Dokumentti

1. Henkilötiedot

Lauri Koskenniemi
652416
Elektroniikka ja sähkötekniikka
29.4.2018

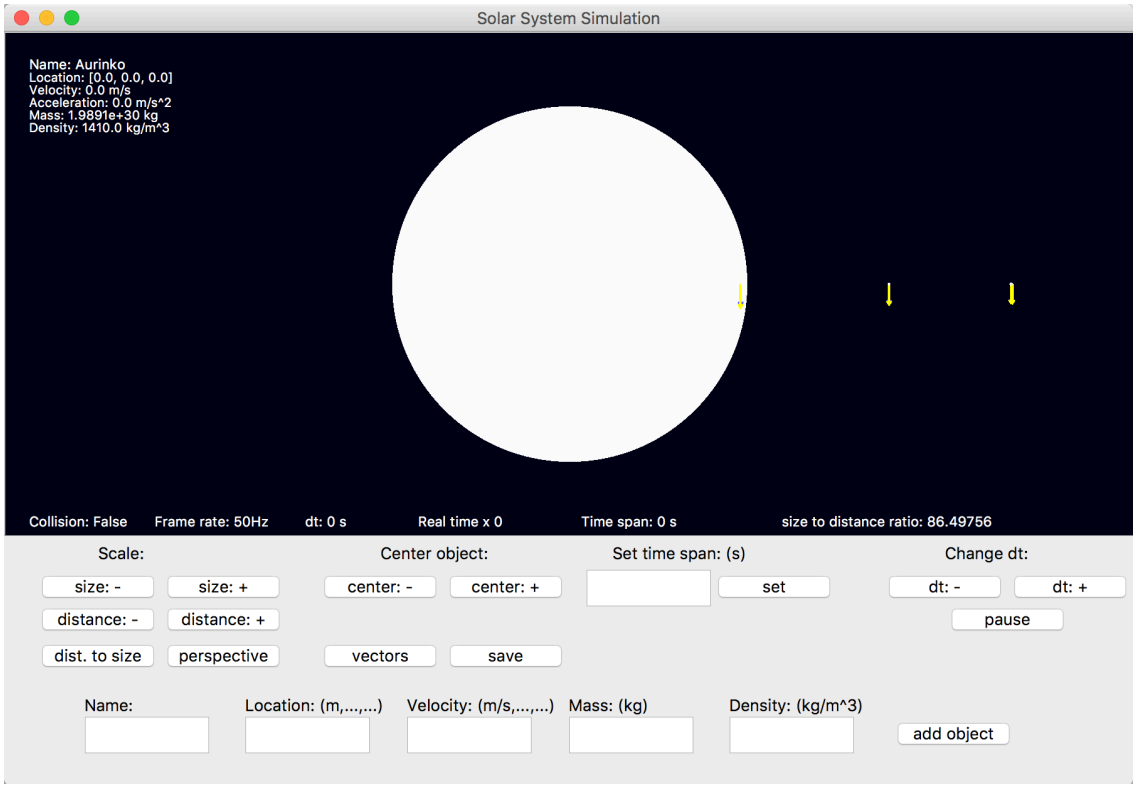
2. Yleiskuvaus

Aurinkokuntasimulaattori mallintaa taivaankappaleiden ja satelliittien liikkeitä. Simulaatioon voi määrittä ajanjakson liikkeiden mallintamiseen, jonka päätyttyä simulaatio pysähtyy. Myös kappaleiden väliset törmäykset pysäyttävät simulaation. Liikkeiden mallintaminen tapahtuu kolmiulotteisesti ja käyttäjä voi valita kuvakulman kahdesta eri suunnasta. Laskennassa käytetään Newtonin toista lakia ja painovoimalakia, sekä Eulerin menetelmää. Käyttäjä voi valita näytölle keskitettävän taivaankappaleen, jolloin myös kyseisen kappaleen tietoja tulostetaan näytölle. Näitä tietoja voi myös tallentaa tekstitiedostoon missä simulaation vaiheessa tahansa.

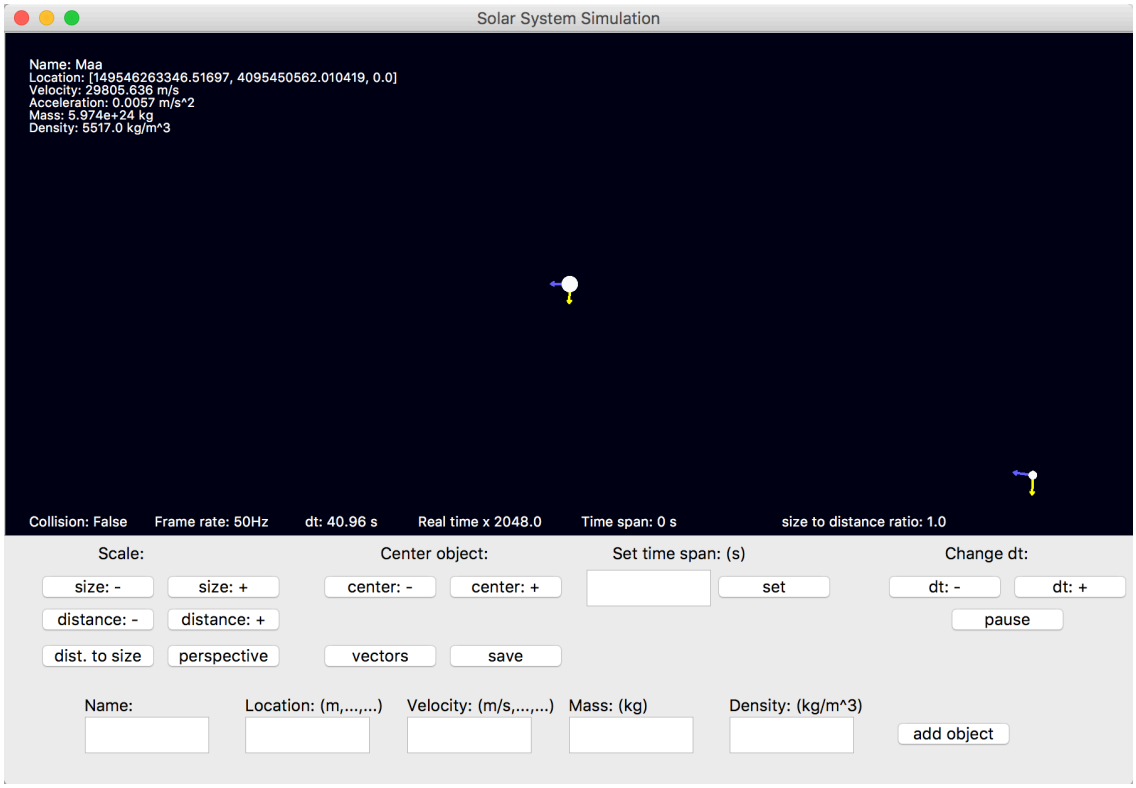
3. Käyttöohje

Ohjelma käynnistetään avaamalla main.py -tiedosto Python-tulkilla, mikä avaa graafisen käyttöliittymän. Ohjelman käynnistyessä simulaatio on pysähdyksissä. Painamalla ”pause” näppäintä simulaatio lähtee liikkeelle. Simulaatioon voi määrittää ajanjakson sekunneissa, jonka kuluttua se pysähtyy. Simulaation nopeutta, eli aika-askelta, voi muuttaa ”Change dt:” näppäimillä. ”Center object:” näppäimillä käyttäjä voi vaihtaa näytön keskellä näkyvää taivaankappaletta. Taivaankappaleiden kokoa ja etäisyyksiä voi skaalata erikseen ja etäisyydet voi skaalata kappaleiden kokoon ”Scale:” näppäimillä. ”Perspective” näppäin vaihtaa kuvakulmaa kahden koordinaatistoakselin välillä. Ohjelman käynnistyessä taivaankappaleiden nopeus-, ja kiihtyvyysvektorit ovat näkyvissä. Painamalla ”vectors”

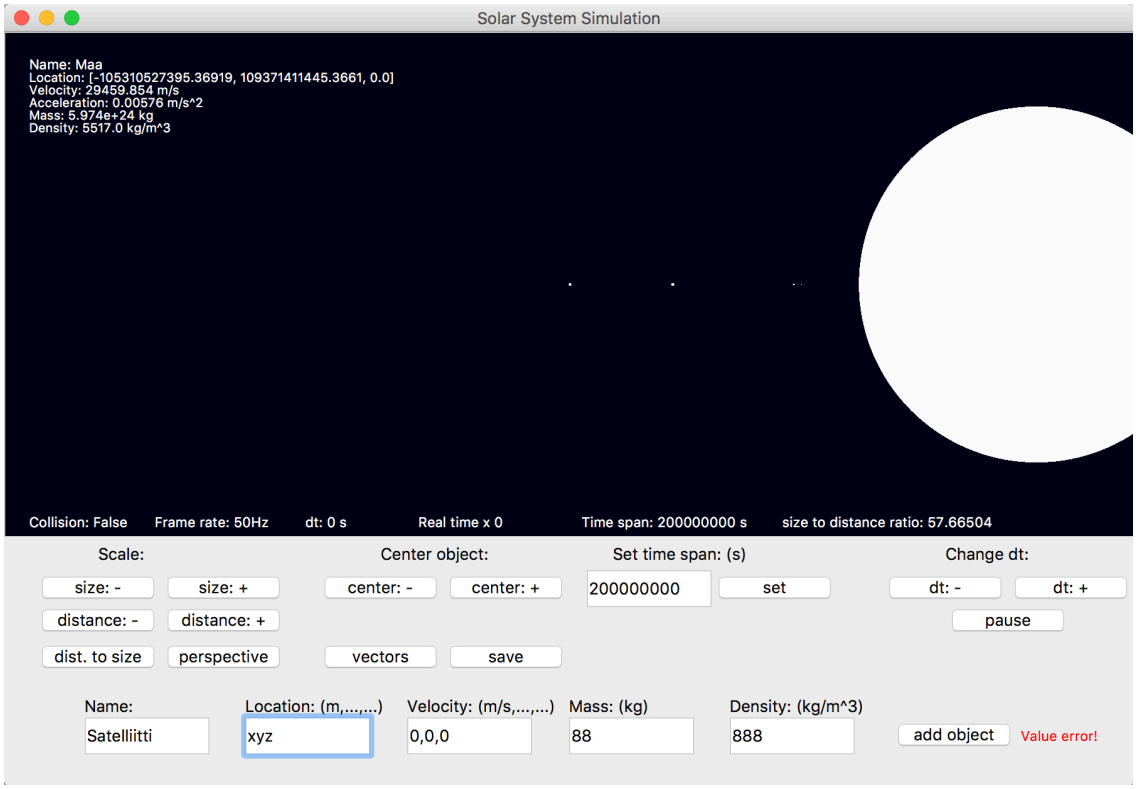
näppäintä vektorit saa piilotettua. Simulaatioon voi luoda uuden kappaleen kirjoittamalla alkuarvot tekstikenttiin ja painamalla ”add object” näppäintä. Kappaleen sijainti ja nopeus annetaan kolmella pilkulla erotetulla arvolla, joissa arvot vastaavat x-, y- ja z-akseleita. Mikäli käyttäjä antaa arvoja väärässä muodossa ”add object” näppäimen viereen ilmestyy teksti ”Value error!”. ”Save” näppäin lisää näytöllä näkyvät arvot merkkijonoon, joka tallennetaan tiedostoon. Ohjelman ajon aikana arvoja voi tallentaa monta kertaa, mutta ohjelman käynnistyessä merkkijono tyhjennetään.



Kuva aloitusnäkymästä.



Kuvassa Maa ja Kuu, sekä Maan tietoja vasemmassa yläreunassa. Koon ja etäisyyden suhde on yksi ja aika-askel on noin 41 sekuntia eli simulaation nopeus on yli 2000 kertainen todelliseen nopeuteen.



Kuva ilman vektoreita, sivuperspektiivistä, sekä virheellisellä alkuarvolla.

4. Ulkoiset kirjastot

Olen käyttänyt projektissani PyQt5 ja Pythonin math, sekä unittest kirjastoja. Math kirjastosta ohjelma käyttää neliöjuurta, logaritmia ja pii:n arvoa. Unittest kirjastoa käytetään ohjelman yksikkötestaukseen. Kaikki ohjelman grafiikka on toteutettu PyQt5 kirjaston avulla.

5. Ohjelman rakenne

Main funktio käynnistää ohjelman luomalla olion System luokasta. FileReader luokan metodi käy läpi aloitustiedostossa olevat kappaleet ja luo niistä Object oliot luotuun System olioon. Tämän jälkeen luodaan PyQt5:n QApplicaton olio, sekä System oliolle GUI olio, joka vastaa grafiikan piirtämisestä. Object luokka sisältää taivaankappaleen tietoja, kuten sijainti ja massa. System luokan tarkoitus on koota luodut Object oliot ja päivittää niiden arvoja. System luokka laskee jokaiseen kappaleeseen vaikuttavan voiman, jonka muut kappaleet siihen kohdistaa ja voiman aiheuttaman kiihtyvyyden. Laskemiseen System luokka käyttää Vector ja Physics luokkia. Physics luokassa on määritelty painovoiman ja kiihtyvyyden laskukaavat. Vector luokalla luodaan vektori olio, joka sisältää kolmialkioisen listan. Vector luokka sisältää myös vektori olioissa tarvittavia laskukaavoja. GUI luokka piirtää ohjelman graafisen käyttöliittymän. Taivaankappaleiden ja vektorien piirtämiseen GUI käyttää ObjectGraphics ja VectorGraphics luokkia. Näissä luokissa käytetään myös Object ja Vector luokkia Object metodeja ja vektorilaskentaa varten. Jokaiselle Object oliolle luodaan GUI luokassa ObjectGraphics olio, sekä kaksi VectorGraphics olioa. ObjectGraphics luokka luo Object oliolle näytöllä näkyvän ellipsin ja laskee näytöllä näkyvän sijainnin Object olion todellisesta sijainnista. VectorGraphics olio kuvaa Object olion nopeus- tai kiihtyvyysvektoria. VectorGraphics luokassa lasketaan ObjectGraphics luokan tavoin vektorin sijainti näytöllä, sekä vektorin suunta ja pituus. GUI luokassa on määritelty kaikki muu näytöllä näkyvä, kuten näppäimet ja tekstikentät. GUI luokka toimii myös simulaation kellona. Se päivittä Object olioiden arvot, sekä grafiikan 50 kertaa sekunnissa. Ohjelmassa on myös Test luokka, jota käytetään yksikkötestaamiseen. UML-tiedosto on repositoryssa doc-kansiossa.

6. Algoritmi

Ohjelman keskeisin algoritmi on painovoiman laskeminen. Sen laskemiseen käytetään Newtonin painovoimalakia:

$$F = G\frac{m_1 m_2}{r^2}$$

Simulaatiossa arvot ovat vektori muodossa, joten myös voima täytyy palauttaa vektorina. Ohjelma laskee ensin kappaleiden välisen etäisyyden skalaarina, jolloin myös voima saadaan skalaarina. Tämän jälkeen lasketaan suunta toiseen kappaleeseen yksikkövektorina ja kerrotaan se voiman skalaariarvolla, jolloin voima saadaan jaettua vektorin komponentteihin.

Kappaleen kiihtyvyys saadaan Newtonin toisesta laista:

$$F = ma \Leftrightarrow a = \frac{F}{m}$$

Myös kiihtyvyyden täytyy olla vektorimuodossa. Kiihtyvyysvektori saadaan jakamalla voimavektorin jokainen komponentti kappaleen massalla.

Taivaankappaleiden sijainnit ratkaistaan Eulerin menetelmää käyttäen:

$$v + dv = v + a * dt$$

$$x + dx = x + v * dt$$

Nopeuden uusi arvo saadaan lisäämällä vanhaan arvoon kiihtyvyyden aiheuttama nopeuden muutos. Nopeudenmuutos taas saadaan kertomalla kiihtyvyys aika-askeleella. Samalla tavalla paikanmuutos saadaan kertomalla nopeus aika-askeleella ja lisäämällä se vanhaan arvoon, jolloin kappaleelle saadaan uusi sijainti.

7. Tietorakenteet

Ohjelman kaikki tietorakenteet perustuvat listoihin. Tiedot varastoidaan joko suoraan listoihin tai Vector luokan olioon, jonka arvot on tallennettu listaan. Kun Vector olio luodaan, sille annetaan kolme alkuarvoa, jotka se siirtää listaan. Kolmiulotteista dataa on helpompi käsitellä laskuissa yksittäisinä olioina, kun niille määritellyt metodit suorittavat laskut jokaiselle vektorikomponentille, eli listan alkiolle, erikseen. Listoja käytetään myös muun muassa varastoimaan simulaatioon lisätyt taivaankappaleet. Kappaleiden määrä voi vaihtua

ja niitä käydään läpi simulaation aikana, joten lista on hyvä muuttuvatilainen tietorakenne tiedon varastoimiseen. Taivaankappaleiden tietoja olisi voinut varastoida myös esimerkiksi sanakirjaan, mutta silloin jokaisen olion tiedot täytyisi päivittää jatkuvasti. Listaan voi lisätä olion suoraan, jolloin myös kappaleen tiedot saadaan kutsumalla kyseisen olion metodeja.

8. Tiedostot

Ohjelma käyttää kahta tekstitiedostoa. Ohjelman käynnistystiedosto, start_file.txt, sisältää taivaankappaleiden alkuarvoja, joista ohjelma luo oliot ja piirtää ne näytölle. Olion tiedot aloitetaan '#'-merkillä, jonka jälkeen olevasta tekstistä tulee taivaankappaleen nimi. Muut tiedot annetaan kirjoittamalla location, velocity, mass ja density, sekä niiden arvot kaksoispisteellä erotettuna. Ohjelma ohittaa rivin, jos se aloitetaan '//'-merkillä. Esimerkiksi Maan alkuarvot voidaan kirjoittaa:

#Maa
location: 149600000000,0,0
velocity: 0,29800,0
mass: 597400000000000000000000
density: 5517

Ohjelman toinen tiedosto on tietojen tallentamista varten. GUI luokassa on merkkijono, johon voi lisätä näytöllä näkyvät tiedot taivaankappaleesta. Merkkijono tallennetaan save.txt tiedostoon.

9. Testaus

Ohjelmaa tehdessä testasin yksittäisten luokkien toimivuutta tulostamalla arvoja konsoliin. Esimerkiksi Object luokan metodeja testasin tulostamalla metodien palautusarvoja konsoliin, sekä Vector ja Physics luokkia testasin laskemalla Vector olioilla erilaisia laskutoimituksia konsoliin.

Ohjelmassa on Test luokka yksikkötestausta varten. Yksikkötestauksessa kokeillaan luotujen olioiden tallentumista System luokan listaan, olioiden siirtymistä oikein System luokan päivittyessä ja ohjelman reagoimista olioiden törmäykseen. Yksikkötestaus ei ole niin laaja kuin suunnitelmassa, koska esimerkiksi ObjectGraphics olion sijaintia ei voi helposti lukea luokan ulkopuolelta.

10. Ohjelman tunnetut puutteet ja viat

Ohjelman laskennan epätarkkuus kasvaa aika-askeleen kasvaessa. Esimerkiksi Kuu voi poiketa huomattavasti radaltaan, jos aika-askel on yli 1000 sekuntia. Simulaatiosta saisi tarkemman käyttämällä Runge—Kutta-menetelmää. Nopeus- ja kiihtyvyysvektorit osoittavat väärään suuntaan (alas), jos kulman laskennassa käytettävä komponentti on nolla. Käyttöliittymään voisi lisätä vielä ominaisuuksia. Esimerkiksi ohjelmaan voisi lisätä uudelleenkäynnistysnäppäimen, koska simulaatio pysähtyy, jos tapahtuu törmäys, eikä sitä voi enää jatkaa sulkematta ohjelmaa. Alkuarvojen syöttämisestä voisi tehdä käyttäjäystävällisempää mahdollistamalla arvojen antaminen kymmenpotenssimuodossa. Myös taivaankappaleiden muokkaamisen ja poistamisen simulaatiosta voisi lisätä.

11. 3 parasta ja 3 heikointa kohtaa

Toteuttamani vektoriluokka helpottaa kolmiulotteisten arvojen käsittelyä. Mahdollisuus skaalata taivaankappaleiden koon ja etäisyyksien suhde todelliseksi. Käyttäjä näkee kaikista kappaleista tietoa ja tiedot voi tallentaa tiedostoon.

Laskennan tarkkuus olisi parempi Runge—Kutta-menetelmällä. GUI luokkaa olisi voinut jakaa vielä muihin luokkiin. Esimerkiksi näppäimet ja tekstikentät voisivat olla toisessa luokassa. Simulaation muokkauksen rajallisuus. Kun olio on luotu simulaatioon, sitä ei voi enää muokata tai poistaa. Ohjelma täytyy käynnistää uudestaan ja halutut muutokset täytyy tehdä alkuarvoihin tiedostoon tai uutta kappaletta luodessa.

12. Poikkeamat suunnitelmasta

Ohjelma noudattaa pääosin suunnitelmaa. En kuitenkaan toteuttanut Coordinates luokkaa, mutta lisäsin VectorGraphics luokan. Suunnitelmassa mainittua Coordinates luokkaa ei tarvittu, koska alkuarvot ja laskenta tapahtuvat molemmat karteesisessa koordinaatistossa, joten koordinaatistomuunnoksia ei tarvita.

Ajankäyttöarvio oli melko tarkka. System luokan tekemiseen ei kuitenkaan kulunut niin kauan kuin arvelin, koska luokka on melko yksinkertainen. GUI luokan tekeminen kesti oletettua kauemmin, koska siinä on paljon sisältöä ja välillä Qt:n dokumentointiin tutustumiseen kului aikaa. Myös ObjectGraphics luokan tekemiseen kului suunnitelmaa enemmän aikaa QGraphicsItemiin tutustumisen takia.

Ohjelman toteuttamisjärjestys poikkesi suunnitelmasta. Aloitin suunnitelmasta poiketen Vector, Physics ja Object luokista ja siirryin vasta sitten System ja GUI luokkiin. System ja GUI luokista olisi ollut vaikea aloittaa, koska en esimerkiksi tiennyt vielä Object luokkaan vaadittavia alkuarvoja.

13. Toteutunut työjärjestys ja aikataulu

Aloitin ohjelman tekemisen 9.3. Vector ja Physics luokista. 16.3. aloitin Object ja Test luokkien teon. Vector ja Physics luokat olivat jo lähes valmiita, joten sain viikonlopun aikana kokeiltua luokkia Object ja Test luokkien kanssa. 23.3. aloitin GUI ja System luokat. ObjectGraphics luokan tekemisen aloitin 27.3. ja 6.4. FileReader luokan. 20.4. aloitin viimeisen luokan, VectorGraphics:n tekemisen. GUI luokkaa tein jatkuvasti muiden luokkien ohella ja käytin siihen suunniteltu enemmän aikaa. Muut luokat sain suunnitelman mukaisesti tehtyä. Yhteensä käytin ohjelmointiin noin 55 tuntia.

14. Arvio lopputuloksesta

Ohjelma toimii pääosin hyvin. Kaikki ohjelmaan toteutetut ominaisuudet toimivat ilman virheitä. Ohjelman huonoja puolia ovat Eulerin-menetelmän epätarkkuus suurilla aika-askelilla ja käyttöliittymän puutteet. Eulerin-menetelmän sijasta voisi käyttää Runge-Kutta-menetelmää ja käyttöliittymään voisi lisätä muokkaus mahdollisuuden taivankappaleille. GUI luokkaa olisi voinut jakaa pienempiin kokonaisuuksiin. Laajennusten tekeminen ohjelmaan onnistuu melko helposti, sillä uusi luokka vaatisi muutoksia vain GUI ja mahdollisesti System luokkaan.

15. Viitteet

https://grader.cs.hut.fi/static/y1/
https://gafferongames.com/post/integration_basics/
https://doc.qt.io/qt-5.10/reference-overview.html
https://docs.python.org/3/

16. Liitteet

https://version.aalto.fi/gitlab/koskenl1/aurinkokuntasimulaattori