Shakkipeli

Testausdokumentti

Tuukka Paukkunen
tuukka.paukkunen@cs.helsinki.fi

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

Sisällysluettelo

1 Testauksen laajuus

Erilaisia shakkilaudan pelitilanteita arvioidaan olevan 10^120 kappaletta, ja näiden välillä erilaisia siirtovaihtoehtoja n. 10^3 kpl [Wik15]. Tämän perusteella lienee selvää, että kattavan testitapauslistan muodostaminen on mahdotonta, joten tutkivalla testauksella sovelluksen toimiminnan varmistamisessa on olennainen osa.

Sovelluksen testauksessa on käytetty laajasti toki myös jUnitia. Testaus jUnitilla on keskittynyt etenkin siihen, että pelinappuloita voi siirtää shakkilaudalla ainoastaan shakin sääntöjen mukaisesti.

2 Testausaineisto

Testausaineisto koostuu 263 yksikkötestistä, jotka testaavat perusasiat mm. nappuloiden liikuttelusta, pelilaudan toiminnasta, pelitilanteen arvioinnista ja käyttäjän siirtokäskyjen vastaanotosta. Lista yksikkötesteistä löytyy jUnit-testien javadoceista: http://tuukkapa.github.io/Shakki-TIRALabra2015/javadoc/tests/

Tärkein rooli kuitenkin on tutkivalla testauksella, joka yksinkertaisesti tarkoittaa sitä, että peliä pelataan mahdollisimman monella erilaisella siirrolla ja samalla kaikkia nappuloita pyritään liikuttamaan kaikilla mahdollisilla tavoilla. Samalla kirjataan ylös, miten tietokone vastaa käyttäjä siirtoihin. Mikäli jotain sääntöjen vastaista tai poikkeavaa tapahtuu, tällöin otetaan ylös, minkälaisessa pelitilanteessa poikkeus tapahtuu, ja pyritään toistamaan tilanne esimerkiksi aloittaen peli suoraan poikkeuksen aiheuttavasta tilanteesta ja seuraamalla sovelluksen toimintaa rivi riviltä.

Pelipuun rakentamisen testaamisessa taas tietokone vastaan tietokone -peli on oivallinen apuväline.

Kuten kappaleessa 1 mainittiin, kattavan listan luominen käsin tehtävistä testitapauksista on mahdoton tehtävä.

3 Miten sovellus on testattu

Projektihakemiston test-hakemistossa on luokat jUnit-testausta varten sovellusta vastaavissa pakkauksissa. Yksikkötestit (jUnit) kattavat sovelluksen perustoiminnallisuudet, kuten nappuloiden sääntöjen mukaisen liikuttelun ja käyttäjän syöttämien komentojen tarkistamisen.

Käyttöliittymälle ei ole tehty jUnit-testejä, vaan käyttöliittymän oikeellisuus on todettu visuaalisesti vertaamalla ohjelman tulostusta määrittelydokumenttiin.

Tämän lisäksi sovellusta on testattu pelaamalla peliä tietokonetta vastaan sekä seuraamalla tarvittaessa sovelluksen toimintaa debug-moodissa etenemällä koodia rivi riviltä.

Minimax-algoritmin toimivuus, eli että sovellus palauttaa aina siirtokomennon ja että

siirtokomento on mahdollisimman järkevä, on testattu ennen kaikkea antamalla tietokoneen pelata itseään vastaan. Tällä tavoin sovelluksesta löydettiin viimeisetkin virheet, mitä jUnit-testein ei kyetty löytämään. Kun annetaan tietokoneen pelata itseään vastaan kymmeniä pelejä, erilaiset minimax-algoritmin virhetilanteet ja poikkeukset paljastuvat väistämättä.

Näissä testeissä sovelluksesta esimerkiksi löytyi virhe, jonka vuoksi kaikkia nappuloita ei käytykään minimax-algoritmissa läpi, vaan jotkin nappulat jäivät läpikäynnissä välistä. Tämä virhe paljastui tietokone vastaan tietokone -pelissä, kun tietokoneen mukaan pelitilanteeseen ei löytynyt enää yhtään laillista siirtoa, vaikka niitä ilmiselvästi oli. Tämä johtui siitä, että pelissä käytettävän List-olion alkiot eivät pysyneet toisiinsa nähden samassa järjestyksessä, vaan järjestys vaihtui listaa läpikäydessä, joka aiheutti sen, että jotkin listan alkioista jäi väliin kokonaan läpikäynnissä. Ongelma korjaantui muuttamalla List-luokkaa siten, että alkoiden järjestys listalla säilyy. Tämä virhe tuskin olisi löytynyt millään jUnit-testillä tai pelaamalla peliä käsin tietokonetta vastaan.

4 Testaustulokset

Kaikki 263 jUnit-testitapausta menevät läpi. Nämä varmistavat sovelluksen perustoiminnan. Tämän lisäksi useita pelejä on pelattu onnistuneesti läpi ilman virhetilanteita sekä ihminen vastaan tietokone -pelissä että tietokone vastaan tietokone -pelissä. Testien perusteella peli noudattaa shakin sääntöjä virheettömästi. Nappuloille sallitaan kaikki sääntöjen mukaiset siirrot (pois lukien ns. ”en passant” siirto, mikä on jätetty tietoisesti pois) ja sääntöjen vastaiset siirrot estetään.

Sovellus myös tekee tietokoneen siirtovuorolla niin järkevän siirron kuin nykyinen pelitilanteen arviointimetodi sallii. Toisinaan tietokone tekee pelipuun vaatimattoman koon ja pelitilanteen arviointimetodin yksinkertaisuuden vuoksi tyhmiä siirtoja, mutta tätä tuskin voi sanoa varsinaiseksi virheeksi.

5 Suorituskykytestaus

Oheisessa taulukossa näkyy sovelluksen suorituskyky erilaisilla pelipuun syvyyksillä ilman alpha-beta -karsintaa sekä alpha-beta -karsinnan kera.

Testi tehtiin antamalla tietokoneen suorittaa viisi siirtoa (väleissä toki oli ihmisen siirto). Siirroista mitattiin kaikkiin viiteen siirtoon kulunut yhteenlaskettu aika millisekunteina.

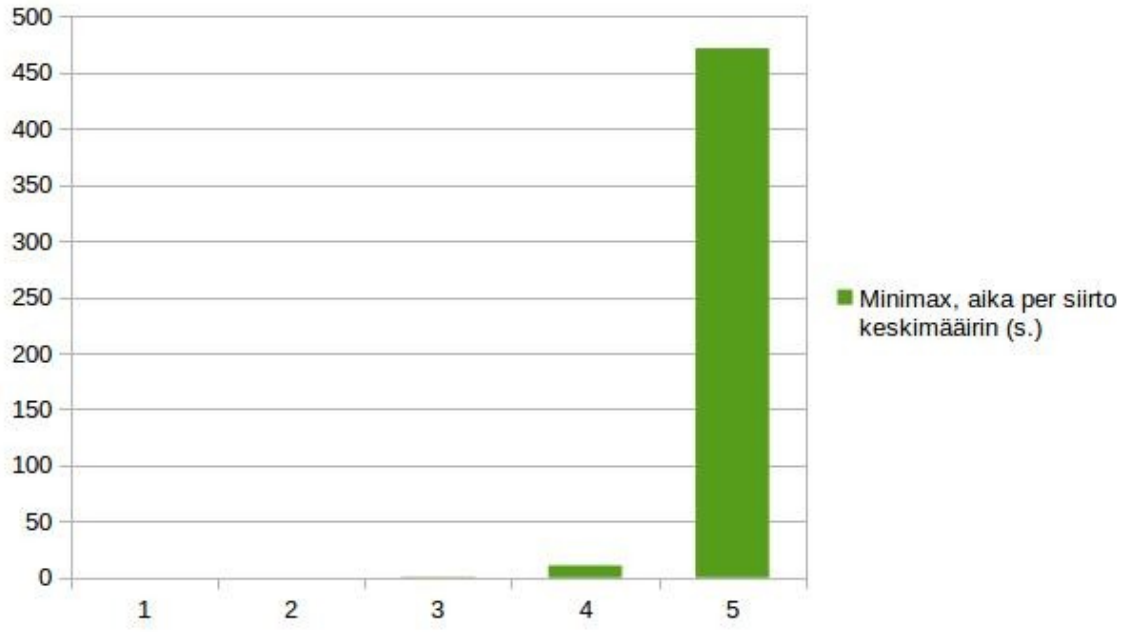
Mikäli yksi siirto kesti yli 10 minuuttia, testi keskeytettiin ja tulokseksi kirjattiin ”> 10 min”.

5.1 Suorituskykytestiympäristö

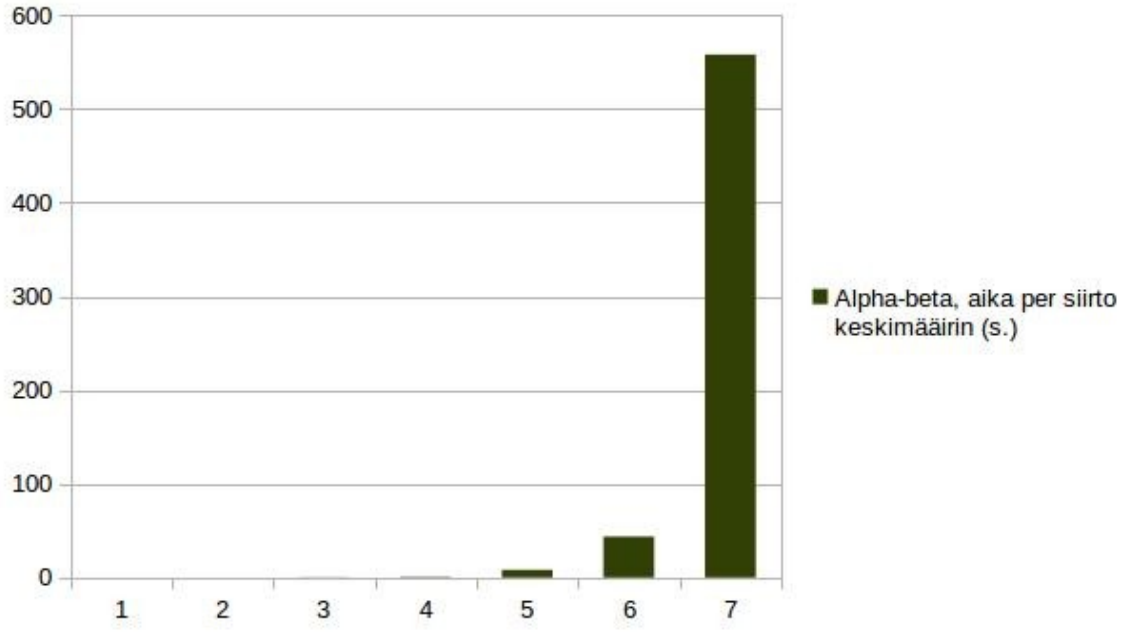
Prosessori: Intel i5 2500k (3,3GHz, 4 ydintä)
Muistia: 8 Gt

5.2 Suorituskykyvertailu

	Minimax		Alpha-beta	
Pelipuun syvyys	**5 siirtoa yhteensä (ms)**	**1 siirto, keskiarvo (ms)**	**5 siirtoa yhteensä (ms)**	**1 siirto, keskiarvo (ms)**
1	19	4	50	10
2	189	378	154	31
3	2891	578	1200	240
4	54077	10815	3483	697
5	2357390	471478	40611	8122
6	-	> 10 min	217623	43525
7	-	> 10 min	2789936	557987
8	-	> 10 min	-	> 10 min



Minimax-algoritmin suorituskyky ilman alpha-beta -karsintaa. X-akselilla pelipuun syvyys, y-akselilla siirron kesto sekunteina.



Minimax-algoritmin suorituskyky alpha-beta -karsinnan kera. X-akselilla pelipuun syvyys, y-akselilla siirron kesto sekunteina.

5.3 Johtopäätökset

Suorituskykytestivertailun tulokset puhtaan minimax-algoritmin ja alpha-beta-karsinnalla varustetun minimax-algoritmin välillä ovat selvät. Kun pelipuun syvyys on 3 tai yli, erot suorituskyvyssä alkavat olla huomattavia.

Pelipuun syvyydellä 3 alpha-beta -karsintaa käyttävä algoritmi suoriutuu siirrosta alle puolessa siitä ajasta, mitä puhdas minimax tarvitsee yhden siirron suoritukseen. Syvyydellä 5 ero on jo monikymmenkertainen.

Sovellus ei nykyisellään sovellu kovin hyvin syviin pelipuihin hitautensa vuoksi. Kun käytetään Alpha-beta -karsintaa, syvyydellä 5 ja 6 sovellus on vielä käytettävä, jolloin tietokone vastaa ajassa, jonka käyttäjä jaksaa vielä odottaa, mutta korkeammilla syvyyksillä sovellus on yksinkertaisesti liian hidas.

Sovelluksellani oli alkuperäinen tavoite olla käytettävä, kun pelipuun syvyys on 4, joten tämä tavoite on kuitenkin täytetty vaivatta.

6 Lähteet

Wik15 Shannon number, Wikipedian artikkeli, https://en.wikipedia.org/wiki/Shannon_number [27.8.2015]