

Aurinkokuntasimulaattorin projektidokumentti – Ohjelmoinnin peruskurssi Y2

1. Henkilötiedot

Kristian Salo

481014

Sähkötekniikka ja elektroniikka, 2015

1.4.2016

2. Yleiskuvaus

Työn päämääränä oli toteuttaa Newtonin lakeihin perustuva aurinkokuntasimulaattori. Lopullinen toteutunut ohjelma onkin suurilta osin tavoitteiden mukainen. Ohjelma laskee taivaankappaleiden ratojen pisteitä sekä simuloi niiden liikettä, kunnes tapahtuu törmäys tai simuloitava ajanjakso kuluu loppuun. Perusaurinkokunnan kaikilla elementeillä (aurinko ja planeetat) on nopeus ja paikka, jotka on asetettu niin, että systeemi toimii mahdollisimman todenmukaisesti ja järkevästi.

Käyttäjän on mahdollista käyttää omaa aurinkokuntatiedostoaan simulaation lähtökohtana tai lisätä annettuun aurinkokuntaan kappaleita, jotka otetaan simulaatioon mukaan. Lisättävästä objektista syötetään nimi, massa, säde, väri, nopeus ja sen sijainti kolmiulotteisessa avaruudessa. Lisäksi käyttäjä valitsee simulaatiossa käytettävän aika-askeleen sekä kokonaiskeston.

Olen pyrkinyt toteuttamaan ohjelman vaikealla vaatimustasolla. Ohjelmassa on graafinen käyttöliittymä alkutietojen syöttämistä varten ja itse simulaatio on toteutettu graafisesti nopeus- ja kiihtyvyyksvektoreiden kanssa. Myös matemaattinen laskenta on optimoitu tarkemmaksi ja nopeammaksi Runge-Kutta 4 –menetelmän avulla sekä testitiedostossa ohjelmaa testataan Lagrangen pisteissä. Lisäksi olen lisännyt ohjelmaan muutamia mielestäni hyödyllisiä toimintoja, kuten mahdollisuuden pysäyttää tai zoomata simulaatiota.

3. Käyttöohje

Ohjelma käyttää pygamen versiota 1.9.2 ja PyQt:n versiota neljä. Pygamen muidenkin versioiden pitäisi luultavasti toimia, mutta graafiset ikkunat vaativat PyQt:n version neljä. Ohjelma käynnistetään suorittamalla moduulista main löytyvä pääfunktion main. Tämän jälkeen ohjelma huolehtii itse ikkunoiden avaamisesta ja simulaation pyörittämisestä, jolloin käyttäjän vastuulle jää oikeiden lähtötietojen syöttäminen ja simulaatiosta nauttiminen. Lähtötiedoissa käyttäjä voi alustaa aurinkokunnan mukana tulevalle aurinkokuntatiedostolla tai käyttää pohjana omaa tiedostoaan, josta luetaan simulaatioon lisättävät kappaleet.

Käyttäjä valitsee myös simulaatioaskeleen pituuden, simulaation keston ja voi halutessaan lisätä uusia kappaleita simulaatioon. Tietojen syöttämisen pitäisi olla melko triviaalia lukuun ottamatta sijainnin ja nopeuden arvoja, jotka syötetään muodossa xx,yy,zz eli kolme pilkulla toisistaan erotettua arvoa. Koordinaatiston origo on keskellä näyttöä, positiivinen x-akseli on simulaatioikkunassa oikealle, positiivinen y-akseli alaspäin sekä positiivinen z-akseli ruutuun kohti. Ohjelma osaa tulkita kymmenen potensseja e-notaation avulla, eli esimerkiksi $5.92e24$ on käypä arvo maan massalle.

Itse simulaatio käynnistyy alkutietojen syöttämisen jälkeen. Simulaatioikkunassa on ohjetekstit käyttäjän mahdollisille toiminnoille. Käyttäjä voi zoomata simulaatiota, muokata sen nopeutta, pysäyttää simulaation, muuttaa aika-askeleen pituutta, siirtyä kokonäytölle, piilottaa vektorit tai piilottaa infotekstit. Simulaatio päättyy joko ajan kuluessa loppuun tai törmäyksen tapahtuessa. Tällöin avautuu ikkuna, josta käyttäjä voi tallentaa juuri suorittamansa simulaation, käynnistää uuden simulaation tai lopettaa ohjelman suorituksen.

4. Ohjelman rakenne

Ohjelman toiminnan kannalta keskeisimmät luokat ovat Simulation ja Object. Object-luokka kuvaa yksittäistä taivaankappaletta, ja sen metodien avulla lasketaan yhden taivaankappaleen sijainti ja nopeus aika-askeleen Δt kuluttua. Käytännössä laskenta tapahtuu kutsumalla NewData-metodia, joka käyttää apunaan muita luokan metodeja: NewAcceleration-metodilla lasketaan kiihtyvyys, joka kääntyy sijainnin ja nopeuden arvoiksi First- ja OtherDerivatives – metodien avulla. Laskenta perustuu RK4-menetelmään, joka on selostettu tarkemmin algoritmit-kohdassa. NewData palauttaa Data-olion, jolla on uudet sijainnin, nopeuden ja kiihtyvyyden arvot.

Simulation-luokka kuvaa koko simulaatiota ja vastaa myös sen pyörittämisestä törmäykseen tai ajan loppumiseen saakka. Simulaatio käynnistyy kutsumalla metodia simulate, joka simuloi aurinkokuntaa muita luokan metodeja apunaan käyttäen. Jokaisella iteraatiokierroksella kutsutaan ensin metodia CalculateNextFrame, joka laskee Object-luokan NewData-metodin avulla kaikille kappaleille uudet sijainnit ja nopeudet. Tämän jälkeen metodilla planetsCollide tarkistetaan, ettei uusissa koordinaateissa tapahdu törmästä ja metodilla checkEvents huomioidaan käyttäjän mahdolliset toimet. Mikäli simulaatio jatkuu näiden tarkastusten jälkeen, kutsutaan metodia plotNextFrame, joka piirtää kappaleet uusiin sijainteihinsa uusilla kiihtyvyyksillä ja nopeusvektoreiden arvoilla.

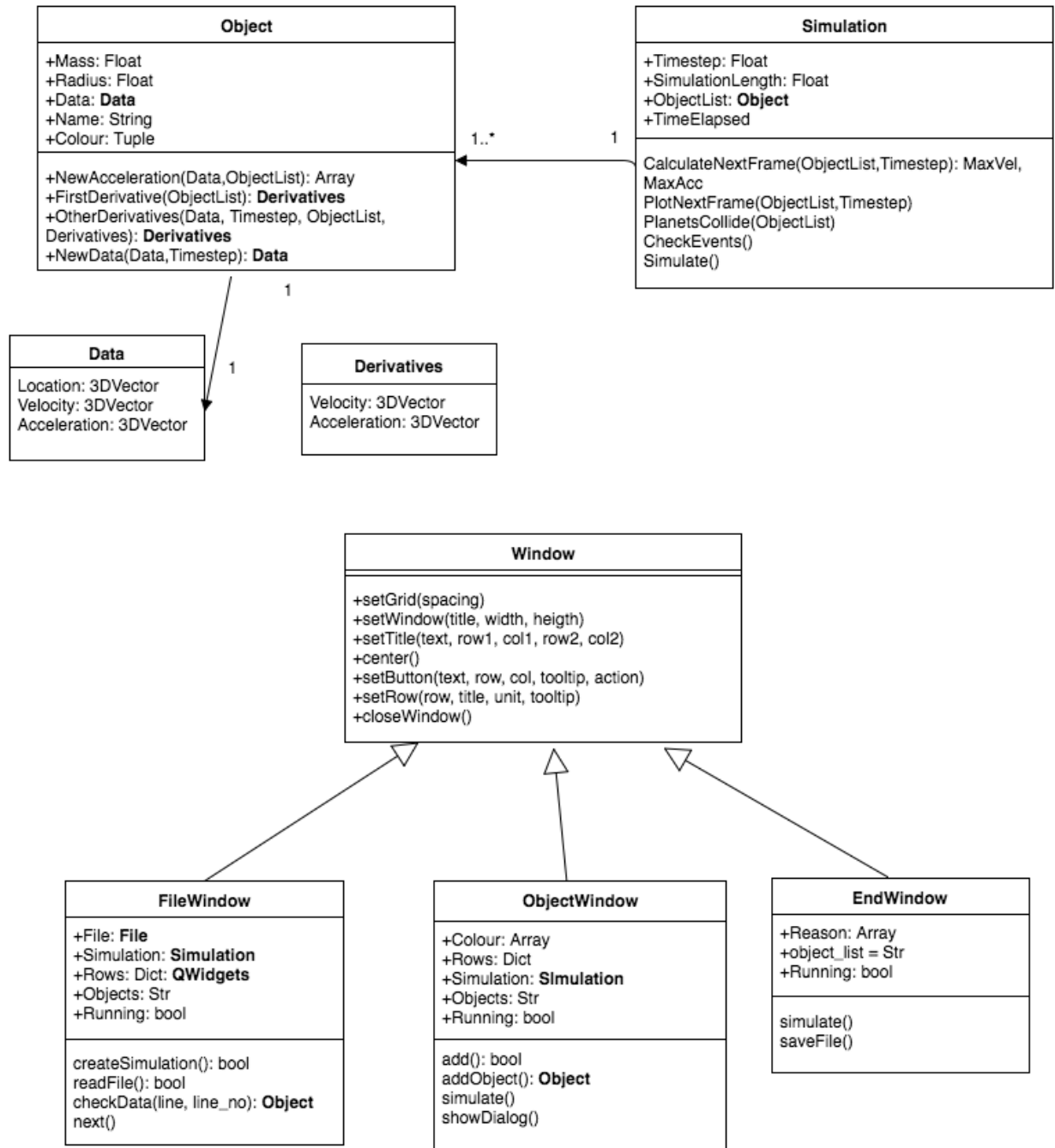
Graafiset ikkunat FileWindow, ObjectWindow ja EndWindow perivät kaikki luokan Window, jossa on metodeja nappien, tekstilaatikoiden ja muiden tarvittavien asioiden lisäämiseksi ikkunaan. Ilman yhteistä yläluokkaa kaikissa kolmessa graafisessa luokassa olisi ollut hyvin paljon identtistä koodia, jolloin tämän ratkaisun avulla koodista tuli elegantimpaa ja kopioi/liitä-tyyppistä menettelyä ei tarvittu yhtään.

FileWindow-ikkuna sisältää kentät aika-askeleelle ja simulaation pituudelle, sekä mahdollisuuden antaa tiedostoselaimesta polku aurinkokuntatiedostoon, jolla simulaatio alustetaan. Next-nappia painamalla käyttäjä kutsuu next-metodia, joka puolestaan kutsuu createSimulation-, readFile- ja checkData-metodeita. Näiden avulla luodaan uusi Simulation-olio, ja aurinkokuntatiedoston mukaiset Object-oliot, jotka lisätään Simulation-olion kenttään list. Next-metodi avaa tietojen ollessa oikein ObjectWindow-ikkunan ja sulkee auki olleen FileWindow-ikkunan.

Käyttäjä syöttää avautuvaan ObjectWindow-ikkunaan haluamiensa simulaatioon lisättävien kappaleiden tiedot. Add-napin painalluksella syötetyt tiedot tallennetaan ja kentät tyhjenevät uuden kappaleen lisäämistä varten. Simulation-napista puolestaan lisätään annetun kappaleen tiedot (tai ei mitään, mikäli käyttäjä ei ole syöttänyt mitään) ja käynnistetään simulaatio. Annetuista tiedoista luodaan Object-olio metodilla addObject, jota kutsutaan nappien painallukseen liitetystä metodeista add ja simulate.

Simulaation päättyessä joko ajan loputtua tai törmäyksen tapahtuttua aukeaa luokan EndWindow ikkuna, jossa käyttäjälle kerrotaan simulaation loppumisen syy. Käyttäjä voi halutessaan tallentaa juuri näkemänsä simulaation tiedostoon, jonka jälkeen hän voi sammuttaa ohjelman tai aloittaa alusta käynnistämällä uuden simulaation. Simulate-metodi sammuttaa EndWindow-ikkunan ja käynnistää ohjelman suorituksen alusta, kun taas saveFile-metodi avaa käyttäjälle tiedostodialogin ja tallentaa simulaatitiedoston haluttuun hakemistoon.

Koko ohjelman suorituksesta vastaa main-moduulin funktio main. Funktio sisältää silmukan, josta kutsutaan ohjelman luokkia järjestyksessä: FileWindow (josta aukeaa myös ObjectWindow), Simulation, ja EndWindow. Mikäli käyttäjä valitsee EndWindowsta uuden simulaation, alkaa silmukan suoritus uudestaan, mutta mikäli ohjelma sammutetaan jossain välivaiheessa, loppuu myös silmukan suoritus kokonaan. Ohjelma siis käynnistetään suorittamalla funktio main.



5. Algoritmit

Ohjelman simulaatio perustuu Newtonin lakien avulla laskettaviin voiman, kiihtyvyyden, nopeuden ja paikan arvoihin sekä niiden uudelleenevaluointiin halutun aika-askeleen välein. Koska tehtävä ratkaistaan kolmessa ulottuvuudessa, käytin paikan, nopeuden ja kiihtyvyyden arvoina pygamen 3D-vektoreita. Perusideana on laskea yhteen taivaankappaleeseen muista taivaankappaleista kohdistuvien voimien summa \mathbf{F}_{kok} yhtälön $\mathbf{F}_n = \frac{\gamma * m_1 * m_2}{r^2}$ avulla, josta saadaan kokonaiskiihtyvyys $\mathbf{F} = m\mathbf{a}$ mukaisesti.

Kokonaiskiihtyvyyden avulla saadaan laskettua kappaleen uusi sijainti $\mathbf{x} = \mathbf{x}_0 + \mathbf{dx} * dt$ ja nopeus $\mathbf{v} = \mathbf{v}_0 + \mathbf{dv} * dt$. Laskuissa käytän apuna Runge-Kutta 4 -metodia, joka on tarkempi kuin suoraan Eulerin menetelmällä eli edellisten yhtälöiden avulla laskeminen. RK4 hyödyntää lopulta aivan samoja paikan ja nopeuden yhtälöitä, mutta siinä evaluoidaan ensin tarkemmin em. yhtälöissä uuden sijainnin ja nopeuden laskemiseen käytettyjä nopeuden \mathbf{dx} (sijainnin aikaderivaatta) ja kiihtyvyyden \mathbf{dv} (nopeuden aikaderivaatta) arvoja. Käytännössä evaluointi tapahtuu laskemalla neljä erillistä derivaattojen arvoa aika-askeleen sisällä, mikä huomioi derivaattojen arvon muuttumisen dt :n aikana. Näistä neljästä derivaattojen arvosta lasketaan painotettu keskiarvo, jolloin saadaan lopulliset, Eulerin menetelmässä käytettävät \mathbf{dx} sekä \mathbf{dv} . Olen toteuttanut RK4-menetelmän paljolti linkeistä löytyvän Glenn Fiedlerin RK4-esittelyn mukaisesti.

Edellä mainittujen fysiikan algoritmien lisäksi ohjelmassa on myös muutamia muita, enemmän simulaation toteutukseen liittyviä algoritmeja. Törmäyksen tarkistaminen tapahtuu tutkimalla, ovatko jotkin kaksi simulaation kappaletta etäisyydellä, joka on pienempi kuin kappaleiden säteiden summa. Lisäksi simulation-luokassa on metodi arvon skaalaamiseen annetulta väliltä uudelle, halutulle välille. Skaalaaminen perustuu Arduinon ohjelmointikielessä olevaan map-funktioon, jossa uusi arvo x_2 saadaan kaavasta $x_2 = \frac{(x_1 - in_min) * (out_max - out_min)}{(in_max - in_min)} + out_min$. Kaavassa arvo x_1 skaalataan väliltä (in_min, in_max) välille (out_min, out_max) . Jokaisella iteraatiokierroksella selvitetään suurimmat kiihtyvyyden sekä nopeuden arvot, joiden avulla kappaleiden nopeus- sekä kiihtyvyysvektorit skaalataan edellä esitellyn metodin avulla järkevän mittaisiksi.

6. Tietorakenteet

Ohjelmassa on käytössä pääasiassa muuttuvatilaisia tietorakenteita niiden uudelleenmuokattavuuden takia. Esimerkiksi yksittäisen taivaankappaleen sijainti, nopeus ja kiihtyvyys on ilmaistu pygamen 3D-vektoreina, joita voi muokata simulaation edetessä ja arvojen muuttuessa. 3D-vektoriolionne oli myös paljon valmiita apufunktioita, jotka helpottavat kappaleiden käsittelyn aritmetiikkaa.

Myös luokan Simulation attribuuttit list ja radius_list ovat listoja. List sisältää kaikki simulaatiossa olevat kappaleet. Lista kuitenkin muutetaan monikoksi simulaation alkaessa, kun sinne ei enää lisätä uusia olioita. Monikko vie ohjelman suorituksen aikana hieman vähemmän tilaa muistista, koska sitä ei voida muokata. Radius_list on kuitenkin lista koko simulaation ajan, koska sen arvoja pitää muokata uudelleen, mikäli käyttäjä zoomaa simulaatioikkunaa, jolloin myös piirrettyjen kappaleiden koko muuttuu.

Graafisissa ikkunaluokissa on käytössä sanakirja, joka sisältää ikkunassa olevia nappeja ja tekstikenttiä. Sanakirjan avulla näihin olioihin on helppo päästä käsiksi mistä tahansa koodin kohdasta, jos esimerkiksi halutaan tietää käyttäjän tekstikenttään kirjoittama arvo.

7. Tiedostot

Aurinkokuntasimulaattorin voi halutessaan alustaa antamalla alkutietojen syöttämisen yhteydessä tiedoston, josta löytyvät simulaatioon lisättävät kappaleet tietoineen. Tiedosto on formaatiltaan csv-tyyppinen ja siinä yhdelle riville on kirjoitettu yhden kappaleen tiedot toisistaan pilkulla erotettuna. Tiedot syötetään seuraavassa järjestyksessä: nimi, massa, säde, väri, sijainti ja nopeus. Väri, sijainti ja nopeus koostuvat sisältävät kukin kolme arvoa, värin tapauksessa rgb-värikoodit ja nopeuden sekä sijainnin kohdalla x-, y- ja z-koordinaatit. Mallirivi voisi näyttää seuraavalta: planeetta,12e23, 6e5, 255, 255, 0, 34e2, 0, 311, 94e2, 23e4, 46. Rivillä on siis aina oltava 12 pilkulla toisistaan erotettua arvoa, muuten ohjelma ei lisää yhtäkään alkutiedoston kappaletta simulaatioon ja ilmoittaa käyttäjälle virheellisestä tiedostosta.

8. Testaus

Ohjelmaa testattiin koko koodausprosessin ajan tasaisin väliajoin. Aluksi tein enemmän yksikkötestejä, jotka löytyvät koodin mukana tulevasta testiluokasta test. Yksikkötesteillä testasin yksittäisiä metodeja, mm. kiihtyvyyden laskentaa, skaalausta ja törmäyksen tunnistamista. Testitapaukset olivat melko yksinkertaisia, ja niillä varmistuin siitä, että metodit toimivat pääpiirteittäin oikein.

Yksikkötestaaminen ei kuitenkaan konseptina sovi koko simulaation testaamiseen, jonka takia isomman mittakaavan testaaminen tapahtui käytännössä ajamalla simulaatiota erilaisilla kokoonpanoilla. Yksi selkeä ja yksinkertainen testitapaus on kahden kappaleen systeemi, jossa pieni massa laitetaan kiertämään suurempaa massaa nopeudella, jonka seurauksena kappaleen pitäisi päätyä ympyräradalle. Ohjelman toteuttama rata on hyvin lähellä täydellistä ympyrää (virhe < 1.5%), mutta aika-askeleen kasvattaminen suuremmaksi lisää epätarkkuutta heti huomattavasti.

Testasin myös ohjelmaa Lagrangen pisteiden avulla, kuten tehtävänannossa oli käsketty. Lagrangen pisteet ovat kohtia, joissa pienimassainen kappale pysyy paikallaan suhteessa kahteen suurempaan kappaleeseen. Pisteitä on kaiken kaikkiaan viisi, joista kaksi on oikeilla alkuarvoilla melko pysyviä ja loput kolme hyvin alttiita epätasapainolle. Sisällytin omaan testiluokkaani tapauksen, jossa Merkurius kiertää aurinkoa systeemissä, jossa on lisäksi kolme Lagrangen pistettä. Simulaatiossa pisteet pysyivät melkein paikallaan, mutta pientä muutosta (virhe < 2%) sijainnissa oli havaittavissa – syy on todennäköisesti epätarkkoissa alkuarvoissa.

9. Ohjelman tunnetut puutteet ja viat

Ohjelma heittää aina simulation-luokan olion luonnin yhteydessä (ainakin omalla Macillani, muista koneista en ole varma) vikailmoituksen, joka varoittaa resurssien käytöstä. Vikailmoitusten mukaan olen avannut jonkin tiedoston, jota en ole sulkenut. Koodissa on

kuitenkaan avaa mitään tiedostoa, vaan teen pelkästään pakolliset pygameen liittyvät alustukset, jolloin kyse on ilmeisesti pygamen sisäisestä ongelmasta. Käytännössä nämä sulkemattomat tiedostot siis kuluttavat hieman resursseja, mutta en osannut itse (eikä kumpikaan kahdesta assarista, joilta asiaa kysyin) poistaa vikailmoitusta.

Simulaattori ei aina tunnista kappaleiden törmäystä, vaikka ne selvästi fysikaalisessa mielessä törmäisivätkin. Tämä voi tapahtua, mikäli sekä aika-askele, että kappaleiden nopeudet ovat suuria lukuja. Tällöin kappaleet ovat yhdellä aikahetkellä hyvin lähellä törmäystä, ja aika-askeleen aikana ne hyppäävät toistensa lävitse, ilman että törmäyksen tunnistus algoritmi sitä huomaa. Ongelman olisi voinut korjata muokkaamalla törmäyksen tunnistusta niin, että se toimisi vektoreiden avulla ja huomioisi kappaleiden liikeratojen päällekkäisyyden myös aika-askeleen sisällä. Valitettavasti huomasin ongelman vasta lopputestauksen yhteydessä, eikä aikaa korjaamiselle jäänyt.

Ohjelma ei toimi fiksusti, jos simulaatioon sijoittaa vain yhden kappaleen. Tällöin origo ei ole näytön keskellä, vaan näytön vasemmassa yläkulmassa. Epäloogisuus johtuu koordinaattien skaalausfunktioista, joka perustaa skaalauksen suurimpaan koordinaattien arvoon. Mikäli simulaatiossa on vain yksi kappale sijainnissa (0,0,0), on suurin arvo 0, ja skaalaus ei toimi oikein, eikä simulaatioikkunaa voi myöskään zoomata, koska lähtöarvo on 0. Ongelman olisi voinut korjata huomioimalla tämän erikoistapauksen skaalausfunktion koodissa, mutta en ehtinyt sitä itse korjata, ilman että muu koodi meni korjausyrityksestä sekaisin.

10. 3 parasta ja 3 heikointa kohtaa

Mielestäni ohjelman hyviä puolia ovat selkeä koodi ja simulaatioon lisätyt itse keksityt lisäominaisuudet. Pilkoin tehtävät mahdollisimman selkeiksi kokonaisuuksiksi yksittäisille metodeille ja koetin näin tehdä koodista selkeämpää. Pyrin lisäksi samalla välttämään identtisten rivien löytymistä monesta eri kohdasta, kuten esimerkiksi graafisissa ikkunoissa olisi ilman perintää käynyt. Graafiset ikkunat olivatkin mielestäni hyvin toteutettuja ja toimivia.

Lisäsin myös mielestäni ohjelmaan onnistuneita ja käytännöllisiä lisäominaisuuksia, kuten simulaatioikkunan informaatiotiedot ja mahdollisuus muokata parhaillaan käynnissä olevan simulaation parametrejä. Nämä mahdollistavat käyttäjälle simulaation muokkaamisen ja muokkausten tuloksen seuraamisen live-ajassa. Lisäksi mahdollisuus tallentaa ajettu simulaatio csv-formaattiin ohjelman päättyessä on mielestäni toimiva lisäys, joka tehostaa testaamista ja mahdollistaa hyvän simulaation toistamisen uudelleen.

Parantamisen varaa ohjelmaan jäi erityisesti suoritustehon optimoinnin osalta. Object-luokan koodia olisi varmasti voinut muokata tehokkaammaksi käyttämällä silmukoita ja näin vähentää lähes identtisten rivien kirjoittamista uudelleen. Approksimaatioiden (esim. satelliitti laskisi radan vain lähimmän kappaleen massasta, eikä huomioisi muita) avulla laskentaa olisi saanut kevennettyä. Lisäksi suoritustehoa olisi voinut parantaa esimerkiksi huomioimalla aurinkokunnan identtisyyden, jolloin jokaiselle kappaleelle ei olisi tarvinnut laskea kaikkia gravitaatiovoimia uudelleen.

Ohjelma käy myös Simulation-luokassa objectlist-listaa lävitse moneen kertaan eri metodeissa, mikä ei varmasti ole suoritustehon kannalta fiksuin vaihtoehto. Tämän takia simulaation nopeutta ei saa kasvatettua kovin suureksi ilman, että kasvattaa aika-askelta ja samalla myös

epätarkkuutta. Silmukoita olisikin voinut yhdistellä suuremmiksi kokonaisuuksiksi, mutta ajan puutteen takia tämä jäi vielä tulevaisuuden kehityskohteeksi.

Myös alkutiedoston olisi voinut toteuttaa Python-koodina, jolloin sitä olisi ollut helpompi käyttää itse ohjelman toteutuksessa. Tällöin siihen olisi myös voinut sisällyttää enemmän parametrejä, joilla voisi kätevästi muokata monia simulaation ominaisuuksia. Tämä olisi nopeuttanut ohjelman testaamisesta ja ajamista eri kokoonpanoilla.

11. Poikkeamat suunnitelmasta

Pysyin projektin toteutuksessa melko hyvin suunnitelmani mukaisessa aikataulussa. Toteutus venyi hieman suunnitellusta aikataulusta muiden koulutehtävien takia, mutta järjestys pysyi melko samana. Ensin toteutin Object- ja Simulation -luokat, joista siirryin graafisin luokkiin. Ajankäyttö oli lopulta hieman nopeampaa kuin suunnitelmassa olin arvioinut (suunnitelma ~ 50h, lopullinen ~40h), mutta se jakautui eri luokkien välille tasaisesti, kuten oli suunnitellutkin. Mikään osa ei ollut erityisen työläs, vaan kaikki vaativat oman, lähes yhtä suuren osansa kokonaistyöstä. Yleistrendi oli, että itse koodin kirjoittaminen oli nopeaa, mutta toisinaan yksittäisten ongelmakohtien kanssa painimiseen saattoi kulua suurikin aika.

12. Toteutunut työjärjestys ja aikataulu

Viikko 8

- Luokan Object kirjoittaminen kokonaan
- Apuluokkien Data ja Derivatives kirjoittaminen
- Yksikkötestejä luokan Object RK4-metodeille

Viikko 9

- Luokan Simulation kirjoittaminen niin, että ohjelma simuloi kahden planeetan liikettä
- Data-, Derivatives- ja Object-luokkien listojen tilalle 3D-vektorit

Viikko 10

- Ominaisuuksien lisäämistä Simulation-luokkaan: vektorit kirjaimineen, värit planeetoille ja zoomaaminen
- PyQt-käyttöliittymäikkunoiden kirjoittamisen aloittaminen

Viikko 11

- Simulaation pysäyttämisen mahdollisuus ja infotekstit Simulation-luokkaan
- PyQt-käyttöliittymäikkunoiden metodien koodausta ja yhdistäminen Simulation- ja Object-luokkien olioihin

Viikko 12

- Lisää ominaisuuksia Simulation-ikkunaan: mahdollisuus piilottaa infoteksti tai vektorit ja nähdä kulunut aika

- Graafisille ikkunoille perintä Window-luokasta, joka sisältää kaikille ikkunoille hyödyllisiä metodeja
- Main-funktio koko simulaation pyörittämiseen

Viikko 13

- Mahdollisuus nopeuttaa tai hidastaa simulaation nopeutta
- Projektiselostuksen aloittaminen

Viikko 14

- Simulaatio näyttää aika-askeleen arvon, jota voi myös muokata
- Testitapaukset Lagrangen pisteille ja ympyräradalle
- Mahdollisuus tallentaa tiedosto simulaation päättyessä
- Kaiken kokoaminen yhteen main-funktiossa
- Projektiselostuksen jatkoa

Viikko 15

- Lepoviikko ohjelmoinnista ja keskittymistä muihin kouluhommiin

Viikko 16

- Viimeistely ympyräradan ja Lagrangen pisteiden testeille
- Mahdollisuus siirtyä simulaatiossa koko näytölle
- Näytöllä näkyvien taivaankappaleiden koko riippuu zoomauksesta ja kappaleiden oikeasta säteestä
- Projektiselostuksen viimeistely

13. Arvio lopputuloksesta

Mielestäni ohjelma on kokonaisuutena melko onnistunut ja täyttää käyttötarkoituksensa. Graafiset ikkunat mahdollistavat selkeän tietojen syöttämisen ja itse simulaatio pyörii sujuvasti. Simulaatiota voi sen suorituksen aikana muokata monella eri tavalla, joka tekee katselusta mielekkäämpää. Lisäksi simulaation jälkeen annetut mahdollisuudet toistaa simulaatio tai tallentaa aurinkokuntatiedosto ovat mielestäni hyviä lisäyksiä.

Valitsemani luokkajako soveltui ongelman kuvaamiseen hyvin. Pyrin toteuttamaan luokat niin, että simulaatiota voi muokata helposti muuttamalla asiaa kuvaavan instanssimuuttujan arvoa. Myös uusien ominaisuuksien lisääminen simulaatioon pitäisi onnistua melko pienellä vaivalla. Graafisissa ikkunoissa jouduin tekemään hieman enemmän epäpäteviä ratkaisuja, koska en aivan ymmärtänyt PyQt:n ja perinnän toimintaa perustavalla tasolla. Tästä huolimatta myös niihin pitäisi pystyä lisäämään uusia ominaisuuksia suhteellisen helposti.

Selvästi suurin kehittämisen kohde ohjelmassa on suoritustehon optimointi. Olisin toivonut, että simulaation nopeutta olisi voinut nostaa suuremmaksi, mutta ainakaan oma koneeni ei pysty nostamaan fps:n arvoa kovinkaan korkealle raskaahkon laskennan takia. Aurinkokunnan symmetrisyyden huomiointi olisi nopeuttanut laskentaa puolella ja myös tilannetta yksinkertaistamalla ja approksimoimalla ohjelma olisi varmasti tullut tehokkaammaksi.

Jatkokehityksen kannalta hyvä lisäominaisuus olisi ollut liikkuvan kameran lisääminen ohjelmaan. Tällöin myös z-koordinaatit tulisivat huomioiduiksi, ja käyttäjä voisi vaikuttaa näkemäänsä simulaatioon enemmän, kuin vain zoomaamalla yhdestä pisteestä ulos- tai sisäänpäin. Myös alkutietotiedoston olisi voinut muokata python-koodiksi, jolloin sen avulla ohjelmaa saisi ajettua ja testattua todella nopeasti ilman graafisten käyttöliittymien käyttöä.

14. Viitteet

<http://stackoverflow.com/>

<http://zetcode.com/gui/pyqt4/>

<https://pythonspot.com/en/pyqt4/>

<http://www.pygame.org/docs/>

<http://nssdc.gsfc.nasa.gov/planetary/factsheet/>

<http://www.astronomynotes.com/tables/tablesb.htm>

<http://www.petercollingridge.co.uk/book/export/html/6>

<https://www.cs.ucsb.edu/~pconrad/cs5nm/topics/pygame/drawing/>

<http://gafferongames.com/game-physics/integration-basics/>

http://lasp.colorado.edu/education/outerplanets/orbit_simulator/

<http://orbitsimulator.com/formulas/LagrangePointFinder.html>

<https://www.python.org/doc/>

15. Liitteet

Liitteenä muutama kuva esimerkkiajosta ohjelmalla. Lähdekoodi löytyy Niksulan gitistä.

Solar system simulator

Insert the parameters for the simulation.

Simulation length: 365 days

Simulation timestep: 0.1 days

Solar system file: /Krisu/Desktop/SolarSystem.txt

Simulaation alkutietojen alustus ja aurinkokuntatiedoston polun antaminen.

Solar system simulator

Insert the information of the object.


Name: Exoplanet

Radius: 13e5 km

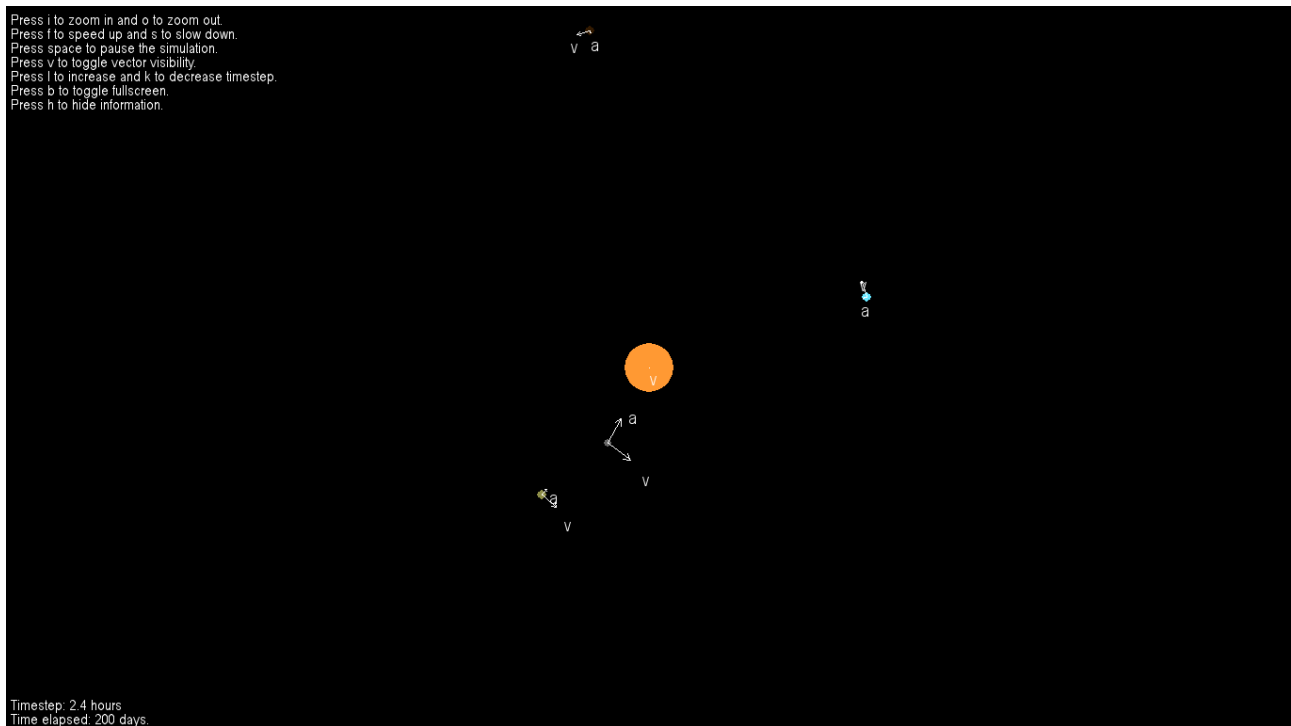
Mass: 20e25 kg

Velocity: 24,0,-2 km/s

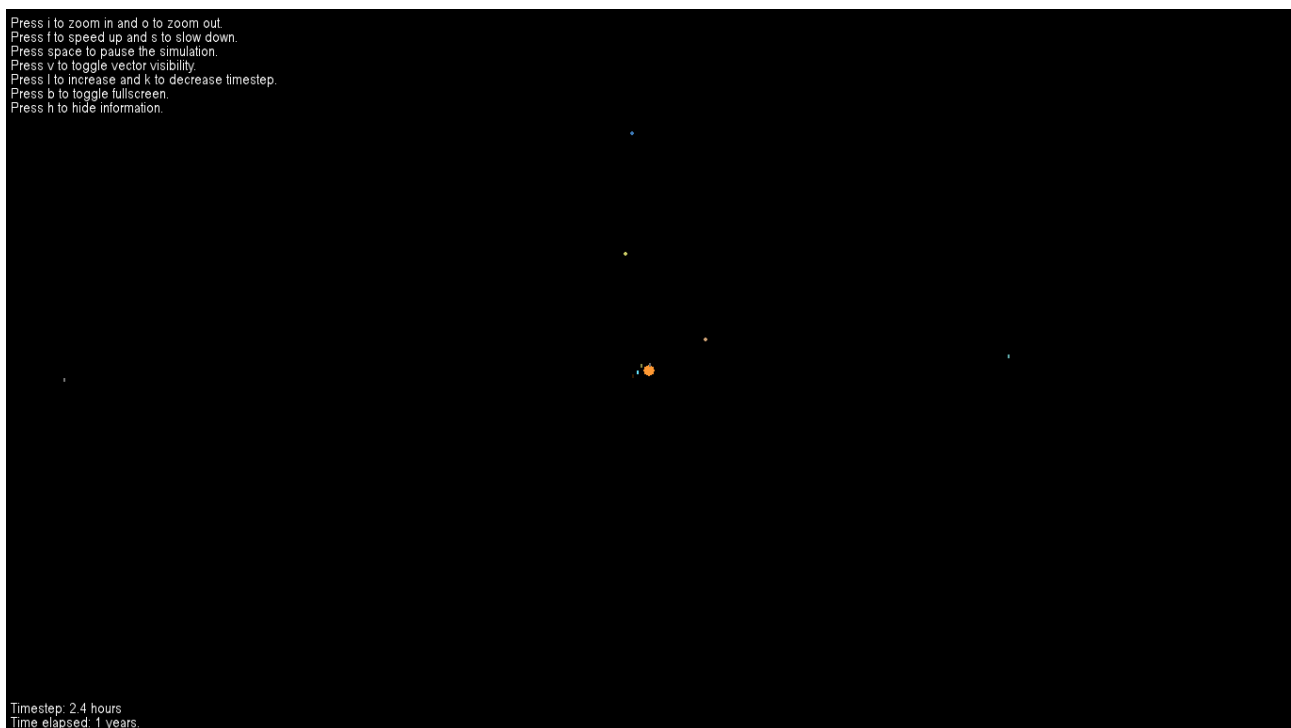
Location: 4,-2.245,0 au

Colour: 

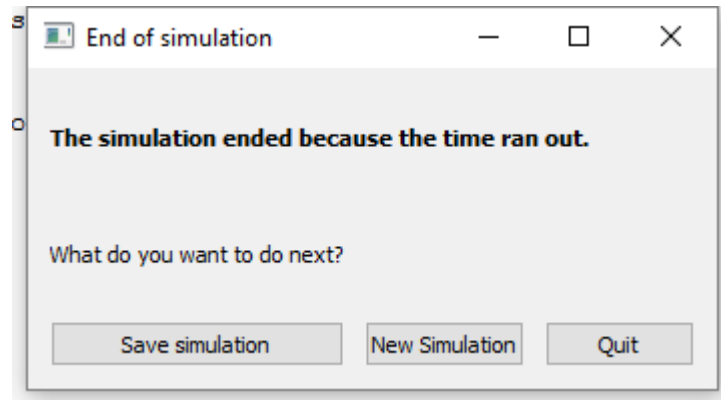
Kappaleen lisääminen simulaatioon.



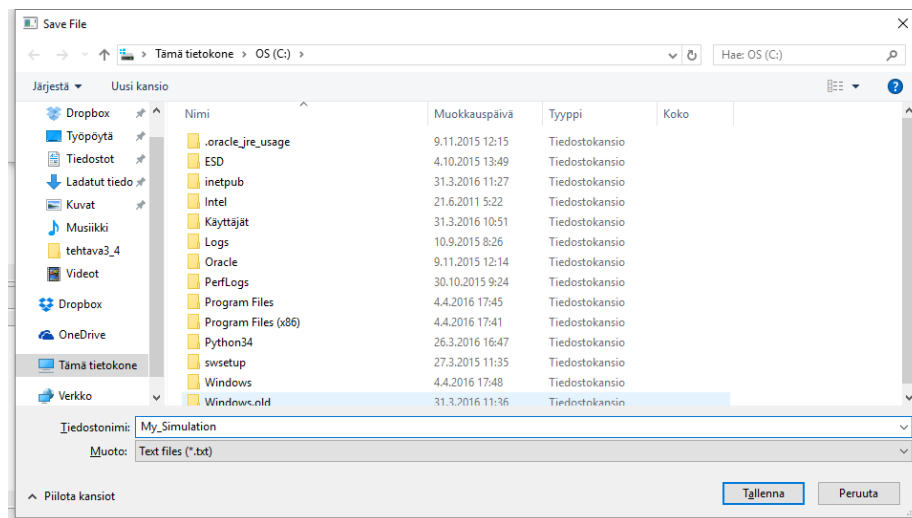
Aurinkokunnan simulointi käynnissä: kuvassa planeetat Auringosta Marsiin.



Sama simulaatio zoomattuna ulos niin, että koko aurinkokunta Plutoon saakka on näkyvillä. Vektorit on otettu pois näkyvistä kuvan selkeyttämiseksi.



Simulaation loputtua käyttäjä voi tallentaa simulaation ja sitten aloittaa uuden tai sulkea ohjelman.



Tiedoston tallennus onnistuu suoraan tiedostoselaimeen käyttäjän haluamaan sijaintiin.