Parvisimulaatio
Tekijä: Juuso Korhonen

Henkilötiedot:

Juuso Korhonen, opiskelijanumero: 62377, Informaatioteknologia, 1.vuosikurssi, 28.4.2019

Yleiskuvaus:

Parvisimulaatio-projekti on kaksiulotteinen simulaatio kappaleiden (esim. lintujen) liikkeestä, jossa pyritään simuloimaan kappaleiden parvikäyttäytymistä (kappaleiden parveutumista, liikettä samaan suuntaan ja samalla nopeudella). Käyttäjä voi vaikuttaa kappaleiden käyttäytymiseen erinäisillä liukusäätimillä ohjelmaikkunan oikeasta laidasta, jotka muuttavat lintujen reagointia ja liikettä. Esimerkiksi nopeussäädin ”maxspeed” luonnollisesti säätää maksimi nopeutta ja ”avoidance”-säädin säätelee millä voimakkuudella kappaleet hylkivät toisiaan niiden asettuessa toistensa turva-alueille (jotka saadaan näkyviin muiden graafisten apuvälineiden tapaan rastimalla ruutu ”draw safespace”).

HUOM! Tulen viittaamaan kappaleisiin lintuina tästä eteenpäin vaikka lopullisessa toteutuksessa kappaleet ovatkin enemmän palloja. Ehkä niiden liikettä on helpompi hahmottaa niihin viitatessa lintuina, näin minä ainakin alusta lähtien ajattelin.

Ohjelman rakenne:

Ohjelmarakenne jakautuu ylhäältä kahteen erilliseen scala-pakettiin, joiden voi ajatella muodostavan omat alikokonaisuutensa: Ohjelmalogiikka (simulaatio) ja graafinen toteutus (gui).

Ohjelmalogiikka (simulaatio-package) käsittää seuraavat luokat:

- Bird
    - Kuvaa yksittäistä lintuoliota, jolle löytyvät sekä paikka- että nopeusvektori
    - Omaa metodit move(), updateVel() ja draw(), jotka ovat vastuussa 1. liikkumisesta 2. nopeusvektorin päivityksestä 3. lintuolion itsensä piirtämisestä
    - Bird-objekti sisältää muutaman Bird-luokalle keskeisen metodin, kuten random lähtönopeuden laskemisen ja nopeuden rajoituksen maksimi- ja miniminopeuden välille (jotka myös objektin muuttujia)

- Birdbox
    - Kuvaa pelialueen ja omaa listan sen sisältämistä linnuista
    - Metodi moveBirds() liikuttaa pelialueen lintuja
- Vector
    - Vektori-luokka kuvaa yksinkertaisuudessaan vektoria, jolla jokaisella on omat i ja j-muuttujat, ja niiden avulla laskettu pituus.
    - Vektori-objekti taas pitää sisällään vektorien laskusäännöt, ja näihin metodeihin objekti Movement pohjautuu
- Movement
    - Movement-objektin metodit määrittävät lintuolioiden liikkumissäännöt, hyödyntäen vektoreiden yhteenlaskua ja Settings-objektiluokan muuttujia (näiden toteutuksesta enemmän algoritmit osiossa). Settings-objektiluokka löytyy samasta tiedostosta Movement-objektiluokan kanssa.

Graafinen toteutus (gui-package) käsittää seuraavat luokat:

- Flocksimulation
    - Flocksimulation on vastuussa graafisen käyttöliittymän komponenttien (kuten Setting luokasta luotavien säätimien) luonnista ja asettelusta, sekä ohjelman käynnistyksestä (mukaan lukien logiikasta, jolla lintuja alkaa ilmestyä pelialueelle)
- Setting (huom. Eri asia kuin Settings-objekti)
    - Setting-luokka kuvaa säätimien toimintalogiikan, jonka avulla käyttäjä pystyy muuttamaan Settings-luokan parametreja käyttöliittymän säätimiä muuttamalla

Algoritmit

Paras tapa kuvata algoritmien käyttöäni on varmaankin kuvata kuinka kierroksen vaihtuminen vaikuttaa linnun nopeusvektoriin, joka sitten liikuttaa lintua, kun se lisätään sen paikkavektoriin (jollain skalaarilla kerrottuna).

1. Flocksimulation-käyttöliittymäluokan (Fs) tapahtumakuuntelija listener reagoi timerin käyntiin seuraavasti:
    a. Jos kyseessä on pelin alku, se alkaa luomaan Fs:n birdbox muuttujalle Bird-olioita 50 ms:n välein kunnes maksimimäärä lintuja on saavutettu
    b. Lisäksi se siirtää lintuja kutsumalla birdboxin moveBirds() metodia ja maalaa sitten repaintilla paneelin, joka on vastuussa pelialueen näyttämisestä uudelleen ja Bird-luokan draw()-luokan kutsumisesta, joka piirtää linnun itsensä.
2. Birdbox- luokan moveBirds()-metodi käy kyseisen boxin linnut läpi ja kutsuu niille Movement-objektiluokan metodia update(), jolle välitetään vuorossa oleva lintu ja boxi parametreina.
3. Movement-objektiluokka on missä nopeusvektorin päivittäminen tapahtuu:
    a. Update() kutsuu Movementin muita metodeja:

i. keepUp() palauttaa lähistöllä (Settings.visibility) olevien lintujen keskimääräisen nopeusvektorin. Tässä kuten muissakin Movementin metodeissa käytetään Vector-objektiluokan vektorien laskusääntöjä. Tässä tapauksessa Settings.visibility -mittaisen etäisyyden päässä olevien lintujen nopeusvektorit lisätään yhteen Vector.add():lla ja sitten kerrotaan Settings.closeByBirdInfluence:n palauttamalla arvolla ja näiden lintujen lukumäärän käänteisluvulla Vector.multp() käyttämällä
ii. gravitateToFlockCenter() on nimenomaan vastuussa hakeutumisessa parven kanssa yhteen ja palauttaa alueen massakeskipisteeseen osoittavan vektorin. Massakeskipisteen laskenta tapahtuu centerOfMass()-metodia kutsumalla (laskee alueen lintujen paikkavektorit yhteen ja jakaa lintujen määrällä) ja tämä ja vuorossa olevan linnun paikkavektori miinustetaan toisistaan ja näin saadaan massakeskipisteeseen osoittava vektori, joka vielä kerrotaan vaikutusta lieventävällä skalaarilla
iii. avoidCollision() summaa kaikki vuorossa olevan linnun ja sen turvasektorille mahdollisesti eksyneen linnun väliset, vuorossa olevaan lintuun osoittavat vektorit, jotka kerrotaan funktion f(r)=1/(4*e^r)+Settings.avoidance antamalla arvolla. Palautettava vektori jaetaan vielä lopuksi turvasektorin sisältämällä lintujen lkm:llä. EDIT: huomasin, että rajaamalla vielä turvasektoria niin, että se ei ikään kuin näkisi taakseen (matemaattisesti lausuttuna tunkeutujan ja linnun välille muodostuva vektori ja linnun nopeusvektorin välinen kulma on yli 3*pi/4) simulaatio muuttui realistisemmaksi.
iv. gravitateToMiddle() pitää huolen parven pysyttelemisestä pelialueella. Toimii samaan tapaan kuin keskipisteeseen hakeutuminen, mutta nyt massakeskipisteen sijaan onkin pelialueen keskusta, johon kuitenkin lisäsin Random()-luokan avulla pientä vaihtelevuutta (vaihtamalla keskipisteen paikkaa vuorojen vaihtuessa vähäsen) , jotta lintujen liikeeseen tulisi luonnollisuutta.

b. Movement-luokan Update() kutsuu vielä lopuksi vuorossa olevan Bird-olion updateVel()-metodia, jolle annetaan parametriksi vaikutusvektori, joka on yhteenlaskettuna edellisten metodien palauttamat vektorit, ja joka sitten lisätään linnun nopeusvektoriin. UpdateVel() rajoittaa myös linnun nopeusvektorin Bird-objektiluokasta löytyvien max- ja minSpeedien välille.

Tietorakenteet

En tarvinnut muita kuin Scalan omia tietorakenteita, ja niistäkin oikeastaan vain Arrayta ja perusmuuttujia.

Tiedosto ja verkossa oleva tieto

Ohjelma ei käsittele muunlaisia kuin scala-tiedostoja.

Testaus

Ohjelman ensimmäisen version tein aika nopeasti, sen kummempaa testailematta. Ja sen kanssa olikin sitten aika tuskallista rueta bugeja etsimään, kun käyttöliittymän kanssa aloin lintujen liikettä tarkkailemaan. Debuggauksessa käytin melko perinteisiä konsteja (errormessagejen lukemista ja rivipaikannusta, sekä rajaamalla ongelma-alueita välitulostuksin), ja useasta kohdasta sainkin selville missä mentiin vikaan, mutta jostain syystä, ongelmaa, että Bird-luokasta kutsutut, nopeusvektoria päivittävät metodit, eivät sitten loppuunsa päivittäneetkään kyseisen linnun nopeusvektoria. Ei mitään hajua miten se oli mahdollista, koitin ratkaista myös ulkoisen avun kanssa, mutta tuloksetta.

Päädyin rakentamaan ohjelmasta toisen version, hyväksi käyttäen vanhan version logiikkaa, mutta nyt selkeämmällä luokkarakenteella. Enää ei yksi metodi kutsunut montaa metodia eri luokista. Lisäksi kun nyt oli käyttöliittymän pohja jo valmiina, oli helppo testata logiikan toimivuutta lisäilemällä sääntöjä lintujen liikkeelle yksitellen. Tämä lähestymistapa, eli alkeellisen käyttöliittymän rakennus ensin, olisi ollut kannattavaa alusta lähtien simulaation ollessa kyseessä.

Ohjelman tunnetut puutteet ja viat

Tehtävänannon keskeisimpien vaatimusten osalta en löydä merkittäviä puutteita ohjelmasta:

1. Linnut välttävät törmäilyä toisiinsa.
2. Kohtien 2. Linnut lentävät keskimäärin samaan suuntaan ja samalla nopeudella ja 3. Lintu pyrkii parven keskelle toteutumista tarkastellessa käyttöliittymän avulla pidän myös niitä onnistuneina.

Puutteita jatkokehitys mielessä toki löytyy. Olisin halunnut lintujen liikkeistä realistisempaa ja sulavampaa, lisätä erilaisia kuvavaihtoehtoja kappaleille, ylimääräisiä tavotteita liikkeelle… Näissä kaikissa tuli deadline vastaan, josta käsi pystyyn virheen merkiksi aikataulutuksessa. Päätin, että keskityn tehtävänannon pääkohtien täyttämiseen ja ohjelman muuhun hiomiseen ylimääräisten ominaisuuksien sijasta. Toki toteutin säätimet, joiden kautta voi säätää lintujen liikettä, mutta se nyt tapahtui ihan väistämättä hioessa lintujen liikettä.

3 parasta ja 3 heikointa kohtaa

Parhaat:

1. Luokkarakenteen selkeys (helppo jatkotyöstää)
2. Movement- ja Vector-objektien yhteistoiminta
3. Näppärä käyttöliittymä

Heikoimmat kohdat:

1. On mahdollista, että linnut joinakin kertoina jumiutuvat yhteen jonkin ajan kuluttua. Tämä pitkälti johtuu siitä, etten ehtinyt toteuttaa tapaa jolla linnut välillä kasvattaisivat välimatkaa toisistaan. Näin keskellä olevat eivät ajan myötä joutuisi puristuksiin. Lisäksi aloitus keskeltä on hiukan ongelmallinen, sillä sekin voi syöttää lisää painetta parven keskustaan.
2. Erilaiset skenaariot jäivät toteuttamatta. Mielessä olisi ollut useiden eri parvien tai vaikkapa Angry birdsit jahtaamassa possuikoneja, mutta aika loppui kesken.
3. VisibilitySlider ei jostain syystä asetu alussa tappiin, vaikka sen pitäisi näin olla. Kannattaa asettaa se siis itse (simulaatio toimii paremmin)!

Poikkeamat suunnitelmasta, toteutunut työjärjestys ja aikataulu

3 tunnin arkipäiväinen työmäärä ei kyllä ollut mahdollista, sillä oli niin kiireellinen jakso. Onneksi luokkien toteutus onnistui oletettua nopeammin, mutta debuggaamisessa meni suhteessa oletettua kauemmin. Tältä olisin säästynyt jos olisin simulaation ollessa kyseessä tajunnut lähteä testaamaan ohjelmalogiikkaa yksitellen graafisen palautteen kanssa (kuten sitten toisen version kanssa tein). Vaikea arvioida kokonaistyöaikaa, sillä viimeisen version tein viimeisen viikon aikana, mutta ohjelmalogiikkaa toki sain mallinnettua aiemmasta. Sanoisin, että 10 (pitkää) työpäivää

tässä meni. Suunnittelemani kaksinkertainen työmäärä olisi varmasti ollut tarpeen, mutta jakson kiireellisyyden huomioon ottaen se ei vain ollut mahdollista.

Kokonaisarvio lopputuloksesta

Olen tyytyväinen, että sain näin loppusuoralla rypistettyä keskeisimmät vaatimukset täyttävän simulaation. Mieli olisi kyllä ollut tehdä enemmän (varsinkin vaihtoehtoisia skenaarioita) ja toki hioa toteutusta edelleen, mutta deadline tuli vastaan ja sekin myöhässä.

Päällimmäisenä mieleen tuleva puute ohjelmassa on se, että parven keskimmäisten lintujen puristuminen kasaan on mahdollista riippuen suorituskerrasta (EDIT: sain kylläkin nyt lopulta asetettua säätimet niin, ettei sitä käynyt useammallakaan kerralla!). Ratkaisuksi olisin lähtenyt kehittämään jonkinlaista algoritmia, joka välillä höllentää parvikäyttäytymistä niin, että paine keskellä pääsee purkautumaan.

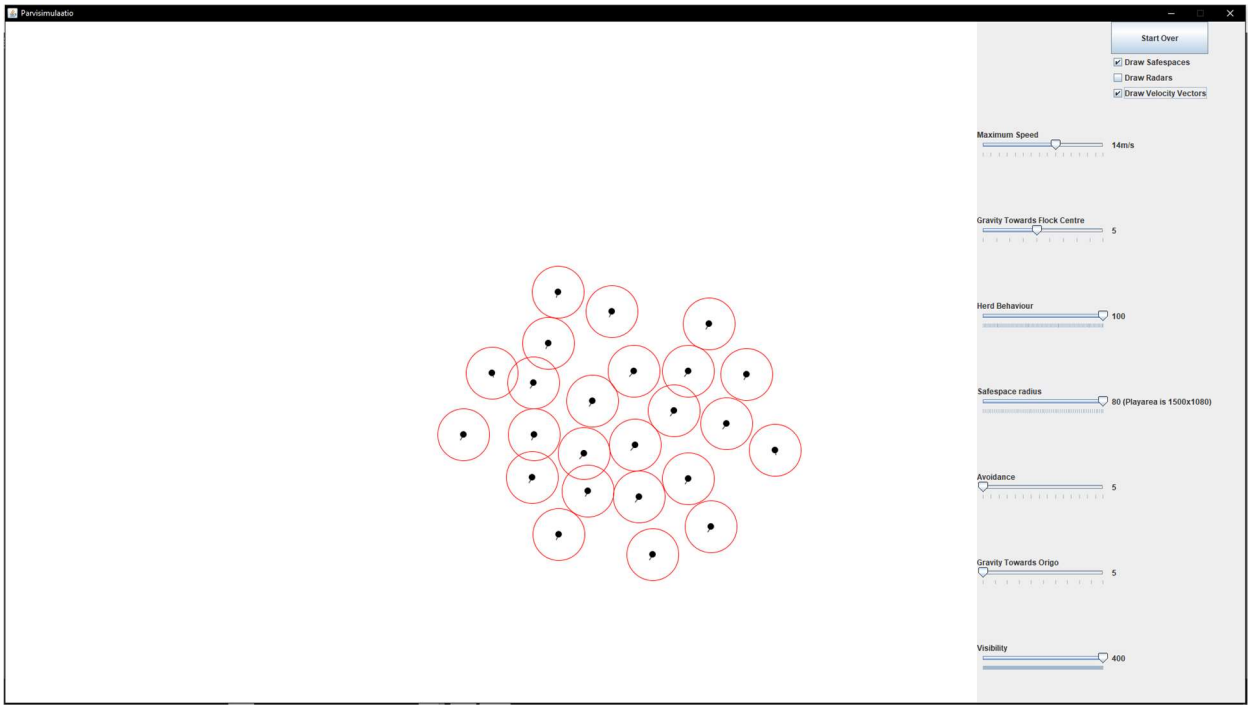
Olen myös ylpeä, kuinka selkeä luokkarakenteen sain aikaan, varsinkin ensimmäisen spagettikoodiversion jälkeen. Uskon, että ohjelma soveltuu mainiosti jatkokehitykseen.

Jotta saisin toimittua paremmin tiukan aikataulun alaisena, jollainen minulla sattui olemaan, tärkein huomaamani kehityksenkohde olisi ollut, että olisin lähtenyt liikkeelle yksinkertaisen gui:n luonnista ja lisännyt sitten siihen logiikkaa yksi kerrallaan. Huomasin toisen version kohdalla, miten ongelmattomasti palaset loksahti yhteen.

Lähteet

Lähteet olivat yksinkertaisuudessaan Craig Reynoldin artikkeli ”Steering behaviors for autonomous characters” http://www.red3d.com/cwr/steer/gdc99/. ja siitäkin aika pääpiirteittäinen. Halusin miettiä vektoriyhteenlaskut yms. Itse fläppitaulun äärellä, mistä ehkä jouduin maksamaan vähän ajassa, mutta toisaalta oli mahtavaa huomata, että itse miettimäni toteutus vastasi sitten aika hyvin Reynoldin logiikkaa artikkelin tarkemmin luettuani.

Liitteet


Parvisimulaatio
Start Over
Draw Safespaces
Draw Radars
Draw Velocity Vectors
Maximum Speed
14m/s
Gravity Towards Flock Centre
5
Herd Behaviour
100
Safespace radius
80 (Playarea is 1500x1080)
Avoidance
5
Gravity Towards Origo
5
Visibility
400