappulan.
DataStructures	List	Ei	Luokka piilottaa sisäänsä yksinkertaisen taulukon, johon taltioidaan esim. Piece-olioita tai Move-olioita.
UI	UserInterface	Kyllä	Käyttöliittymän piirtävä luokka.

Sovelluksen toimintaan voi tutustua tarkemmin javadoceissa:
http://tuukkapa.github.io/Shakki-TIRALabra2015/javadoc/code/

1.1 Luokkakaavio



Sovelluksen luokkakaavio, jossa on kuvattuna olennaisimmat luokat ja niiden väliset yhteydet. Nuoli katkoviivalla tarkoittaa, että luokka käyttää nuolen osoittamaa toista luokkaa, ellei kaaviossa ole toisin mainittu.

2 Saavutetut aika- ja tilavaativuudet

Aika- ja tilavaativuudet määrittelydokumentin mukaan tulivat olla seuraavat: aikavaativuus on O(b^(m/2) ja tilavaativuus on O(bm), jossa b on sallittujen siirtojen määrä (eli siirrettävissä olevien nappuloiden määrä) kussakin pelitilanteessa ja m on pelipuun korkeus (tarvittaessa rajattu tiettyyn korkeuteen).

Sovellus nykyisellään tekee kopion pelilaudasta jokaista pelipuun noodia varten. Tämä tapahtuu siis yllä olevan tilavaativuuden rajoittamissa puitteissa. Aikavaativuudessa sen sijaan on käytännössä muitakin kertoimia kuin sallittujen siirtojen määrä, sillä jokaisen sallitun siirron generoinnissa joudutaan käymään pelilaudan vastustajan nappulat läpi, ettei siirron jälkeen mikäään nappula ole shakkaamassa pelaajan kuningasta. Käytännössä siis tämän sovelluksen saavuttama aikavaativuus on O((b*p)^(m/2)), jossa p on vastustajan nappuloiden lukumäärä.

3 Työn puutteet ja parannusehdotukset

3.1 Nappulat ja niiden sallitut liikkeet

Tällä hetkellä sovellus selvittää nappuloiden sallitut liikkeet ainoastaan matematiikalla ja if-else -lausein.

Ehkäpä kehittyneempi ja elegantimpi tapa olisi käyttää tässä verkkoja. Tällöin kullakin nappulalla voisi olla sallittujen siirtojen muodostama verkko koko pelilaudan alueelta, jolloin nappulan sallitut siirrot voisi selvittää yksinkertaisesti selvittämällä verkon tietyn solmun vierussolmut. Tähän olisi yksinkertaista yhdistää se, ettei syödä omaa nappulaa tai että mahdollisesti syödään vastustajan nappula.

Ainoastaan sotilas aiheuttanee tästä poikkeuksen, koska sen sääntöjen mukaisia liikkeitä on hankala kuvata verkkona.

Ns. ”en passant” -syöntitapa sotilaille päätyi jäämään pois lopullisesta sovelluksesta, koska se olisi aiheuttanut liian selkeän poikkeuksen sovelluksen koko muuhun rakenteeseen. En passant -syönnissä siis syödään nappula, vaikkei nappulaa liikutetakaan syötävän nappulan ruutuun.

3.2 Pelilaudan esittäminen

Pelilauta olisi todennäköiessti tehokkaampaa esittää bitboardeina, jossa jokaista nappulatyyppiä edustaa yksi 64 bitin bittijono, jossa aktiivinen bitti (1) kuvaa, että ko. kohdassa on nappula ja 0 kuvaa, että ko. kohta on tyhjä.

Tällöin näistä bitboardeista voisi Javan bittioperaatioita käyttäen muodostaa tehokkaasti muita hyödyllisiä näkymiä, kuten shakkilaudan vapaat ruudut, shakkilaudan varatut ruudut, vastustajan varaamat ruudut ja niin edespäin, mitkä kaikki olisivat hyödyllisiä sallittuja liikkeitä generoitaessa [CPW15].

Sovelluksen tehokkuus voisi siis kasvaa ja tilan tarve pienentyä ehkä huomattavastikin, mikäli bitboardeja käytettäisiin.

3.3 Pelitilanteen arvioiniti

Pelitilanteen arviointialgoritmia voisi periaatteessa hioa loputtomiin. Nykyinen soveluksen algoritmi ottaa huomioon nappuloiden lukumäärän sekä hieman suuntaa antaen joitain painotuksia, missä päin pelilautaa nappuloiden kannattaa sijaita. Jos nappula on suojattu, ko. nappula saa pistebonuksen. Jos nappula on hyökkäyksen kohteena, nappula saa sakon pisteissä. Myös yksin jätetty nappula saa sakon. Näissä arvioissa ei kuitenkaan oteta tehokkuussyistä huomioon nappulan arvoa. Esim. tietokoneen mielestä on hyväksytty tilanne, jos vastustajan sotilas uhkaa kuningatarta, kunhan kuningatar on suojattu, vaikka tämä todellisuudessa on pelin kannalta epäedullinen tilanne.

Arvioinnista puuttuu myös mobiliteetin arviointi, eli kuinka monta sallittua siirtoa pelitilanteessa on mahdollista tehdä. Se olisi hidastanut jo ennestään hidasta sovellusta ennestään.

3.4 Tehokkuus

Pelipuun rakentamisen edetessä monista lasketuista arvoista voisi olla myöhemmin hyötyä. Esimerkiksi samanlaiselle pelitilanteelle todennäköisesti lasketaan pelitilanteen arvo toistuvasti eri siirtokerroilla. Mikäli pelitilanteiden pisteet olisivat talletettuna johonkin tehokkaasti haettavaan tietorakenteeseen (esim. binäärihakupuu), pelipuun generointi saattaisi nopeutua.

4 Lähteet

CPW15 Chess Programming Wiki, Bitboards. https://chessprogramming.wikispaces.com/Bitboards [27.8.2015]