Riku Autio
riku.autio@aalto.fi
585279
Tietotekniikka
Vuosikurssi 2016
25.4.2017

Reflections

1. Yleiskuvaus

Reflections on linssi- ja peilisimulaattori. Siinä asetetaan ensin simulaatioympäristöön erilaisia heijastavia objekteja. Objektien lisäyksen jälkeen voidaan asettaa yksi, tai useampia valonlähteitä halutulla suunnalla, jonka jälkeen voidaan aloittaa simulaatio. Heijastavat objektit ovat yksinkertaisuuden vuoksi kuvattu simulaatioikkunassa ulkoisesti suorina viivoina, vaikka olisivatkin koveria tai kuperia. Simulaation käynnistyessä valonlähteet lähettävät valonsäteen liikkeelle joka kimpoilee eri pinnoista ja piirtää valonsäteen reitin näkyviin.

Teknisessä suunnitelmassa esitettyihin ajatuksiin tuli muutama muutos. Tiedostoja on vain kolme: Simulator, Reflective ja Light. Myös ajatukseni tietorakenteista muuttui hieman, sillä käyttöön tuli Hashmap, sekä linssien ja koveran peilin taittokulmat toteutin eri tavalla kuin mitä alunperin suunnittelin. Vaihtoehtoina tälle projektiaiheelle oli tasot helppo, ja keskivaikea. Projektityö toteuttaa kaikki keskivaikeaan vaaditut ehdot.

2. Käyttöohje

Ohjelma käynnistetään käynnistystiedostosta Simulator.scala. Esiin aukeaa mustataustainen ikkuna. Vasemmasta yläreunasta löytyy “Start”-valikko, jonka alta löytyy “Help”, joka kertoo tärkeimmät ohjeet ohjelman käyttöön. Tästä valikosta löytyy myös valinnat: “Add random objects”, joka lisää satunnaisia objekteja näkymään, “Preset scenario: Box”, eräänlainen valmis asetelma, “Clear everything”, joka tyhjentää näkymän, “Save state” ja “Load state”, tallennus ja lataus, sekä “Quit”, joka sulkee ohjelman. Huomioitavaa on, että lataus ei tyhjennä ensin ikkunaa, vaan lisää tallennuksen objektit tämänhetkiseen tilaan.

Mustalle taustalle lisätään “objekteja”, eli heijastavia pintoja. Heijastavia pintoja on neljää erilaista, valkoisella kuvattu suora peili, punaisella kuvattu kovera peili, vihreällä kuvattu kupera linssi, sekä sinisellä kuvattu kovera linssi.

Näiden eri objektityyppien väliltä valitaan käyttämällä näppäimistön Z-näppäintä, joka valitsee seuraavan tyypin. Suora peili -> kovera peili -> kupera linssi -> kovera linssi -> takaisin alkuun. Asetettavan objektin kulma valitaan
näppäimillä X, 0, 1, 2, 3. X-näppäin kasvattaa kulmaa yhden kerrallaan kun numeronäppäimet asettavat kulmaksi 0, 90, 180 ja 270. Kulma ajatellaan ohjelmassa enemmänkin “suuntana”. 0 on suoraan oikealle, 90 suoraan alas, 180 suoraan vasemmalle, 270 suoraan ylös jne.

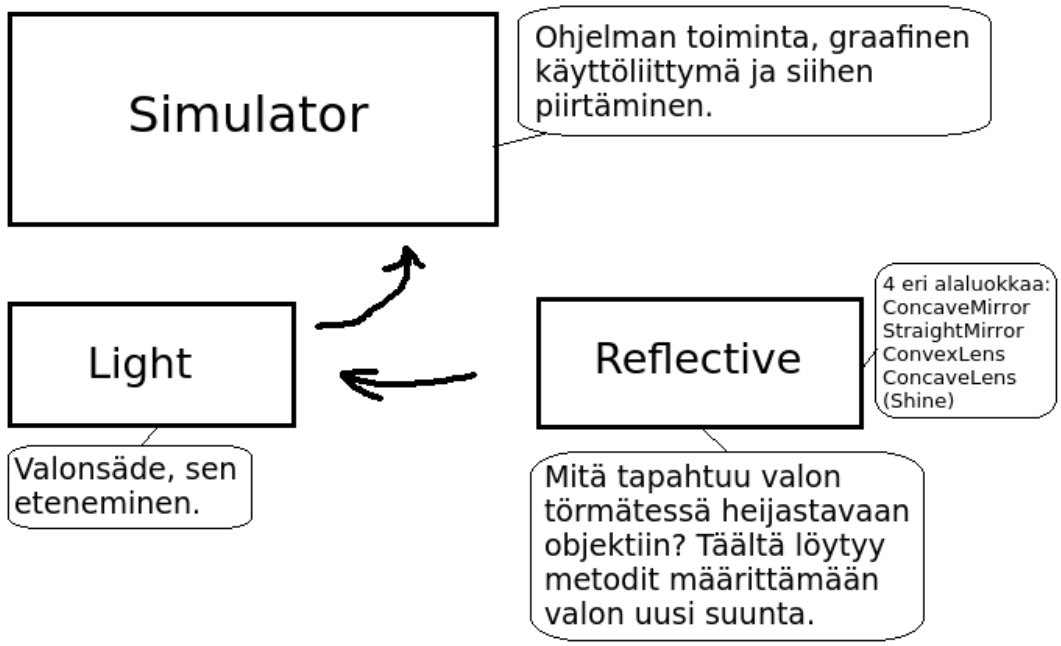
Kun haluttu objekti ja asetuskulma on valittu, klikataan hiirellä objekti paikoilleen. Se piirtyy siis hiiren klikkauksen sijainnista haluttuun suuntaan.

Objektien asettamisen jälkeen siirrytään L-näppäimellä valonlähteen asetukseen. Valonlähteelle kulma tarkoittaa samaa, valo piirtyy kulman suuntaan simulaation alkaessa, kuten yllä kuvattu. Myös valonlähteitä voi asettaa useampia kuin yhden.

Itse simulaatio aloitetaan painamalla G-näppäintä, jolloin alareunan status-teksti pyytää klikkaamaan mistä tahansa aloittaaksesi simulaation. Klikkauksen jälkeen valonsäteet piirtyvät ruudulle. Tämän jälkeen seuraava klikkaus poistaa valot, mutta pitää paikoillaan asetetut objektit. Objektit pystyy poistamaan “Start”-valikon kautta “Clear everything”-valinnalla. “Start”-valikosta löytyy myös ominaisuus lisätä satunnaisia objekteja ruudulle ja yksi valmis asetelma.

R-näppäimellä saa lisättyä myös alkuvalikon tavoin satunnaisia objekteja.

3. Ohjelman rakenne



Toisin kuin alkuperäisessä teknisessä suunnitelmassa, on lopullisessa ohjelmassa kolme tiedostoa: Simulator, Light ja Reflective. Simulator sisältää ohjelman käynnistyksen sekä kaikki graafisen käyttöliittymän toiminnot ja objektien sijoittamisen metodit.

Reflective on abstrakti luokka, joka sisältää kaikki neljä eri luokkaa objekteille: ConcaveMirror, StraightMirror, ConvexLens ja ConcaveLens. Näiden lisäksi pikkuluokan Shine joka on yksinkertainen tietämätön valonsäteen osa. Eri heijastavilta pinnoilta voi kysyä uutta kulmaa kun valonsäde törmää siihen.

Light on valonlähteen luokka, joka sisältää metodit valon liikkumiseen ja Shine-objektien lisäämiseen Simulatorin tietoon, että ne piirtyvät. Sisältää metodit “go”, jota kutsutaan Simulatorista kun on aika simuloida. Tämä kutsuu “advance”-metodia joka rekursiivisesti määrittää reitin valolle. Sen lisäksi apumetodit distance, getEndLong ja bresenham jotka auttavat määrittämään kulmia, valon kohteita ja reittiä.

Mielessä kävi myös ajatella heijastavien esineiden sijainnit jonkinnäköisenä Map[Reflective -> Set[(Int, Int)] -rakenteena, mutta huomasin nykyisen Map[(Int, Int) -> Reflective] -ratkaisuni toimivana enkä lähtenyt muokkaamaan sitä.

4. Algoritmit

Light:

distance: pythagoraan lause. Neliöjuuri(sijainnin erotus x- suunnassa ^2 + sijainnin erotus y-suunnassa^2). Tämä metodi tutkii etäisyyttä objektin alkupisteeseen ja auttaa siten määrittämään kuperille ja koverille esineille kimpoavan valon uuden kulman.

getEndLong:

Laskee valolle kohteen mihin se lähtee siirtymään. Ottaa parametrikseen alkukoordinaatit ja kulman ja laskee loppupisteen koordinaatit. loppuX = alkuX + etäisyys * kosini(kulma) ja loppuY = alkuY + etäisyys * sini(kulma).

bresenham:

Bresenhamin algoritmin kaltainen metodi joka kerää iteraattoriin kaikki pisteet kahden koordinaatiston pisteen välillä. Näitä käytetään sitten määrittämään valonsäteen pisteet. Pseudokoodi wikipediasta:

(alkupiste s0,t0, loppupiste s1,t1) (piirtämisen sijaan lisätään iteraattoriin)

```
if t0 < t1
  tstep := 1
else
  tstep := −1
```

```
virhe := w / 2
t := t0
for each s in [s0, s1]
  if s-akseli on x-akseli
     piirrä_piste(x, y)
  else
     piirrä_piste(y, x)
  virhe := virhe + h
  if virhe >= w
     t := t + tstep
     virhe := virhe − w
```

Lisää tästä viitteiden linkeissä.

Reflective-luokkien uudet kulmat/suunnat valolle:

Itse pähkäilty paperin kanssa sen tiedon ohella, että tulokulma ja lähtökulma suhteessa peilaavan pinnan normaaliin ovat yhtäsuuret.

Suora peili:
(360 - valonSuunta + 2 * (pinnanSuunta % 180)) % 360

Kupera peili:
Sama kuin yllä, mutta valon kulkusuunnasta riippuen lisätään tai vähennetään Light.distance-, ja Reflective.concave- apumetodien avulla määritetty määrä loppusuunnasta.

Kupera ja kovera linssi:
Valo ei peilaannu, joten siihen ainoastaan lisätään tai vähennetään yllämainittujen apumetodien avulla saatu määrä loppusuuntaan.

Simulator:

Ei sisällä matemaattisia apumetodeja, poislukien hieman lyhyemmälle matkalle modifioidut versiot Light-luokan loppukoordinaattien laskemiselle.

5. Tietorakenteet

Simulator käyttää useampaa muuttuvatilaista Map-rakennetta joihin tallennetaan objektien, valonlähteiden ja valojen sijaintitiedot. Sen lisäksi tekstisekvenssiä ja värisekvenssiä joista status-teksti ja piirtävä elementti lukee tiedot.

Light käyttää muuttuvatilaista HashMap-rakennetta jolla se tiedostaa simulaatiossa olevien objektien sijainnit ja osaa kimmota niistä oikein. Sen lisäksi bresenham-metodilla luodaan Iterator valonsäteen reitistä, joka siirretään advance metodissa Set-rakenteeseen.

Harkitsin pitkään myös Map[Reflective -> Set[(Int, Int)]] -rakenteen käyttöä, mutta mielestäni toiminnallisuus on hyvä tällä tavalla ja en lähtenyt rikkomaan ehjää ratkaisua.

6. Tiedostot

Ohjelmalla on mahdollista tallentaa ruudulla näkyvä skenaario “Start”-valikon kautta “Save state” -valinnalla ja ladata se sitten vastaavasti “Load state”:lla. Ohjelma kirjoittaa objektien ja valojen sijainnit ja suunnat tekstinä tiedostoon sav.sv, joka tulee ohjelman mukana. Tiedosto on mahdollista kirjoittaa käsin, mutta ohjelmaa ei ole alustettu virheellisen tiedon käsittelyyn, joten varminta on käyttää yllämainittua sisäistä tallennusmetodia. Ohjelma kuitenkin ladattaessa tarkistaa, löytyykö tallennustiedostoa. Jos ei, ei latausta suoriteta.

7. Testaus

Aivan kuin olin suunnitelmassanikin ajatellut, lähdin rakentamaan ohjelmaa minulle vaikeimman, eli graafisen käyttöliittymän kautta. Se osoittautui tämäntyyppisessä ohjelmassa toimivaksi ratkaisuksi, sillä suoraan graafisen käyttöliittymän kautta ohjelman testaaminen kokonaisuudessaan kertoi valtavan paljon ja ongelmia oli helppo paikantaa vaikkapa lisäämällä koodin väleihin println-komentoja. Muun muassa linssien toiminta eri puolilta hahmottui tällä tavoin hyvin.

Myös osaa metodeista kokeilin REPL:issä erikseen. Esimerkiksi Reflective-luokkien newAngle-kaavoja pähkäilin paperilla ja REPL:in kanssa.

8. Ohjelman tunnetut puutteet ja viat

Testauksen alkuvaiheissa sain ohjelman kaatumaan suurella määrällä satunnaisia objekteja, mutta en ole onnistunut tuottamaan ongelmaa uudelleen. Ohjelman ensisijainen tarkoitus ei varsinaisesti muutenkaan ole mielestäni lisätä näytölle sataa satunnaista objektia ja kymmentä satunnaista valonlähdettä, niin en lähtenyt murehtimaan asiaa

enempää. Lisäsin valolle myös ns. kimpoamisrajan, joka estää valon liikkeen sadan kimpoamisen jälkeen. Tämä siksi, että on mahdollista saada valo jumiin jo vaikka kahden samansuuntaisen peilin välille.

Ohjelman käyttö on myös pidemmän päälle jokseenkin kankeaa, ja erilaisten objektien asettamista ja kulmien säätöä voisi yksinkertaistaa vaikkapa painikkeilla ja sliderilla. Objektien pituuden muokkauksen voisi myös lisätä ominaisuutena. Ajattelin, että tässä muodossa se kankeuttaisi entisestään käyttöliittymää.

Tein ennen tallennus- ja latausmahdollisuutta valmiiksi määritellyn preset scenarion käsin kirjoittaen koodiksi pitkähköksi preset-metodiksi. Tämä olisi toki tallennuksella mahdollista yksinkertaistaa ja ulkoistaa .sv tiedostolle jos laajentaisi latausmetodia ottamaan eri parametreja ja tiedostoja. Toisaalta en halua liikaa omituisia tiedostoja roikkumaan mukana niin en lähtenyt muuttamaan tätä.

9. Parhaimmat ja heikoimmat kohdat

Itselleni hieman heikolta tuntuu Simulatorin MainFrame, jonka sisään on suoraan rakennettu suuri osa toiminnallisuudesta, jonka vuoksi myös osa metodeista on pitänyt tunkea sinne. Näin internetiä selaillessani yksinkertaisemman swing-applikaation joka oli toteutettu tähän tapaan ja tartuin toteutustapaan itsekin. Varsinaisesti se ei kuitenkaan rajoita ohjelman toimintaa tai jatkokehitystä.

Toinen heikkous on Light-luokan objects HashMapin tarve paksuntaa objekteja lisäämällä myös x+1 -koordinaattiin kyseinen objekti. Tämä kuitenkin huomattavasti vaikeuttaa valonsäteen karkaamista objektien läpi, eritoten ei-vaakasuuntaisten.

Joitakin metodeja voisi muokata niin, että ne ottavat parametrin. Esimerkiksi getEnd-metodi on kopioitu Simulator -luokasta Lightiin, vaikka periaatteessa metodin voisi toteuttaa myös lisäämällä parametriksi haluttu pituus. Toinen esimerkki on automatisoitujen objektien lisääminen, esimerkiksi tallennustiedostosta. Metodi placeObj voisi periaatteessa ottaa parametriksi kulman, eikä käyttää aina variable anglea. Tämä hankaloittaa hieman muun muassa tiedostosta lataamista, kun kulma pitää aina määrittää variableen.

Joskin hieman sekava, olen tyytyväinen rekursiiviseen advance-metodiini, joka piirtää valonsäteen. Kokonaisuutena olen myös tyytyväinen koko projektin toimintaan.

10.Poikkeamat suunnitelmasta

Suunnitelmassani oli eroavaisuuksia lopputulokseen. Ajattelin, että koverien ja kuperien linssien eroavat taittokulmat sijaitsisivat Array-rakenteessa. Loppujen lopuksi valo laskee

törmätessään taittokulman muutoksen sen perusteella, kuinka kaukana törmäyspiste on objektin alkupisteestä. En avannut suunnitelmassani laajasti algoritmeja. Pidin kuitenkin ajatukseni että kulma on vain suunta, kuitenkin niin, että 0 on suoraan oikealle ja siitä kiertäen myötäpäivään. Ajatukseni valonlähteen toiminnasta oli aikalailla samalla suunnalla kuin lopputuloskin.

Ajankäyttösuunnitelmani ei pitänyt paikkaansa. Toteutusjärjestyksestä pidin kuitenkin kiinni, ja siitä, että en ole ihan viimeisenä päivänä liikkeellä.

11.**Toteutunut työjärjestys ja aikataulu**

Itselleni vaikein alue on toteuttaa graafinen käyttöliittymä. Tehtiin kolmen hengen ryhmässä projekti aiemmalle kurssille, mutta toteutin siihen enemmän backendiä, jonka vuoksi odotinkin innolla haasteena käyttöjärjestelmän toteutusta. Tämän vuoksi, niinkuin olin suunnitellutkin, lähdin käyttöliittymän kautta toteuttamaan koko ohjelmaa. Muut käynnissä olevat kurssit veivät kuitenkin huomioni kunnes oikeastaan vasta huhtikuun alussa rupesin vakavasti paneutumaan ohjelmaani. Git-commiteistani ei ole juurikaan apua antamaan tarkempia päivämääriä, sillä päädyin usein toimimaan vanhanaikaisesti kopioimalla vanhaa koodiani tietokoneen tekstieditoriin, versionhallinnan sijaan.

Toteutusjärjestys: käyttöliittymän piirtyminen ruudulle, erinäköisten objektien lisääminen käyttöliittymän ikkunaan, valon siirtymisen metodit, heijastavien pintojen sisäinen toiminnallisuus (uuden kulman laskeminen), tallennusmahdollisuus ja muut lisäominaisuudet käyttöliittymään.

12.Arvio lopputuloksesta

Kaiken kaikkiaan olen tyytyväinen. Ohjelma toimii niinkuin tehtävänannossa vaaditaan. Käyttöliittymää voisi vielä ajan kanssa pohtia ja hioa, mutta ensimmäiseksi versioksi ohjelmani on kyllä toiminnaltaan mielestäni oikein hyvä. Käyttöliittymään voisi kenties lisätä esimerkiksi oikeaan reunaan nappulat joilla valitaan objekti, slideri jolla valitaan kulma. Mahdollisesti myös yksittäisen objektin poisto, hiiren perässä raahautuva “haamu”-objekti ennen varsinaista asettamista. Aloitin projektin alusta moneen kertaan vuorotellen swingillä, scalafx:llä ja processingilla, mutta päädyin kuitenkin itselle näistä tutuimpaan swingiin. Eräässä vaiheessa myös käytin hiirikäyttöisempää ohjausta, esimerkiksi kulma valittiin hiiren rullalla. Tämä ei tuntunut kuitenkaan niin luontevalta, kun tapaan käyttää hiirtä harvoin.

13.Viitteet

Suurimpana apuna itselleni toimi juurikin suunnitelmassani mainitsemani http://www.stackoverflow.com ja http://math.stackexchange.com. Näistä löytyy runsaasti keskustelua googlella. Linkkejä:

https://en.wikipedia.org/wiki/Bresenham%27s_line_algorithm

http://stackoverflow.com/questions/7082801/bresenhams-line-algorithm-error

http://stackoverflow.com/questions/15544549/how-does-paintcomponent-work

http://stackoverflow.com/questions/11907947/how-to-check-if-a-point-lies-on-a-line-between-2-other-points

http://stackoverflow.com/questions/3136457/responding-to-key-events-in-scala

http://www.scala-lang.org (mm. Swing.event, Dialog, Math, HashMap, Iterator)

Näiden lisäksi suurena apuna toiminut wolframalpha, perinteinen funktiolaskin sekä paperi ja kynä.

14.Liitteet

Liitteenä eclipse-projekti. Huom! Mukana myös sav.sv -tallennustiedosto.