Testausdokumentti **Anu N. 13.1.2014**

Labyrintti-ohjelman testaustilanne 13.1.2014 klo: 01:08

Käsin testaus ja yksikkötestit

Olen käyttänyt Labyrintti-ohjelman testauksessa pääasiassa JUnit -testejä, 'käsin testausta' pääohjelmametodista käsin sekä lopuksi myös käsin testausta graafisen käyttöliittymän kautta. Yksikkötestien kirjoittaminen on minulla edelleen hieman kesken (mikä ei tietysti ole kovin TDD-ihanteiden mukaista), ja olenkin huomannut erityisesti bugien ilmettyä ohjelman jo laajennuttua, kuinka hyödyllistä TDD ja jokaisen pienenkin koodipätkän huolellinen testaaminen on. Testauksessa on vielä parannettavaa myös esimerkiksi siinä, että yhdessä testissä tulee nyt monesti käytettyä useampaa kuin yhtä metodia, ja toisaalta jotkut testit ovat ehkä vähän turhiakin. Kaiken kaikkiaan JUnit -testien kirjoittaminen on minulle vielä melko uutta, joten järkevien testien toteuttamisessa on varmasti vielä paljon opittavaa, jotta testitapauksia osaa kirjoittaa kaikki mahdolliset poikkeustilanteet huomioon ottaen.

Esimerkiksi sellaisia metodeja, jotka vain tulostavat jotakin (esim. keossa olevat alkiot), en ole testannut, enkä myöskään yksityisiä metodeja, joita jokin julkinen metodi käyttää. Testien kirjoittaminen jäi osin kesken - jotkin metodit jäivät lopuksi testaamatta, ja monia testitapauksia olisi voinut vielä paljon parantaa.

Kaikki testit löytyvät Labyrintti-ohjelman *Test Packages → <default package>*- testipakkauksesta. Ne ovat luokkia, joiden nimessä on pääte "-Test" (esim. *AstarTest.java*).

Tällä hetkellä olen testannut luokkien metodeja seuraavasti:

AstarJaJPS

- rekonstruoiPolku()
- getPolku()
- searchOmallaKeollaJaJumpPointilla()
- suunta(Solmu nykyinen, Solmu edellinen)
- step(Solmu solmu, String suunta)

Astar

- rekonstruoiPolku()
- getPollku()
- searchOmallaKeolla(Labyrintti labyrintti, Solmu lahto, Solmu maali)
- searchJavanPriorityQueuella(Labyrintti labyrintti, Solmu lahto, Solmu maali)

Keko:

- Keko(int koko)
- lisaaKekoon(Solmu solmu)
- vanhempi(int i)
- vasen(int i)
- oikea(int i)
- poistaPienin()
- isEmpty()
- contains()
- haeIndeksi(Solmu s)
- onkoTaynna()

Labyrintti:

- getSolmu(int x, int y)
- getLahto()
- getMaali()
- getNaapurit(Solmu s)
- getNaapurit2(Solmu s)
- etaisyysValilla(Solmu nykyinen, Solmu naapuri)
- esteVasemmalla(Solmu s)

- esteOikealla(Solmu s)
- esteYlapuolella(Solmu s)
- esteAlapuolella(Solmu s)
-

Pino:

- push(Solmu s)
- pop()
- empty()
- full()

Ruutu:

- getXKoord()
- getYKoord()
- setEste()

Solmu:

- setMatkaAlkuun(int alku)
- setKokonaisKustannus(int kustannus)
- setEdellinen(Solmu edellinen)
- getX()
- getY()
- compareTo(Solmu toinen)

Suorituskykytestaus

Minun oli tarkoitus suunnitella, suorittaa ja visualisoida harjoitustyötä varten kattavat suorituskykytestit, mutta viimeisinä päivinä huomasin ohjelman ydinrakenteissa tähän asti piilevinä olleita* bugeja, joiden metsästämistä ja korjaamista pidin tärkeimpänä asiana. Koska algoritmit ja tietorakenteet eivät toimineetkaan joka kohdassa oikein, niin kuin olin osin jo ehtinyt kuvitella, ei algoritmien suoritusaikojen vertailu olisi ollut kovin informatiivista. Sen verran tein kuitenkin muutamia testejä pääohjelmasta käsin, että suunnilleen samoissa aikavaativuusluokissa liikuttiin omalla kekototeutuksellani (tavallisella A*:lla ja JPS:llä) sekä Javan PriorityQueuella.

* en ollut osannut löytää bugeja, eli testata algoritmeja ja tietorakenteita oikein ja tarpeeksi isoilla ja erilaisilla syötteillä