Reittihaku ja Minimikeko

Toteutettuani minimikeon tein nopean tehokkuusvertailun javan PriorityQueueen. En tehnyt mitään teknisesti hienoa, ajoinpahan vain tuhansia reittihakuja eri kokoisilla, randomgeneroidulla kartoilla käyttäen eri tietorakenteita ja katsoin, kumpi on nopeampi. Alla hieno taulukko tuloksista.

Kartan koko	**Toistoja**	**Keko**	**PriorityQueue**	**Kekoetu**
3-5	100000	567ms	710ms	~25%
3-10	10000	112ms	147ms	~31%
10	10000	224ms	299ms	~33%
3-100	1000	1066ms	6566ms	~515%
100	1000	2993ms	27655ms	~824%
1000	1	1261ms	216292ms	~17052% (?!?)

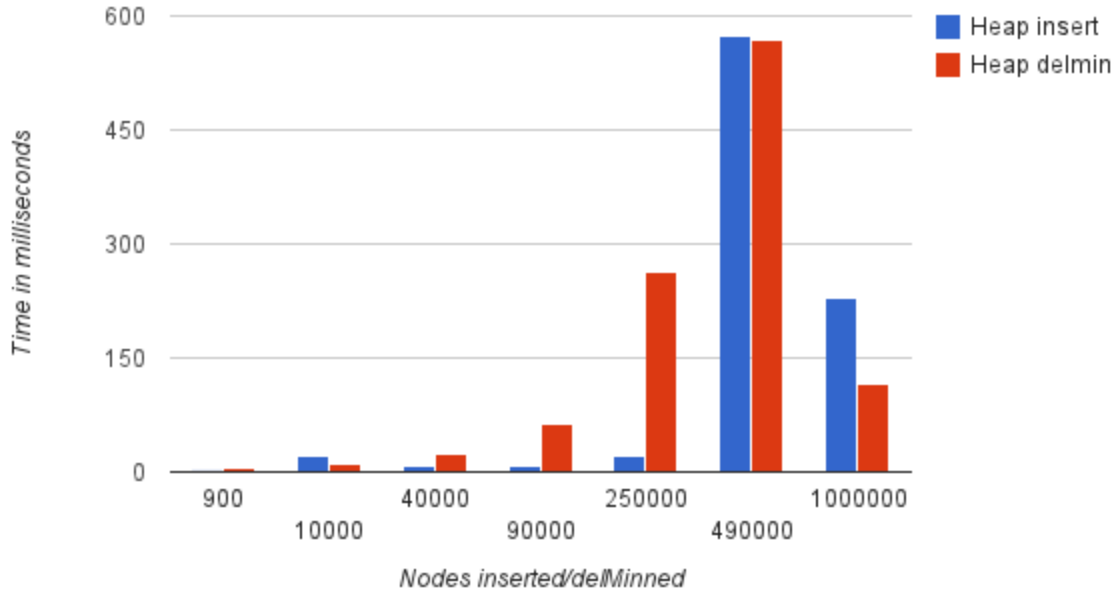
Tekemäni tehokkuustestit voi ajaa itse valitsemalla pelin alkuvalikosta Performance tests-napin. (Kartan suurin testikoko siinä 500 eikä 1000 koska PQfail.)

Kuten huomaamme, keko on selkeästi tehokkaampi pienilläkin kartoilla, mutta suuremmilla se on aivan ylivertainen. En tosin ole täysin varma, toimiiko algoritmi edes oikein jos kartat ovat kovin suuria. Ongelmia tulee viimeistään silloin, kun reittien pituus lähestyy miljardia (∞ on algoritmissani max-int/2…).

‘Hupaisasti’ algoritmin suhteellisesta tehokkuudesta on rajallisesti hyötyä, koska

1. tekoälyn hitaus ei liity reitinhakuun lainkaan ja joka tapauksessa
2. tehokkuudesta isoilla kartoilla (jos sellaisille nyt edes viitsisi pelata) ei ole mitään hyötyä, koska nopeusrajoitus rajaa tutkittavan osan karttaa neliöön, jonka sivujen pituudet ovat (reittiä hakevan yksikön nopeus * 2) + 1 eli luokkaa 10.

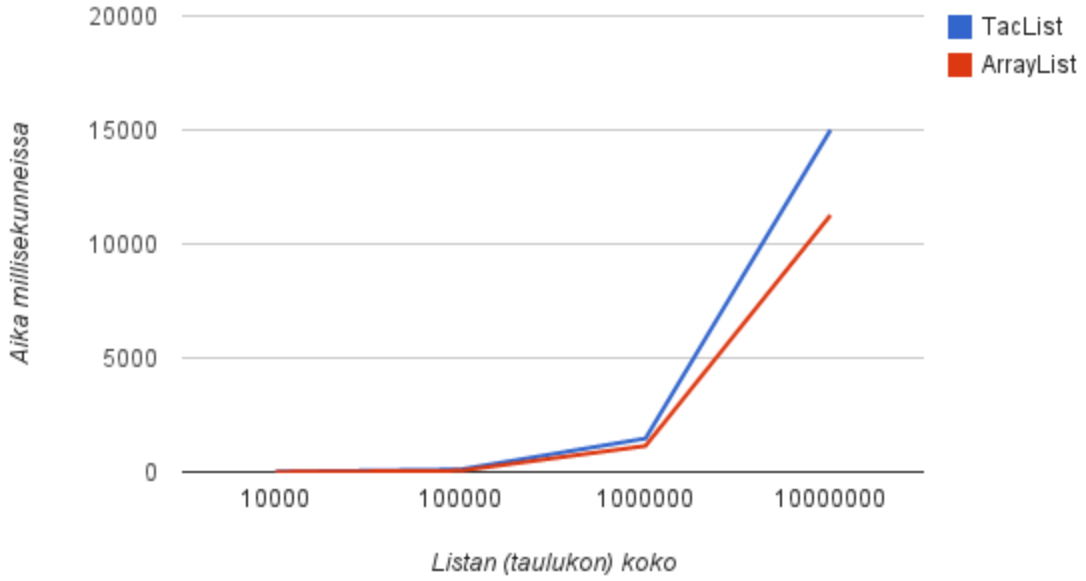
Tein lisäksi hieman tehokkuustestejä keon delMin- ja insert-operaatioille, mutta niiden tulokset ovat melko hämäriä, ehkä joistain javan hupaisuuksista johtuen.



Nuo siis yhdeltä ajolta. Huippujen sijainti vaihteli ajosta toiseen. Käytännössä aina kun päästiin satoihin tuhansiin solmuihin, alkoi jossain vaiheessa ilmestyä hupaisia tehonsyöntipiikkejä. Kymmenen miljoonan (ei kuvassa) kieppeillä jokin lakkasi lopullisesti toimimasta, ja aikaa kului pahimmillaan jo kymmeniä sekunteja. Tässä välissä kärsivällisyyteni loppui, varsinkin kun megasimppeli kekoalgoritmini sinänsä on *kohtuullisen varmasti* toimiva. Nämäkin testit kyllä löytyvät ohjelmasta.

Lista

Listalle lisääminen on ilmiselvästi vakioaikaista, koska taulukkoon vain tallennetaan size-indeksiin uusi arvo poislukien harvinainen tilanne, jossa taulukon kokoa pitää kasvattaa. (Pelissä tätä ei edes käytännössä tapahdu.) Poistamisen pitäisi olla $O(n)$, mitä testasin. Seuraavalla sivulla hiaano graafi tuloksista niin saadaan vähän väriä tännekin.



Vaaka-akselilla on listan koko ja pystyakselilla tuhannen ensimmäisen alkion poistamiseen (yksi kerrallaan) kuluva aika. Tulkinta toteutusdokumentissa.

Tekoäly

Tekoälyn ε-versio (kuvastaa sen älyn kokoa) on toteutettu min-max-sovelluksena, mikä tarkoittaa että se on älyttömän raskas. Ainakin tämänhetkinen tekoäly toimii hyvänä demonstraationa eksponentiaalisten algoritmien ongelmallisuudesta…

Tekoälyllä on neljä parametriä: aggressiivisuus, puolustavuus ja sattumanvaraisuus, jotka vaikuttavat sen tapaan arvioida eri toimintojen keskinäistä paremmuutta, sekä vuorot, joka vaikuttaa siihen, kuinka monta simuloitua kierrosta tekoäly pelaa kerrallaan.

Testasin tekoälyä pelauttamalla sitä eri parametreillä, vaihtelevilla yksikkömäärillä ja karttakoilla sekä toimintojen arvot määrittävää ValueLogic -luokkaa virittelemällä. AI toimii kohtalaisen loogisesti, ainakin sitten kun lopulta huomasin että olin sotkenut new Tekoäly():ssä parametrien järjestyksen. (Se mitä luulin aggressiivisuudeksi olikin sattumanvaraisuus ja se mitä luulin sattumanvaraisuudeksi olikin defense.)

Lopulta toteutin AIDueler-aliohjelman. Valitsemalla pelin aloitusvalikosta ‘AI tester’ terminaaliin ilmestyy tekstikäyttöliittymä, jolla voi syöttää

tekoälyille ja pelille eri arvoja ja pelauttaa kahta AI:ta toisiaan vastaan (ilman visualisointia). Nyt saatoin tutkia tekoälyn toimivuutta tehokkaammin. Alla joitakin havaintoja:

- Jos aggressiivisuus on liian matala osa peleistä jäi jonkinlaiseen looppiin jossa tekoälyt eivät hyökkää toisiaan vastaan.
    - Tätä sattuu sitä harvemmin, mitä enemmän yksiköitä pelaajilla on, mikä on melko intuitiivista.
    - Tätä ei tapahtunut kertaakaan, jos jompi kumpi tekoäly simuloi kolme vuoroa yhden asemasta.
- Pelit joissa tekoäly on säädetty aggressiiviseksi kestävät vähemmän aikaa (vuoroja) kuin pelit, joissa tekoäly on puolustavampi.
- Ajoin joitakin satoja pelejä pienellä (5 * 5) kentällä ja yksikkömäärällä (3) joissa toinen tekoäly simuloi kolme ja toinen vain yhden vuoron verran eteenpäin, ja monta vuoroa simuloiva todellakin voitti useammin…
    - ...Ei kuitenkaan mitenkään radikaalisti useammin, ero ehkä 10% enemmän voittoja.
    - Vain kahden vuoron simulointi isommalla kentällä ei antanut minkäänlaista havaittavaa etua yhden vuoron simulointiin.
    - Mutta: Jos pelimuotona oli ‘kill leader’, kolme vuoroa simuloiva AI voitti jopa kaksi kolmasosaa matseista pienellä kentällä ja yksikkömäärällä.
    - Samoin kävi kun pelikenttä oli isompi (14 * 14) mutta yksikkömäärä sama 3.
    - Tein sekopäätestin, jossa pelaajilla oli **massiiviset** 6 yksikköä 12 * 12 - kentällä *kill leader*-pelimuodossa, jonka ajoin 100 kertaa. Tätä koneparkani sitten raksutteli tunnin jos toisenkin (hyvää aikaa kirjoitella näitä raportteja siinä odotellessa...)
        - Testi kesti noin neljä ja puoli tuntia, mutta taas 3-vuoroäly voitti, joskin pienemmällä marginaalilla (54-46).
- Testasin myös tekoälyä, jonka sattumanvaraisuus oli 1000000 eli niin suuri, että se kumoaa kaiken muun arvonlaskentalogiikan.
    - Tällöin ei-sattumanvaraisesti pelaava tekoäly (1-vuoro) voitti noin yhdeksän peliä kymmenestä, ja odotetetusti enemmän kun yksiköiden määrä kasvati. (18 yksiköllä 199-1)
    - Pelin voittoehdoilla tai toisen tekoälyn agg/def-suhteella ei ollut vaikutusta lopputulokseen.
    - Jälleen def-korkea tekoäly pelasi varovaisemmin, sen 100 matsia randomia vastaan kesti melkein kaksi kertaa kauemmin.
- Lopuksi ajoin vielä pari settiä 1-vuoropelejä joissa kokeilin eri yhdistelmiä aggressiivista ja puolustavaa.
    - Tekoäly, jonka parametrit olivat matalia (10-100) voitti tekoälyn, jonka parametrit olivat korkeita (1000).
    - 1000000 aggressiivisuudessa aiheutti null pointer exceptionin yhdessä pelissä, oletettavasti siksi että max int aiheutti hankaluuksia...

        - ‘Negatiivinen aggressiivisuus’- pelit päättyivät odotetusti vuororajan täyttymiseen.
            - Negatiivinen puolustavuus päättyi hyvin lyhyisiin peleihin…
                - Ja sillä varustettu tekoäly hävisi kaksi peliä kolmesta.
- Poislukien miljoona-aggressiotapaus, peli ei jämähtänyt tai kaatunut kertaakaan.
- Tekoälyn arvolaskennassa on bugi (metodissa singleMoveAndAllAttacks), jonka poistaminen heikensi 3-vuorotekoälyn etua selkeästi.
    - On epäselvää, heikensikö bugin korjaaminen 3-vuoroa vai paransi 1-vuoroa. Aika ei kuitenkaan riittänyt tarkempaan tutkimukseen.
    - Epäkorjasin bugin, koska on hauskempaa, jos 3-vuoro pelaa paremmin..

Tekoälyn käytös vaihtelee sille annettujen parametrien puitteissa odotetulla tavalla. Lisäksi kolme vuoroa simuloiva tekoäly- joka on pienehköillä yksikkömäärillä vielä siedettävän nopea, pelaa ainakin jonkin verran paremmin kuin vähemmän vuoroja simuloiva. Täysin satunnaisesti tekemisensä arpova tekoäly murskataan. Kaiken kaikkiaan parin tuhannen automaattitestipelin jälkeen voin sanoa että tekoäly pelaa ainakin itseään vastaan suunnilleen niin kuin sen kuuluukin.

Ps. Jos jää kaipaamaan tekoälyn aikavaativuuksien graafista visualisointia, se sijaitsee itse pelissä tekoälyn “simulated turns”-nappien ja quick startin takana...

Muu testaus

Käsitestauksen ohella ohjelmassa on melko kattavat JUnit-testit. Projektin päähakemistosta dokumentointi-kansiosta löytyvät niistä generoidut cobertura- ja pit-raportit. Testien rivikattavuus tira-osiossa on yli 95% ja mutaatiokattavuuskin 70-90%, poikkeuksena ArtificialIntelligence ja SimulatedRound-luokat, joiden mutaatiotestaaminen on qamalaa. Niidenkin mutaatiokattavuus silti noin 60%.