Projektidokumentti: Aurinkokuntasimulaattori

Lauri Myöhänen, 430764

Elektroniikka ja sähkötekniikka

05.05.2017

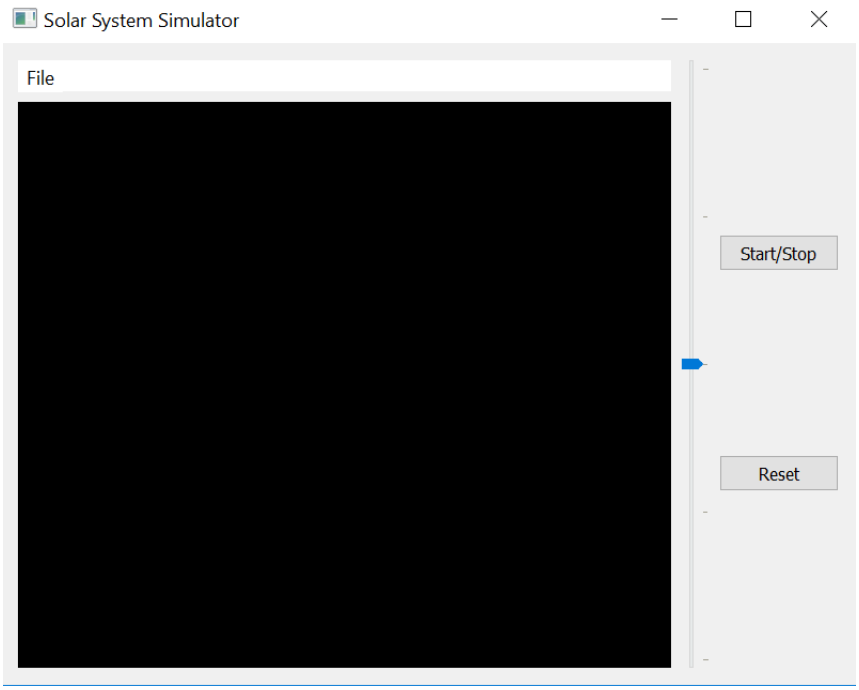
Yleiskuvaus

Ohjelma simuloi taivaankappaleiden liikettä avaruudessa. Kappaleiden alkutilanne, joita voidaan luoda useita, luetaan tiedostosta. Suunnitelmasta poiketen ohjelmassa ei ole ominaisuutta satelliitin tai kappaleen lisäämiseen (ks. Puutteet & viat). Projektin toteutuksen vaikeustaso on helpon ja keskivaikean työn välimaastossa.

Käyttöohje

Ohjelman voi käynnistää esimerkiksi Windowsin komentoriviltä hakeutumalla lähdekoodin tiedostaan ja ajamalla moduulin main.py komennolla *python main.py* tai avaamalla projektin ohjelmointiympäristössä ja ajamalla sen sitä kautta.

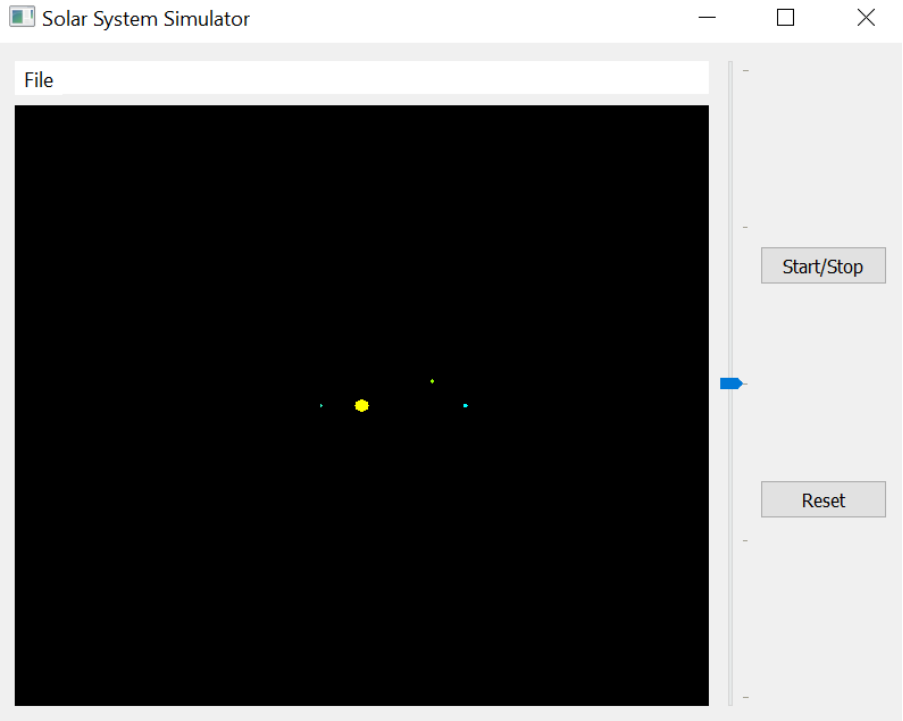
Ohjelman käynnistyttyä aukeaa kuvan 1 mukainen ikkuna. Tässä vaiheessa oikeassa reunassa olevat komennot eivät tee mitään, sillä alkutilannetta ei ole ladattuna. Alkutilanne ladataan menemällä *File*-valikkoon, josta löytyvät vaihtoehdot *Load System* ja *Exit*. Valitsemalla *Load System* aukeaa tiedostodialogi, josta voi valita alkutilatiedoston.



Kuva 1: Käyttöliittymä

Alkutilanteen lataamisen ohjelma piirtää kappaleen simulaatioikkunaan (kuva 2), minkä jälkeen simulaation voi käynnistää *Start/Stop*-painikkeella. Simulaation nopeuden voi säätää

liukukytkimellä. Nopeutta voi kuitenkin muuttaa vain simulaation ollessa pois päältä. *Reset*-painike palauttaa simulaation alkutilanteeseen. Kuvakulmaa voi muuttaa painamalla hiiren vasenta tai oikeaa näppäintä simulaatioikkunan sisällä ja vetämällä hiirtä. Zoomaus toimii hiiren pyöröpainikkeella.



Kuva 2: Alkutilanne ladattuna

Rakenne

Ohjelma jakautuu kahteen osaan, joista toinen käsittelee kappaleiden sijainnin laskentaa ja toinen graafista käyttöliittymää. Laskennallisen osan ytimessä ovat luokat *System* ja *CelestialBody*. *System*-luokka sisältää tiedot kaikista simulaatiossa mukana olevista kappaleista sekä keskeiset metodit *update_body_accelerations*, *update_body_velocity*, *update_body_position*. Näillä metodeilla päivitetään kappaleiden kiihtyvyys, nopeus ja sijainti simulaation jokaisessa askeleessa.

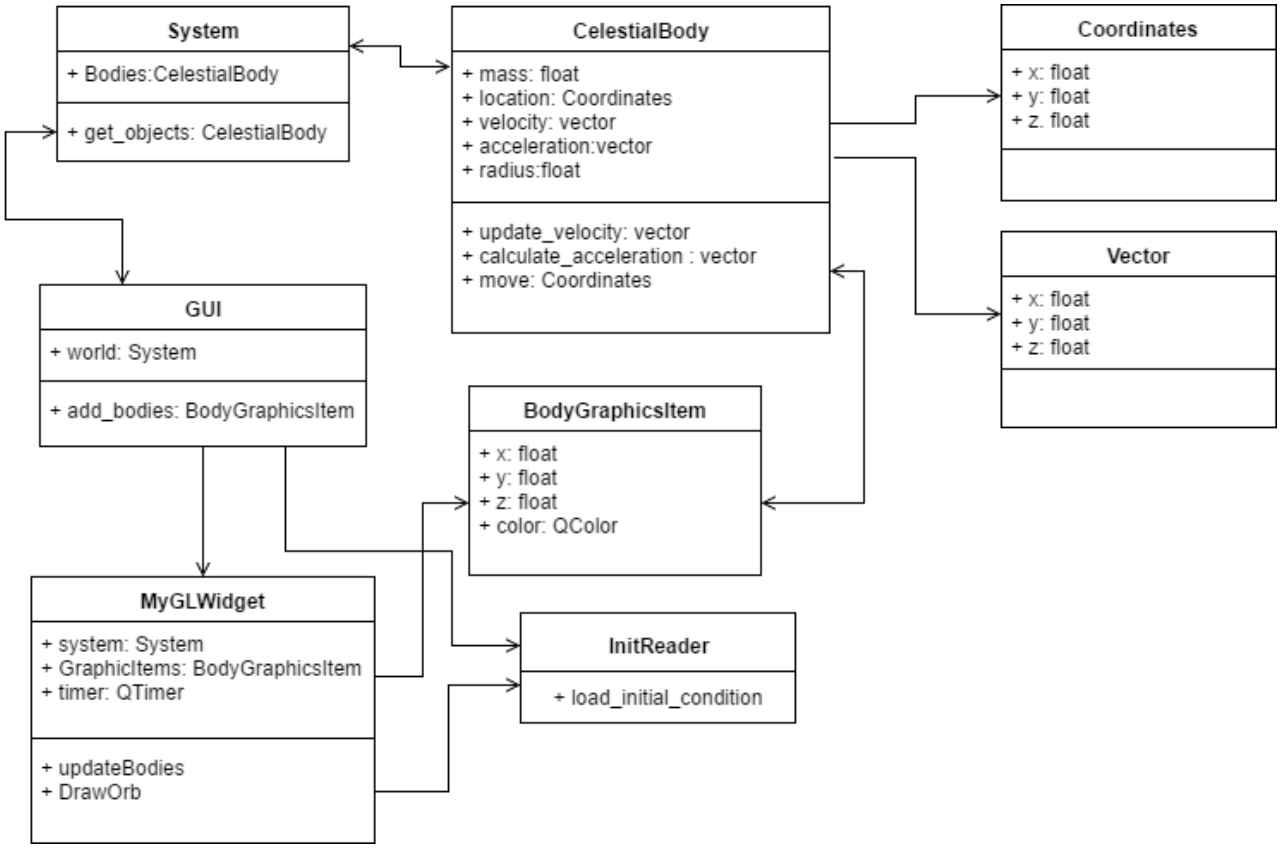
Varsinainen laskeminen tapahtuu kuitenkin *CelestialBody*-luokan sisällä. Tämä luokka sisältää tiedot yhdestä kappaleesta ja ohjelman tärkeimmät metodit: *calculcate_acceleration*, *update_velocity* ja *move*, joilla lasketaan kappaleen kiihtyvyys, nopeus sekä seuraava sijainti.

Kappaleen sijainti on *Coordinates*-luokan instanssi, joka sisältää koordinaatit *x*, *y*, *z* ja metodit niiden palauttamiseen ja muuttamiseen. Oikeastaan sijainnin voisi tallentaa myös pelkkiin muuttujiin *x*, *y* ja *z*, mutta jos ohjelmaa haluaa parantaa tai lisätä uusia ominaisuuksia oma luokka sijainnille helpottaa muutoksien tekemistä. Nopeus ja kiihtyvyys ovat *Vector*-luokan instansseja, joka on hyvin samanlainen luokka kuin sijainnin, mutta siinä muutamia lisätoimintoja mm. vektorin pituuden ja suunnan hallitsemiseen.

Graafinen käyttöliittymä muodostuu kahdesta pääluokasta *GUI* ja *MyGLWidget*. *GUI* on johdettu PyQt:n luokasta QWidget, ja sillä hallitaan pääikkunan muita widgettejä. Näistä tärkein on edellä

mainittu *MyGLWidget* (johdettu luokasta QOpenGLWidget), joka on vastuussa kappaleiden piirtämisestä. Luokan keskeisin metodi on *drawOrb*, jolla piirretään annettujen arvojen (sijainti, koko, väri) mukainen pallo. Luokalla on pääsy *System*-luokkaan, josta se hakee tiedot kappaleista sekä kutsuu simulaatiometodeja ajastimen tahdittamana.

Alkutilanteen lukemiseen käytetään luokkaa *InitConditionReader*, joka luo systeemin sekä siihen kuuluvat kappaleen alkutilatiedoston tietojen mukaisesti. Luokkaa käytetään kahdessa eri toiminnossa: systeemin lataamisessa *File*-valikon kautta sekä simulaation nollaamisessa *Reset*-painikkeella.



Kuva 3: Luokkakaavio

Algoritmit

Ohjelmassa on neljä olennaista algoritmia. Näistä kolme ovat kappaleiden kiihtyvyyden, nopeuden ja sijainnin laskeminen. Ohjelman kappaleet liikkuvat kolmiulotteisessa avaruudessa, joten ominaisuudet on jaettu niiden x, y, ja z -komponentteihin, joista jokainen käsitellään erikseen.

Kiihtyvyys lasketaan Newtonin toisen lain sekä Newtonin gravitaatiolain avulla.

$$F = m_1 a, F = \gamma \frac{m_1 m_2}{r^2}$$

ratkaistaan kappaleen 1 kiihtyvyys a

$$a = \gamma m_2 r^{-2}$$

jossa γ on gravitaatiovakio, m_2 toisen kappaleen massa ja r kappaleiden välinen etäisyys. Yhden kohteen kokonaiskiihtyvyys saadaan laskemalla yhteen muiden kappaleiden sille aiheuttamat kiihtyvyydet.

Nopeus tai nopeuden muutos saadaan kaavalla

$$v = v_0 + at$$

jossa v_0 on alkunopeus ja t on muutokseen kulunut aika eli ohjelman tapauksessa simulaatioaskel

Kappaleen sijainnin muutos simulaatioaskeleen aikana saadaan kaavalla

$$x = x_0 + v_0 t + 0.5at^2$$

jossa x_0 on kappaleen sijainti ennen simulaatioaskelta.

Yhden simulaatioaskeleen laskemisessa suoritetaan kaikki edellä mainitut toimenpiteet siten, että ensin lasketaan jokaisen kappaleen kokema kiihtyvyys, toiseksi lasketaan niiden uudet sijainnit ja viimeiseksi niiden uusi nopeus.

Tietorakenteet

Ohjelmassa eniten käytetty tietorakenne on lista. Esimerkiksi taivaankappaleita kuvastavat oliot tallennetaan muuttuvan kokoiseen listaan. Kappaleen koordinaatit, nopeus ja kiihtyvyys ovat olioita, jotka sisältävät muuttujat kunkin ominaisuuden x, y ja z -komponenteista. Nopeuden ja kiihtyvyyden osalla oliolla on myös muuttuja, joka kertoo vektorin pituuden eli kokonaisnopeuden tai -kiihtyvyyden.

Tiedostot

Ohjelman alkutilatiedostot ovat tekstitiedostoja, jossa on ensimmäisellä rivillä ilmoitettu, että tiedosto sisältää ohjelman tarvitsemat tiedot systeemin luomiseksi. Rivit on jaettu osiin käyttämällä kaksoispistettä (:). Välilyöntien määrä ei vaikuta tietojen lukemiseen.

Ensimmäisen rivin *SOLARSYSTEM*-tunniste on ainoa merkkikokoriippuvainen osa tiedostoa, joka kertoo, että ohjelma osaa lukea tiedoston. Rivin toinen osa kertoo tiedoston tyypin. Tällä hetkellä ohjelma tuntee vain yhden tiedostotyypin, mutta tämä mahdollistaa uusien tiedostotyyppien lisäämisen. Seuraavalla on tunniste *#System*, joka ilmoittaa, että rivillä kerrotaan systeemin nimi ja skaalausarvo. Näiden kahden rivin tulee olla olemassa ja aina kahtena ensimmäisenä rivinä.

Kahden ensimmäisen rivin jälkeen tulevat tiedot systeemin kappaleista. Kappaleiden järjestyksellä ei ole väliä, mutta jokaisen kappaleen tiedot tulee aloittaa rivillä ”*#NewObject*” ja lopettaa riviin ”*#*”. Pakolliset tiedot sisältävät: kaikki komponentit koordinaateista ja alkunopeudesta, värin tiedot sekä kappaleen nimi, massa ja säde. Valinnainen tieto siitä, onko kappale paikoillaan muiden suhteen, ilmaistaan rivillä ”*fixed:*”.

Massan, säteen, koordinaattien ja nopeuksien riveillä on kolme osaa. Ensimmäinen osa kertoo, mikä ominaisuus on kyseessä. Osa ei ole merkkikokoriippuvainen, mutta ne tulee kirjoittaa oikein. Toinen osa ilmoittaa ominaisuuden koon perusyksiköissä (kg, m, m/s) ja kolmas ilmoittaa

eksponentin. Esimerkiksi yksi kilometri x suunnassa ilmoitettaisiin ”coordinate_x : 1: 3”. Värin tiedoissa lukujen tulee olla nollan ja yhden välillä.

Skaalausarvo riippuu siitä, minkälaisia etäisyyksiä halutaan kuvata. Ohjelman piirtämisen ääriraja on 30 yksikköä origosta. Tästä, johtuen kuvaan piirtyvät vain kappaleet, joiden etäisyys on skaalattu tähän sopivaksi. Esimerkissä Maan etäisyys on noin 147,1e+9 ja skaalaus arvo 1e-10 sijoittaa Maan ”graafiselle” etäisyydelle 14,7, jolloin se on raja-arvon sisäpuolella.

Kappaleiden säteet on ilmaistu graafisen käyttöliittymän yksiköissä, sillä todellisia säteitä käytettäessä skaalausarvo kutistaisi säteet lähestulkoon olemattomiksi. Tällöin suuriakin kappaleita olisi hyvin vaikea ja pienempiä lähes mahdotonta havaita.

Esimerkki alkutilatiedostosta:

```
SOLARSYSTEM :InitialConditionFile
#System : Sun&Earth:10:-10
#NewObject
name: Sun
fixed:
mass: 1.989:30
coordinate_x: 0:0
coordinate_y: 0:0
coordinate_z: 0:0
velocity_x:0:0
velocity_y:0:0
velocity_z:0:0
radius:1:0
color:0:0:1
#
#NewObject
name: Earth
mass : 5.974:24
radius:0.2:0
coordinate_x:147.09090:9
coordinate_y: 0:0
coordinate_z: 8:8
velocity_x:0:0
velocity_y: 29300:0
velocity_z:0:0
color:1:0:0
#
```

Testaus

Ohjelman testauksessa yksi tärkeimmistä työkaluista on sen graafisen käyttöliittymän simulaatioikkuna. Sen valmistuttua pystyi tarkistamaan, laskeeko ohjelma gravitaation vaikutuksen uskottavasti, eli alkavatko oikeanlaisilla massoilla ja nopeuksilla simulaatioon sijoitetut kappaleet kiertämään keskustähteään.

Toinen tärkeä työkalu oli komentoikkunaan tulostettu informaatio esimerkiksi kappaleiden sijainnista, nopeudesta tai kiihtyvyydestä. Esimerkiksi eräs ongelma, jossa kuva kappaleista katosi simulaation oltua päällä jonkin aikaa, ratkesi käyttämällä tulostusta apuna.

Yksikkötestejä on vain ohjelman tiedostonluku luokalle, sillä ne voivat olla käyttäjän kirjoittamia ja siten ovat virhealtis osa ohjelmaa. Testit testaavat, että lukuluokka antaa oikean virheen, jos jokin tiedoston osa on virheellinen tai oleellista tietoa puuttuu.

Suunnitelmassa oli kaavailtu testit kappaleen sijainnin tarkistamiseen, mutta samasta alkutilanteesta lähtevä kappale päätyy aina samaan pisteeseen tietyn ajan jakson jälkeen, kunhan simulaation aika-askel pysyy vakiona. Tämän pystyin helposti tarkistamaan tulostamalla kappaleen sijainti tietoja ja ajamalla simulaatiota tietyn askelmäärän eteenpäin.

Puutteet & viat

Kaikkein suurin ja olennaisin puute on satelliittien lisääminen. Toiminto oli suunnitelmassa ja tehtävänannossa hyvin keskeisessä roolissa. Graafisen käyttöjärjestelmän luominen oli odotettua hankalampaa ja erityisesti piirtämisen opetteluun OpenGL:ää käyttäen kului huomattavasti enemmän aikaa kuin suunnitelmassa oli otettu huomioon. Tein kuitenkin päätöksen, että sujuva kappaleiden piirtäminen oli tärkeämpää kuin satelliittien lisäämistoiminto, sillä jälkimmäisestä ei olisi paljon iloa, jos kyseisten satelliittien liikettä ei voisi seurata. Olen varma, että saisin toiminnon toteutettua, jos jatkaisin projektia. Viimeisempänä ideana oli, että satelliitteja voisi tallentaa tiedostoon, josta voisi valita yhden ja käyttämällä hiirtä sijoittaa sen simulaatioon. Samalla tavalla voisi myös lisätä planeettoja tai tähtiä, ja ehkä jopa rakentaa oman systeemin ilman tarvetta systeemin lataamisen tiedostosta.

Tällä hetkellä kuvakulmaa voi käännellä hiirellä, mutta ohjelma ei anna mitään palautetta siitä, mistä kulmasta simulaatiota katsotaan. Tämä käy ongelmaksi, kun käyttäjä haluaisi johonkin tiettyyn kuvakulmaan tai palata alkuperäiseen. *Reset*-toiminto palauttaa tällä hetkellä alkuperäisen kuvakulman, mutta se on hyvin karkea ratkaisu. Jonkinlainen referenssi siitä, missä asennossa koordinaattiakselit ovat katsojaan nähden tai kentät joihin ohjelma tulostaisi kuvakulmat (atsimuutti ja korkeuskulma) olisi tuleva ratkaisu tähän ongelmaan.

Ohjelman tarkkuus ei ole paras mahdollinen. Tarkkuus on paljolti ohjelman nykyisessä toteutuksessa riippuvainen simulaation aika-askeleen suuruudesta. Pienemmillä aika-askelilla saavutetaan tarkempia tuloksia, mutta ohjelma käy kovin hitaaksi. Tämän ongelman huomaa, sitä paremmin mitä pienemmällä radalla kappale on, jolloin kappaleella on kaareutuneempi etenemissuunta ja ohjelman approksimaatio ei ole tarpeeksi tarkka pitääkseen nopeat, pienillä radoilla kulkevat kappaleet siellä mihin ne kuuluvat. Esimerkiksi alkutilanteessa *ThreeRocks*, joka sisältää Maan, Venuksen ja Merkuriuksen huomaa, kuinka erityisesti Merkurius alkaa harhailla kauemmaksi kuin sen kuuluisi, mutta Maa pysyy melko hyvin omalla radallaan. Ongelman voisi korjata tarkemmalla laskennalla, mutta tällä hetkellä en tiedä kuinka sen voisi toteuttaa.

Alkutilanteita on tällä hetkellä vain kaksi ja niiden luominen on hieman työlästä, sillä kappaleiden sijainnit ja alkunopeudet täytyy laskea käsin. Ohjelmaa voisi laajentaa, siten että se sisältäisi alkutilanne-editorin, jossa ohjelman sisällä voisi sijoittaa planeettoja tähden ympärille ja antaa niille haluamansa massan ja nopeuden ja ohjelma laskisi tarkemmat yksityiskohdat (x, y, z – komponentit) ja tallentaisi tilanteen tiedostoon.

3 parasta ja 3 heikointa kohtaa

Hyvät kohdat

Mielestäni yksi paras osa ohjelmaa on, kuinka se piirtää kappaleet. Piirtäminen on mielestäni sujuvaa ja kuvakulman vaihtaminen puutteistaan riippumatta tuo kolmiulotteisuuden tunteen. Ottaen huomioon, että en ennen ole tehnyt graafisia käyttöliittymiä onnistuin siinä melko hyvin.

Simulaation nopeuden vaihtaminen on toinen suosikkini. Vaikkakin se on yksinkertainen. eikä nopeutta voi vaihtaa kesken simulaation on se mielestäni hyvä toiminto.

Huonot kohdat

Toimintojen puute on mielestäni ohjelman suurin heikkous. Suunniteltuja toimintoja on paljon enemmän kuin toteutettuja, mutta aika ei riittänyt kaikkien lisäämiseen.

Alkutilainteiden luominen on hankalaa ja kuten aiemmin jo mainittu asian voisi korjata lisäämällä editorin ohjelmaan tai muokkaamalla ohjelman sellaiseksi, että alkutilanteita ei tarvittaisi vaan kaikki kappaleet lisättäisiin simulaatioon erikseen.

Poikkeamat

Suunnitelmasta poiketen simulaatio toimii päälle/pois periaatteella. Eli kun simulaation käynnistää se pyörii niin kauan kuin ohjelma on päällä tai kunnes käyttäjä päättää pysäyttää sen. Suunniteltu toimintaperiaate oli, että ohjelma simuloisi vain käyttäjän antaman ajan eteenpäin pysähtyisi odottamaan lisäohjeita. Mielestäni ohjelmaa on mukavampi käyttää ja seurata ilman, että sille tarvitsee tehdä mitään. Lisäksi tämä ratkaisu yksinkertaisti käyttöliittymää.

Suunnitelman ajankäyttöarvio oli virheellinen piirtämiseen arvioidun ajan suhteen, sillä siihen kului huomattavasti odotettua enemmän aikaa. Laskennan osalta arvio osui melko lähelle totuutta.

Aikataulu

Ensimmäisenä osiona toteutuivat kappaleiden liikkeen laskemiseen tarvittavat luokat 27. marraskuuta sekä viimeinen liikkumiseen liittyvä metodi 31. marraskuuta. Viimeinen suuri muutos liikkeen laskentaan toteutui 3. huhtikuuta, jolloin lisättiin vaihtoehto kappaleen kiinnittämiseen paikoilleen.

Seuraavaksi toteutettiin käyttöliittymän perusikkuna sekä piirtämiseen tarvittava *MyGLWidget*-luokka. Suurin osa projektiin käytetystä ajasta kului näiden kahden ja erityisesti piirtoluokan toteuttamiseen. 19. huhtikuuta mennessä molemmista oli kehitetty prototyypit, joilla ikkunaan sai piirrettyä kappaleita.

25. huhtikuuta ohjelma alkoi olla jo viimeisessä muodossaan ja tämän jälkeen ohjelmaan on lisätty muutamia lisäominaisuuksia ja parannuksia. Lisäksi koodia optimoitu ja ylimääräisiä funktioita poistettu 30.4-5.5.

Arvio

Suunnitelmassa asetettu vaikeustaso oli keskivaikea. Ohjelman puutteista johtuen koen, että tämän vaikeustason tavoitteita ei kuitenkaan täysin saavutettu. Vaikka ohjelmalla on keskivaikean toteutuksen vaatima graafinen käyttöliittymä, siinä on hyvin vähän toimintoja ja yksi keskeisimmistä, eli satelliitin lisääminen, puuttuu. Muuten ohjelma toimii suunnitellusti, mutta tuntuu yhä keskeneräiseltä, ja koodista löytyykin muutama keskeneräinen funktio, jotka eivät tällä hetkellä tee mitään ohjelman suorituksessa.

Ohjelma on kuitenkin rakennettu niin, että puuttuvien toimintojen lisääminen onnistuisi helposti. Eri kokonaisuudet on jaettu omiin luokkiinsa, joten erityisesti pienten muutosten tekeminen on helppoa, sillä muutos vaikuttaa lähinnä yhteen luokkaan. Toteutetuissa luokissa olisi vielä parannettavaa virtaviivaisuuden osalta, mutta en muuttaisi ohjelman perusrakennetta. Tosin *BodyGraphicsItem*-luokan voisi poistaa ja sisällyttää tiedot siitä, miten kappale piirretään suoraan *CelestialBody*-luokkaan, jolloin ylimääräiseksi käyneen luokan voisi poistaa.

Viitteet

http://www.tutorialspoint.com/pyqt/ , pvm. 02.05.2017

http://doc.qt.io/ , pvm. 02.05.2017

https://docs.python.org/3/ , pvm. 05.05.2017

http://stackoverflow.com/questions , pvm. 30.04.2017

Mark Segal, Kurt Akeley: The OpenGL Graphics System: A Specification (Version 1.4), 24. heinäkuuta 2002, https://www.khronos.org/registry/OpenGL/specs/gl/glspec14.pdf

https://nssdc.gsfc.nasa.gov/planetary/factsheet/