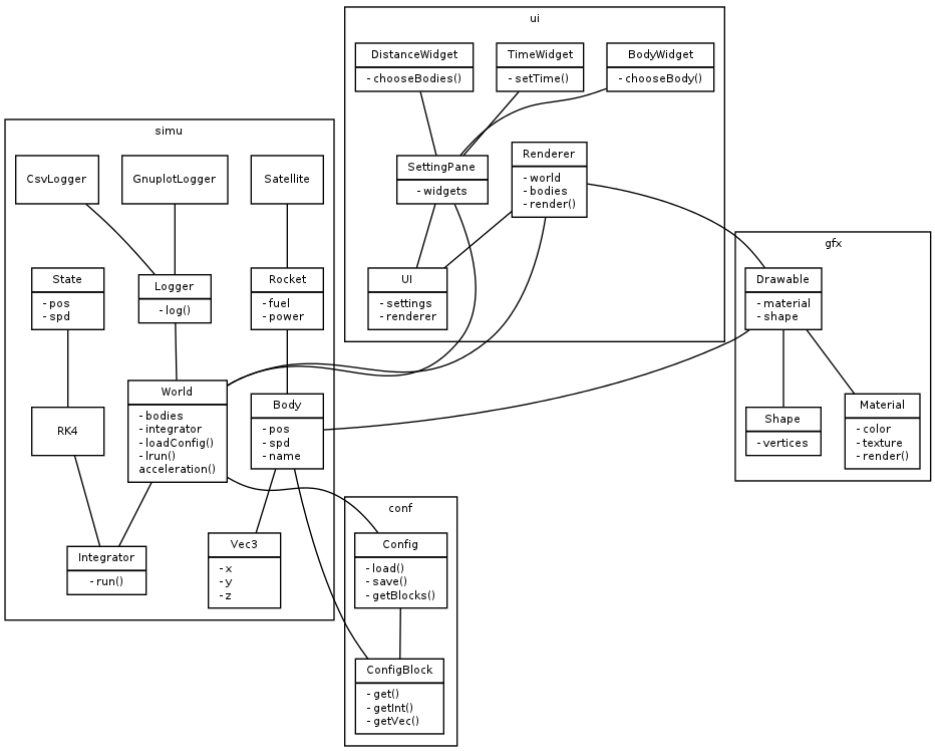
Aurinkokuntasimulaattori

T-106.1243 Ohjelmoinnin jatkokurssi L1
Projektin tekninen suunnitelma
Aihe 129

Konsta Hölttä
79149S
AUT 4. vk
konsta.holtta@aalto.fi
25.2.2011

1 Ohjelman rakennesuunnitelma



Kuva 1: Karkea luokkajako. Kaavio ei ole UML:ää eikä mitään muutakaan standardia muotoa. Viivat kuvaavat jonkinlaista interaktiota luokkien välillä.

Koodihierarkia hajautuu neljään erilliseen pääkategoriaan, joihin on suunniteltu ohjelman pääkomponentit. Erittäin karkea luokkajako on esitetty kuvassa 1. Luokkien väliset riippuvuudet ja varsinkin käyttöliittymän inputhandlerit tulevat vielä elämään merkittävästi.

- Konfiguraatiotiedostojen hallinta: geneerinen konffijärjestelmä, joka ei tiedä mihin sitä käytetään.
- Simulaatio integrointeineen: kappaleet jotka sisältävät mm. paikan, nopeuden ja massan; maailma, joka sisältää kappaleet, sekä oheisluokkia mm. integrointiin ja lokittamiseen.
- Käyttöliittymä ja dialogit: käli on pääasiassa yksi ikkuna, jossa on modulaarisesti eri asetuspaneeleita sekä 3d-canvas erillisinä luokkina.
- Renderöinti ja sen käyttöliittymä: renderöijä kyselee kappaleiden tie-

dot maailmalta ja piirtää ne näytölle, sekä käsittelee mm. näppäimistösyötteet joilla näkymässä voi liikkua.

1.1 Konfiguraatiotiedostot

Maailmankaikkeuksien rakenne kuvataan tiedostoissa, sekä ohjelman ajon aikana kaikkeuden kappaleita vastaavissa luokissa. Tiedostojen ja rakenteiden välillä on yksinkertainen tiedostoparseri, joka tuottaa geneerisiä rakennepuita asetetuista parametreista. Itse kaikkeusluokat lukevat näistä tiedot itselleen.

Config-luokka vastaa yhtä kokonaista konfiguraatiota, joka saadaan yhdestä tiedostosta. Tiedoston sisällä voi olla include-määre, joka aiheuttaa toisen tiedoston sisältämien tietojen lukemista siihen kohtaan, missä määre on tiedostossa. Tiedosto sisältää lohkoja, joissa on sisällä asetuksia. Pääasetukset eivät ole eksplisiittisesti missään tiedoston lohkossa. Luokka lataa tiedoston muistiin konstruktorissaan, ja instantioi lohko-olioita kun niitä esiintyy tiedostossa. Metodilla getBlock kysellään tietyn nimiset lohkot, joita voi siis olla samalla nimellä useampia (esim. kappale-lohko). Päätason globaalit asetukset, jotka eivät ole missään erillisessä lohkossa, saadaan getGlobals-metodilla. Luokan sisäisiä metodeja on mm. read, parseLine ja merge tiedoston lukemiseen, rivin käsittelyyn ja kahden asetustiedoston yhdistämiseen.

ConfigBlock-luokka kuvaa yhtä tällaista lohkoa. Se pohjautuu hashmapiin, jossa on avain/arvo-pareja tiedostosta luettuna. Konfigurointia käyttävät luokat ottavat näitä ja lukevat avainten perusteella arvoja. Luokka tarjoaa pari apumetodia mm. kokonaislukuarvon lukemiseksi: pelkkä get lukee String-tyyppisen arvon, getInt kokonaisluvun. Konfigurointia käyttävät luokat ottavat näitä blokkeja ja lukevat avainten perusteella arvonsa.

Asetusten lukeminen lohkoista luokkien jäsenmuuttujiin helpotetaan mahdollisesti annotaatioiden avulla, jolloin sijoituksia ei tarvitse erikseen koodata, vaan lataus hoituu taianomaisesti reflektiolla.

Tiedostomuoto tulee olemaan jokseenkin seuraavanlainen:

```
include constants.conf // luonnonvakiot
include moon.conf // body-blokki kuulle

/* Monirivisetkin kommentit kelpaavat.
Puolipisteistä ei ole vielä päätetty. */

body {
    name Earth
    position 0, 0, 0 // :-)
    mass 5.9736e24
    radius 6371e3
    material {
```

```
        texture earth.png
    }
}
```

1.2 Simulaatio

Simulaatio sisältää fysiikkaan liittyviä luokkia. Se on kokonaan eristetty kokonaisuus, eikä vaadi esim. graafista käyttöliittymää toimiakseen.

Vektoriluokka Vec kuvaa kolmiulotteista matemaattista vektoria ja helpottaa laskentaa, kun ohjelmassa toimitaan 3d-avaruudessa. Laskuoperaatiot (add, sub, mul yms; ottavat parametrikseen toisen vektorin) suoritetaan vektoreina sen sijaan, että komponentteja käsiteltäisiin erikseen skalaareina. Vektorien kanssa pitänee kiinnittää tehokkuussyistä huomiota siihen, koska vektoriolioita luodaan, jotta ylimääräinen muistinvarailu ja -vapauttelu minimoitaisiin. Esimerkiksi vektorien summaoperaatio muokkaa oliota suoraan, eikä luo uutta. Tämän enempää tehokkuuteen ei vielä kiinnitetä huomiota, eikä yksinkertaisissa tilanteissa tehokkuusongelmia tulekaan, mutta ohjelma suunnitellaan järkevästi skaalautuvaksi ainakin kymmenille tuhansille kappaleille.

Kappaleille on kappaleluokka Body, joka sisältää yleistä tietoa kaikista vuorovaikuttavista kappaleista, kuten sijainnin, nopeuden, massan ja nimen. Tästä periytyvät erilaiset kappaleet, joilla on erityisominaisuuksia: esim. planeetta ja raketti/satelliitti; raketilla on moottorin tuottama vakiokiihtyvyys vetovoimien lisäksi. Kappaleessa itsessään ei ole paljoakaan toiminnallisuutta, vaan se sisältää tilansa ja getterit ja setterit niille.

World-luokka kuvaa koko maailmankaikkeutta ja varastoi listassa Body-oliot. Se hoitaa simulaation ulkoisten luokkien interaktion, laskee kiihtyvyyksiä ja hoitaa integroinnin kun jokin ulkoinen taho niin käskee. loadConfig-metodi lukee asetuksista mm. gravitaatiovakion, aika-askeleen ja kappalelohkot ja luo lohkoista kappaleolioita. Metodi run ajaa yhden aika-askeleen eteenpäin valitulla integraattorilla (Worldin jäsenmuuttuja), joka kutsuu vastaavasti Worldin acceleration-metodia, joka laskee kiihtyvyyden tietylle kappaleelle tietyssä pisteessä.

Itse integraattori on Worldin jäsenmuuttujana jokin Integrator-rajapinnan toteuttava luokka. Integraattoria käsketään ajamaan yksi aika-askel (run-metodi), jolloin se lukee Worldilta kappaleiden tilat ja laskee niille uudet sekä lopuksi asettaa ne. Menetelmästä riippuen run saattaa ajaa maailmalle useamman pienen aika-askeleen peräkkäin.

Integraattori käyttää sisäisesti State-luokkaa, jossa on yhdistetty paikka- ja nopeusvektorit yhteen käsittelyn helpottamiseksi. Erikseen voisi myös siirrellä paikkoja ja nopeuksia paikasta toiseen kahtena eri muuttujana.

Simulaatiossa on mukana lokitiedostokirjoittimia, jotka periytyvät Loggerista. Ainakin GnuplotLogger ja CsvLogger toteutetaan. Ne ottavat referenssin Worldiin ja kirjoittavat sen tilan tiedostoon eri muodoissa.

1.3 Käyttöliittymä

Käyttöliittymä on monitasoinen; siinä voi tarkastella koko maailman tilaa numeerisesti taulukossa, muokata kappaleiden tilaa muokkauspaneeleista, sekä tarkastella maailmaa kolmiulotteisesta näkymästä, jossa voi kulkea fps-pelien tapaan näppäimistön ja hiiren avulla. Käyttöliittymä koostuu monista palasista, joten sekin on jaettu keskeisimpiin luokkiin.

Pääikkuna (luokka UI) on oma luokkansa, ja se avaa muita dialogeja tai piilossa olevia asetuspaneeleja kun käyttäjä niin valitsee. Pääikkunan alla on myös renderöintinäkymä. Renderöintiä päivitetään aina kun yksi aika-askel on ajettu.

UI-luokka hoitaa korkeimman tason pääasiat ja ikkunan rakentamisen. SettingPane on hybridinäkymässä oleva konfigurointipaneeli, josta voi säätää keskeisimmät ja monet renderöintiin liittyvät asiat: 3d-näkymässä voi klikata kappaletta, jolloin sen tiedot (sijainti, nopeus, massa ym) tulevat näkyviin tähän sivupaneeliin, josta niitä voi myös muokata välittömästi. Selkeyden ja modulaarisuuden vuoksi SettingPane ei hoida itse juuri mitään asetuksia, vaan se säilöö useita eri asetuswidgettejä.

BodyWidgetin vastuulla on kappaleen tietojen esittäminen. Se koostuu valikosta, josta valitaan käsiteltävä kappale (jonka voi toisaalta valita myös 3d-näkymästä), sekä tekstilaatikoista, joissa on nimi, paikan ja nopeuden x- y- ja z-komponentit, massa, ja muut kappaleen ominaisuudet.

Joskus halutaan seurata kahden kappaleen välistä etäisyyttä. DistanceWidget seuraa tätä arvoa kokonaisuutena ja komponenteittain, sekä se sisältää valintalaatikot seurattaville kappaleille.

Simuloinnin historiasta pidetään kirjaa mm. ratojen piirtämistä varten. TimeWidget sisältää sliderin, jolla aikaa voi kelata taaksepäin, sekä lisäksi napin sekä valintalaatikon, joiden avulla simuloida aikaa lisää eteenpäin tietyn aikaa, kunnes tapahtuu törmäys, tai pysähtymättä lainkaan.

Widgeteillä on joitakin metodeita tiedon kuljettamisesta edestakaisin; esimeriksi BodyWidgetillä on chooseBody-metodi, jota kutsutaan kun 3d-tilasta klikataan planeettaa, ja DistanceWidgetillä chooseBodies, jolla valitaan minkä kahden kappaleen välistä etäisyyttä seurataan.

Näiden lisäksi projektin edistyessä saatetaan toteuttaa muitakin widgettejä sitä mukaa kun lisää ideoita ja tarvetta tulee.

1.4 Renderöinti

Simulaation nykytila ja kappaleiden ratahistoriat renderöidään kolmiulotteiseen opengl-näkymään. Koska simulaatio pidetään erillisenä grafiikasta, mutta eri kappaleilla voi olla eri muotoja ja tekstuureita, grafiikkapuoli sisältää kappaleille omat luokat, jotka viittaavat vastaaviin simulaation kappaleluokkiin ja tietävät myös jotain piirtämisestä.

Renderöinnin pääasiat hallitsee Renderer-luokka, joka periytyy GLEvent-

Listeneristä. Kun näyttö halutaan päivittää, Rendererin piirtometodia kutsutaan. Renderer sisältää listan GfxBody-luokan olioista, joita se käskee piirtämään erikseen. Lisäksi se hoitaa mm. kuvakulman pyörittelyn oikeaan pikkaan.

GfxBody syö sisäänsä simuloinnin Bodyn ja tiedon mm. kappaleen muodosta, ja se osaa piirtää itsensä näytölle, kun Renderer niin käskee. Piirtometodeina mm. draw(), drawBody() ja drawHistory(). Yksityiskohdat opengl:n piirtotempuista sivuutetaan triviaaleina ja irrelevantteina.

Shape-luokka vastaa erilaisten muotojen piirtämisestä opengl:llä. Se varastoi mm. verteksilistan ja värin/tekstuurin tiedot.

2 Käyttötapauskuvaus

2.1 Komentorivi

Käyttäjä ajaa ohjelman komentoriviltä parametreilla -f gnuplot -o plot.log -c test.conf -t 1000. Ohjelma lukee tiedostosta test.conf kaiken mitä sieltä löytyy, rakentelee Body-oliot ja ajaa ladattua maailmaa 1000 sekuntia eteenpäin tulostaen plot.log-tiedostoon gnuplotin ymmärtämässä muodossa GnuplotLoggerilla simuloinnin tulokset, jonka jälkeen se sulkeutuu avaamatta muita ikkunoita.

2.2 GUI

Käyttäjä ei anna komentoriviparametreja, jolloin ohjelma luo tyhjän maailman aluksi ja näyttää hybridinäkymän. (Ikkunat alustelevat itsensä yms, muttei simulaatio tee mitään kummempaa). Käyttäjä lataa test.conf-tiedoston asetukset valikoista naksuttelemalla, jolloin asetusnäkymiin tulee tietoja kappaleista ja 3d-näkymään maailman yleistilanne; simulaatio alustuu vastaavasti kuin komentoriviltä ajettaessa Bodyiksi. Samalla renderi lukee samaisista asetuksista sen, miltä kappaleiden tulisi näyttää (säde, tekstuurit, muoto yms). Käyttäjä vaihtelee 3d-näkymän tilaa, koska ei ole siihen tyytyväinen, navigoimalla siellä hiiren ja näppäimistön avulla fps-pelien tapaan. Seuraavaksi käyttäjä asettaa ohjelman simuloimaan jatkuvasti, jolloin kappaleet alkavat pyöriä 3d-näkymässä. Taustalle aukeaa säie pyörittelemään simulaatiota eli integroimaan kappaleiden tiloja eteenpäin, ja 3d- ja konffinäkymät päivittelevät itseään samassa tahdissa. Tästä voi jatkaa seikkailemalla renderillä tai virittelemällä kappaleiden parametreja.

3 Algoritmit

Pääalgoritmi tehtävässä on simulaation laskenta; muualla esiintyy pieniä ja suhteellisen merkityksettömiä tavallisia teknisiä algoritmeja, joita ei edes vielä käsitellä.

3.1 Simulaatio

Varmasti suunniteltuna on vain neljännen asteen Runge-Kutta-menetelmän (RK4) käyttö. Kappaleiden tilat y ja aika t muuttuvat seuraavasti (alaindeksit y:lle ja t:lle kuvaavat integrointikierrosten indeksejä):

$$y_{n+1} = y_n + \tfrac{1}{6}h\left(k_1 + 2k_2 + 2k_3 + k_4\right) \quad (1)$$
$$t_{n+1} = t_n + h \quad (2)$$

Arvot k_i kertovat tietoja kulmakertoimesta ja määräytyvät seuraavasti:

$$k_1 = f(t_n, y_n) \quad (3)$$
$$k_2 = f(t_n + \tfrac{1}{2}h, y_n + \tfrac{1}{2}hk_1) \quad (4)$$
$$k_3 = f(t_n + \tfrac{1}{2}h, y_n + \tfrac{1}{2}hk_2) \quad (5)$$
$$k_4 = f(t_n + h, y_n + hk_3) \quad (6)$$

k_1 on kulmakerroin nykyhetkellä; k_2 on Eulerin menetelmällä laskettu puolikkaan aika-askeleen jälkeen; k_3 vastaava, mutta kulmakertoimen k_2 avulla; k_4 lopussa eli aika-askeleen päässä nykyhetkestä k_3:lla. Näiden painotettu keskiarvo määrää oletetun derivaatan kohdassa n.

Koska maailman tila on yhtenäinen eri kappaleiden suhteen ja kyseessä on periaatteessa yksi suuri tilavektori, ei kappaleiden tiloja päivitellä vaiheiden välillä erikseen, vaan kaikkien kappaleiden tilat lasketaan askeleiden välillä. Teknisesti tilavektori on hahmottamissyistä hajautettu erillisiin kappaleluokkiin. Esimerkiksi siis k_2:n laskemisessa yhtä kappaletta varten muuta maailmaa ei pidetä paikallaan, vaan vaikutus lasketaan kaikelle.

Muita vaihtoehtoisia integrointimenetelmiä olisi mm. Verlet ja Dormand-Prince. Nämä saatetaan implementoida vaihtelun vuoksi. Euleria ei lasketa, koska se on kelvottoman huono.

Simulaatio saatetaan toteuttaa tukemaan hyvin myös erittäin suurta määrää partikkeleita optimoimalla monet kaukana olevat kappaleet olettamalla yhdeksi siten, että ne approksimoidaan yhdeksi pisteeksi: jos aikaa jää, tutkitaan esim. Barnes-Hut-simulaatiota.

3.2 Muuta

Yllä oleva algoritmi on koko ohjelman pääasia, mutta tässä myös joitain pienempiä huomioon otettavia asioita.

Askelpituutta saatetaan muuttaa lennossa, jos käyttäjä sen sallii. Kappaleiden ollessa kaukana toisistaan simulaatiota voidaan tietysti laskea varsin pitkälle, mutta lähekkäisille kappaleille pitkä askelväli aiheuttanee väärää käytöstä. Askelpituuden muuttaminen adaptiivisesti onnistunee tarkastelemalla kappaleiden välisien etäisyyksien muutoksia. Tämä nopeuttaa simulaatiota, kun lyhyt väli on tarpeeton.

Grafiikan piirto on pääosin suoraviivaista, mutta jos simulaatio kestää erityisen pitkään, esim. rataviivojen piirtoa ei välttämättä jatketa aivan alusta saakka. Myös joitain askelia saatetaan hyppiä välistä pois, koska karkeampikin viiva riittää.

Historiadatan tallentamiseen ei välttämättä käytetä jokaista askelta hyväksi, sillä tällöin ohjelman muistinkäyttö voi nousta pilviin nopeasti. Takaisin kelatessa suurempien askelien välistä voidaan interpoloida arvoja lennossa.

Myöskään lokitiedostoon ei välttämättä aina tallenneta jokaisen askeleen pistettä. Asetustiedostossa on mahdollisuus määritellä esim. miten monen askeleen välein maailman tila tulostetaan lokiin.

4 Tietorakenteet

Perustietorakenteisiin ei tarvita mitään listoja kummempaa. Kappaleiden määrä voi muuttua ohjelman toiminnan aikana, historiadataa kertyy jatkuvasti jne, jolloin ArrayList tullee olemaan varsin käytetty perustaulukoiden sijaan.

Jos jää aikaa raskaaseen partikkelisysteemiin, Barnes-Hut-menetelmässä tarvitaan octree, joka toteutetaan itse. Octree jakaa 3d-maailman puumaisesti eri osiin, mikä nopeuttaa suuren partikkelimäärän käsittelyä.

5 Aikataulu

Aikataulu on jaettu arvioihin, mitä kunkin viikon loppuvaiheessa tulisi ainakin olla tehtynä. Seuraavassa viikkonumerot ja vastaavan viikon toiminnan kuvaus.

7 Tutustutaan menetelmiin ja arvioidaan vaihtoehtoja, jotain randomkoodailua, yleissuunnitelma

8 Teknosuunnitelma; Softa etsii muotoaan ja tekee jotain järkevää joka suhteessa; konffit latautuu, simu toimii ja kirjoittaa lokeihin, kälissä on pari nappia ja opengl-canvas piirtää palluroita

9 Suunnitelmademo, softa kehittynyt hieman joka suhteessa edellisestä

10 Käyttöliittymässä on käytettäviä nappeja ja 3d-näkymässä on jotain automagiikkaa jolloin testaaminen on vikkelää

11 3d-näkymässä on kauniita tekstuureita ja siellä voi kävellä ja valita asioita, jolloin niiden tiedot tulevat settingspaneen

12 Lokitiedostot voi konffata ja konffauspaneelissa (laajempi settingspane, josta näkee koko maailman tiedot selkeämmin) on kokonaisuudessaan jotain, mm. lista kappaleista

13 Checkpoint; konffitiedostojen virheellisyys hallitaan oikein, konffauspaneeli myös reagoi muutoksiin

14 3d-näkymässä toimii ratahännät ja kappaleen seuranta ja origosäätö

15 Kiillotusta sieltä täältä, ehkä jopa octree-optimaatio

16 20.4. DL

Työtunteja kuluu viikottain 10-15. Koodin syntyessä suunnitellaan ja kirjoitetaan mahdollisuuksien mukaan myös testikoodia sekä dokumentoidaan ja harkitaan erilaisia toteutusvaihtoehtoja; itse koodausprosessi on tästä vain osa. Viikkoroadmapit ovat vain suuntaa-antavia.

6 Yksikkötestaussuunnitelma

- Config: tarkistetaan, että latausmetodit lataavat globaalit muuttujat ja blokit kunnolla sekä skippaavat kommentit.
- ConfigBlock: getterit palauttavat oikean arvon niitä hakiessa. Esim. getInt heittää poikkeuksen mikäli arvoa ei voi tulkita kokonaislukuna. Setterit vastaavasti.
- Simulaation toimivuus: Pyöritellään maapalloa auringon ympärillä pari aika-askelta ja katsotaan, että on siellä minne ennustettiin. Kiihtyvyysfunktiosta katsotaan, että kaikki planeetat vaikuttavat toisiinsa.
- Lagrangen pisteessä oleva kappale ei integroidessa liiku suhteessa muihin.
- Törmäykset: törmäystarkistusmetodi reagoi oikein kun kappaleet ovat liian lähekkäin.
- Käyttöliittymän napit tekevät mitä pitääkin; testattava lähinnä käsin.
- Asetuswidgettien chooseBody-metodit ym. reagoivat esim. valitsemalla oikean kappaleen.
- Renderer piirtää sille annetut palikat oikeisiin kohtiin. Apurakenteeksi jokin testihomma joka käynnistää rendausikkunan dummykappaleilla.
- Rendererin navigointimetodit; testattava käsin.

Konepellin alla olevat asiat, eli asetushärpättimet ja simulaation, voi testata hyvin helposti, ja ne vaativatkin algoritmisuudessaan eniten testausta. Käyttöliittymän testailu on hankalampaa, kun se vaatii ihmistä heiluttelemaan hiirtä. Sen testaaminen on parasta tehdä siten, että kun simulaation toimivuus on todettu, havainnoidaan että käyttöliittymä reagoi halutusti –

suuri osa tästä hoituu ihan käyttämällä ohjelmaa. Esim. rendererin piirtohässäkät saa testattua helposti siten, että ladataan jokin alkutilanne josta tiedetään miltä sen pitäisi näyttää, ja katsotaan että näytöllä on se mitä kuuluu olla.

7 Kirjallisuusviitteet ja linkit

Tietoa ammennetaan mm. seuraavista lähteistä:

- http://gafferongames.com/game-physics/integration-basics/
- http://en.wikipedia.org/wiki/Runge-Kutta_methods
- http://en.wikipedia.org/wiki/Verlet_integration
- http://www.gamasutra.com/resource_guide/20030121/jacobson_01.shtml
- http://en.wikipedia.org/wiki/Dormand-Prince
- http://en.wikipedia.org/wiki/Barnes-Hut_simulation
- http://en.wikipedia.org/wiki/Octree
- https://noppa.tkk.fi/noppa/kurssi/t-106.1240/luennot
- http://download.oracle.com/javase/6/docs/api/