Testausdokumentti

Luokkien testaukset

Level:

Luokkaa on testattu unit testien lisäksi käsin. Ohjelma tulostaa Levelien toStringit joka kerta, josta näkyy, jos jotain on vialla. Olen myös vertaullut levelin toStringin tulostusta PNGExportterin png kuviin.

Tile (ja TileType):

Tile luokassa on lähinnä gettereitä ja settereitä, joita ei minun mielestä tarvitse testata. Tile luokan toimivuus näkyy muiden luokkien testauksessa.

MazeGenerator:

MazeGeneratorilla on unit testi, joka käy labyrintin läpi ja epäonnistuu, jos ruudussa ollaan jo käyty (eli labyrintissä on sykli). Lisäksi ohjelma tuottaa png kuvan generoidusta labyrintista, jonka aina tarkistan.

RoomGenerator:

RoomGeneratorille en ole tehnyt unit testejä. Luokan testauksen olen tehnyt kokonaan käsin (eli katsonut generoituja png kuvia). Luokan kehitys vaiheessa jouduin käyttämään debuggeria huonon generoinnin korjaamiseen. Huono generointi ilmeni png kuvissa (ja levelin toStringissä)

DungeonGenerator:

Sama kuin RoomGeneraattorilla eli aina katsonut generoidut png kuvat ja sen mukaan korjaillut luokkaa.

PNGExporter:

PNGExporterin toiminnan olen testannut täysin käsin. Luokka tuotaa png kuvia Level luokista. Joka kerta kun ajan ohjelman luokka tuottaa kuvat generoiduista tasoista. Jos jotain olisi vialla PNGExportterissa se näkyisi kuvissa.

Stack:

Unit testit.

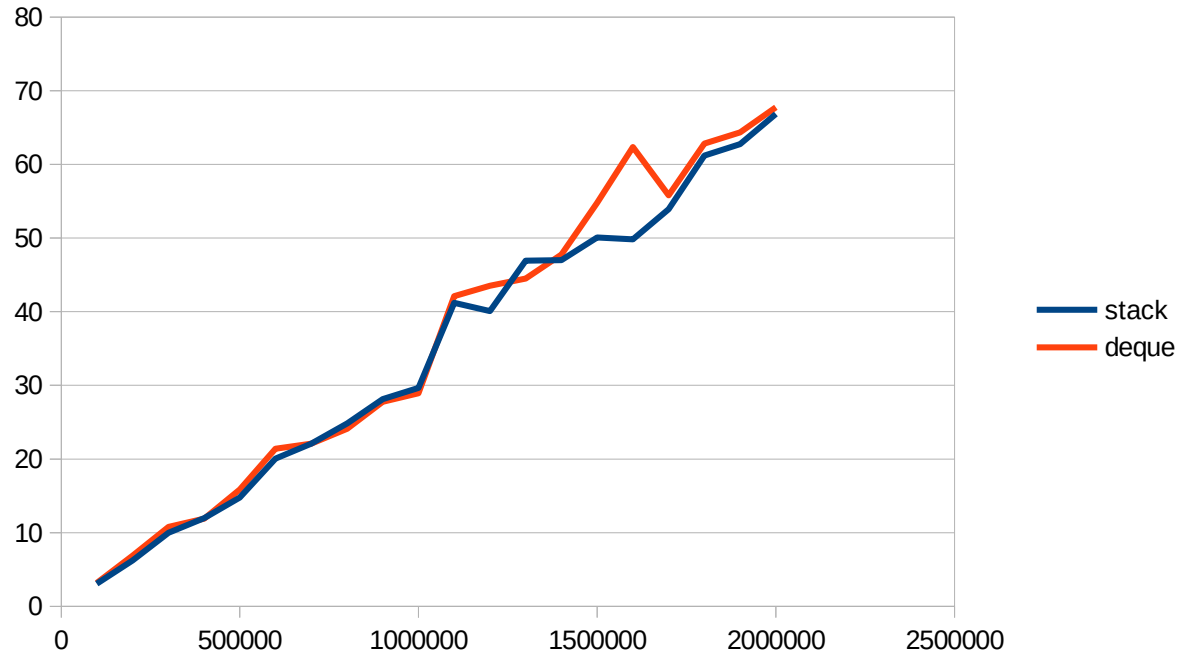
Vec2:

unit testit.

Suorituskykytestejä

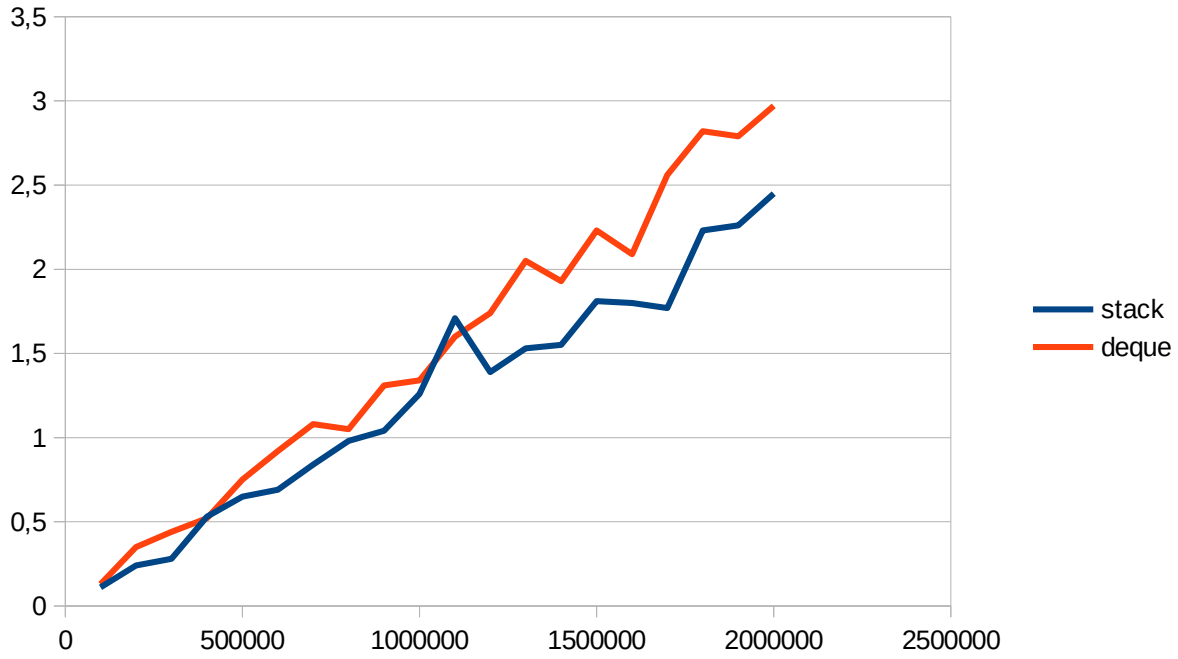
Suorituskykytestien luokat löytyvät projektin paketista **com.unknownpotato.dungeon**

Stack vs ArrayDeque Push vertailu:



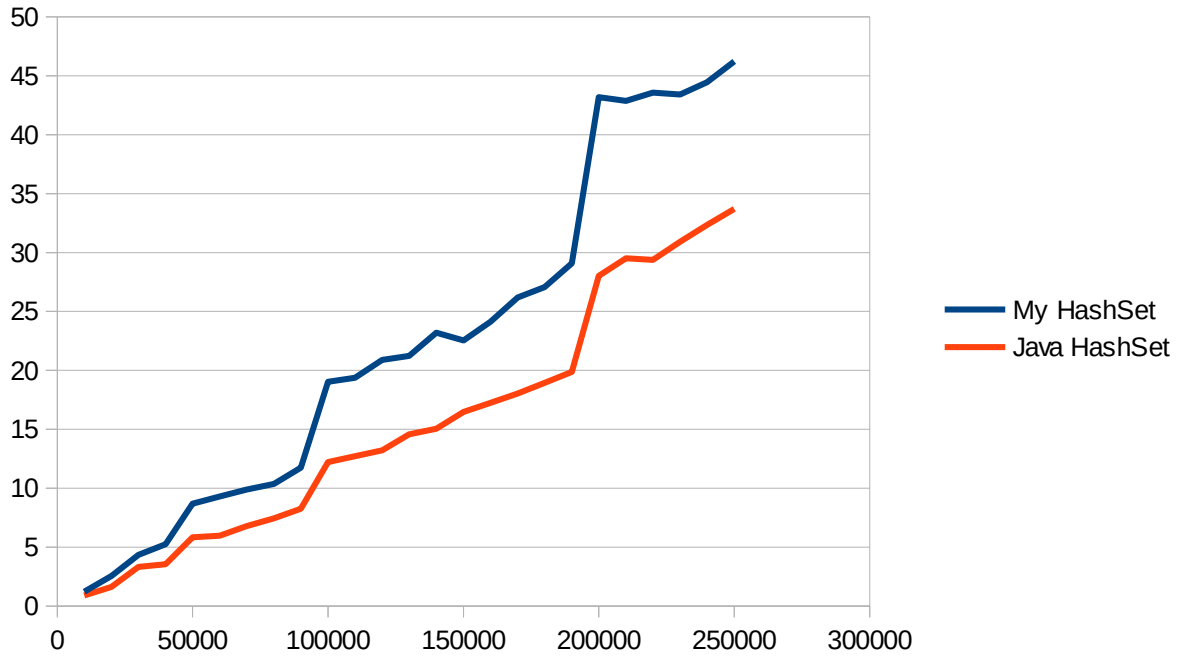
Testissä lisäsin tietorakenteeseen x määrän random generoituja kokonaislukuja ja otin aikaa kuinka kauan lisäämisessä meni. Lisäsin lisäsin lisättävien alkioden määrää 100000 ja otin uudestaan ajan. Yllä olevasta graaffista näkyy, että oman stack toteutuksen ja Javan ArrayDequen push operaatiot ovat suunnilleen yhtä tehokkaita. Suuremmat piikit ovat todenäköisesti mittausvirheitä.

Stack vs ArrayDeque Pop vertailu:



Testientulokset viittaisivat siihen, että oman stack toteutuksen pop operaatio olisi tehokkaampi kuin dequen. Testit tehtyäni huomasin kuitenkin, että en ollu ”lämmittänyt” JVM:ää toisin kuin pushin vertailussa. Tämä voi vaikuttaa tuloksiin. Kuitenkin molempien testien tulokset eivät olleet hirveän yllättäviä. Toteteutin pinon taulukkototeutuksella (linkitetyn toteutuksen sijasta). Javan arraydeque on nimensä mukaisesti myös taulukkototeutus joten samankaltainen tehokkuus ei ole yllättävää. Lisäksi koska deque toimii myös jonona sen täytyy tehdä joitakin lisä operaatioita jotka voivat selittää miksi oma stack toteutus toimii hieman paremmin.

Oma HashSet vs Java HashSet Add vertailu:

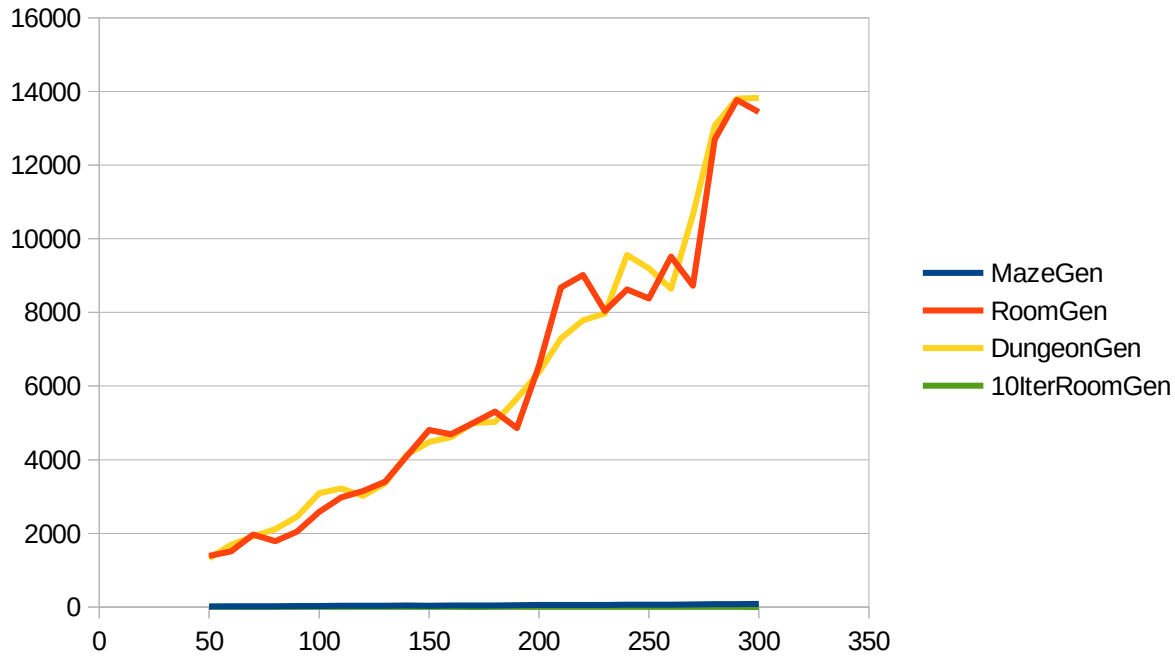


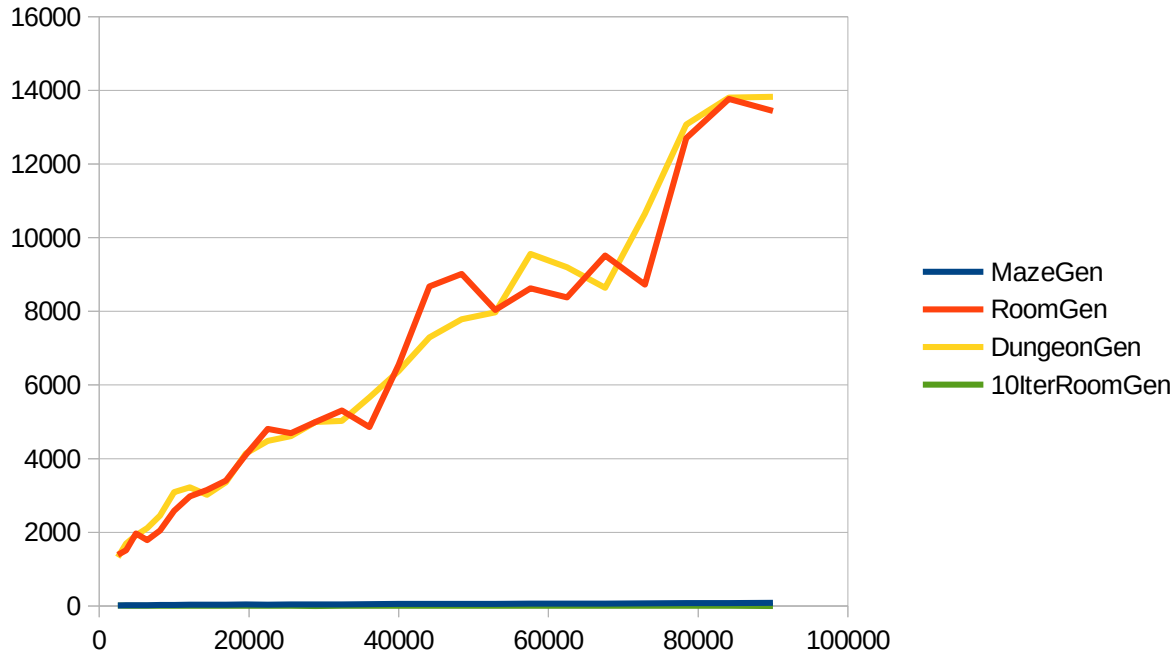
Kuten käyristä näkyy oma HashSet toteutukseni on epäoptimaalisempi (oman käyrä kasvaa

nopeampaa tahtia). Kuvista näkyy myös että toteutuksessani on samanlaisuuksia javan toteutukseen. Molemmissa käyrissä näkyy samoissa kohdissa ”hyppäykset” lisäysajoissa (100000 ja 200000 kohdilla). Hyppäykset johtuvat HashSetin sisäisen taulukon kasvattamisesta. Käyrien samankaltaisuudesta voidaan päätellä, että molempien toteutuksien kasvunopeuden funktio on samankaltainen mutta eri kertoimilla eli add operaatioiden aikavaativuus vaikuttaisi olevan sama.

Generaattorien vertailu:

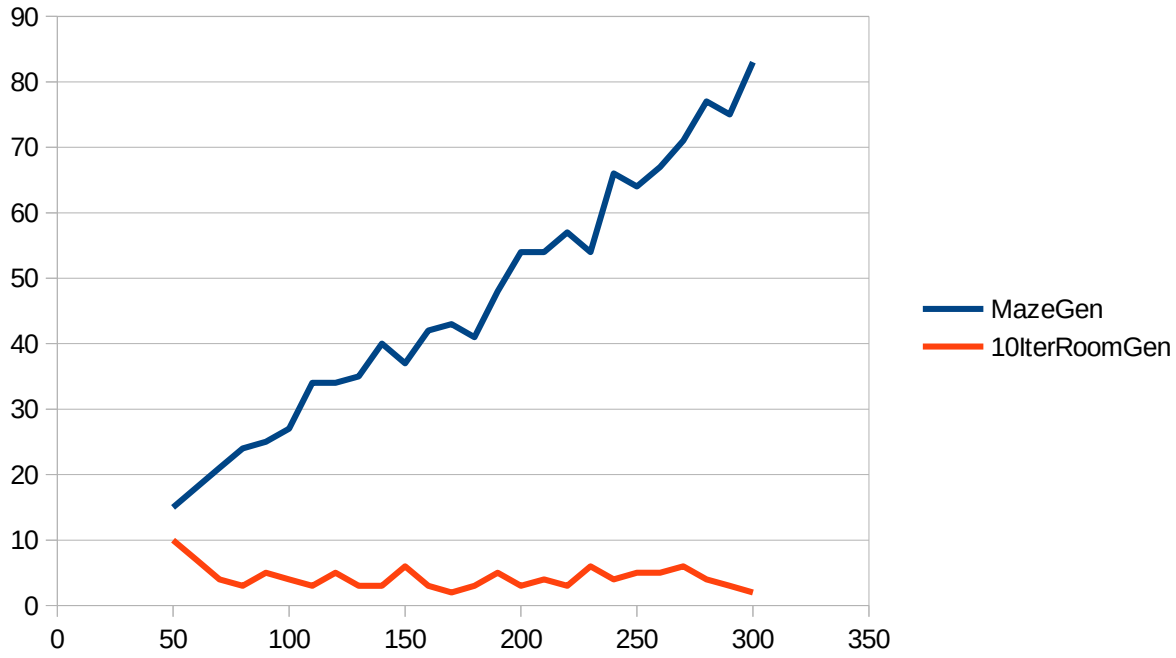
DungeonGenerator käyttää sisäisesti MazeGeneratoria ja RoomGeneratoria. Siksi olen laittanut MazeGeneratorin ja RoomGeneratorin parametrit samoiksi kuin DungeonGeneratorin sisäisten instanssien parametrit. Kasvatin testeissä levelin sivun kokoa ja mittasin kuinka kauan Generaattorin generointi kesti.





Ylemmässä kuvassa graaffi millisekuntteja sivun pituuteen nähden ja alemmassa millisekuntteja pinta-alaan nähden. Kuvassa MazeGenerator ja 10 iteraation RoomGenerator vievät huomattavasti vähemmän aikaa verrattuna DungeonGeneratoriin ja 10000000 iteraation RoomGenerator joten tässä vielä MazeGen ja 10IterRoomGen erillisessä graaffissa.

Suoritusajan kasvu sivun pituuteen nähden:



Suoritusajan kasvu pinta-alaan nähden: