Projektityön dokumentti

Henkilötiedot

Nimi	Tuomas Myllymäki
Opiskelijanumero	652940
Koulutusohjelma	Tekniikan kandidaatti (Tietotekniikka)
Vuosikurssi	2019
Päiväys	23.4.2019

Yleiskuvaus

Projektina oli luoda lintuparvien käyttäytymistä simuloiva ohjelma. Ohjelma käyttää graafista käyttöliittymää, jonka avulla käyttäjä voi hallita sekä tarkkailla simulaatiota. Linnut kuvataan kaksiulotteisesti, joten lintujen liikettä kuvataan niin kuin tarkkailija katsoisi niitä suoraan ylhäältä. Lintuja ohjelmassa kuvataan kolmioilla, joiden kärki edustaa linnun ”päätä”. Lintujen ”siivet” kuvaavat samalla lintujen näkökentän kulman suuruutta. Linnun käyttäytymisen simuloimiseen hyödynnetään kolmea eri peruskäyttäytymistä: *cohesion*, *alignment* ja *separation. Cohesion* kuvaa linnun pyrkimistä olemaan parven keskellä. *Alignment* kuvaa linnun pyrkimistä etenemään samaan suuntaan ja samalla nopeudella läheisten lintujen kanssa. *Separation* kuvaa linnun halua välttää muita lintuja tulemasta liian lähelle. Linnun käyttäytyminen määritetään painottamalla näitä käyttäytymismalleja tietyillä painoilla. Käyttäjä pystyy käyttöliittymän avulla määrittämään painot ennen simulaation käynnistämistä.

Työ on tehty vaikeimmalla vaikeusasteella.

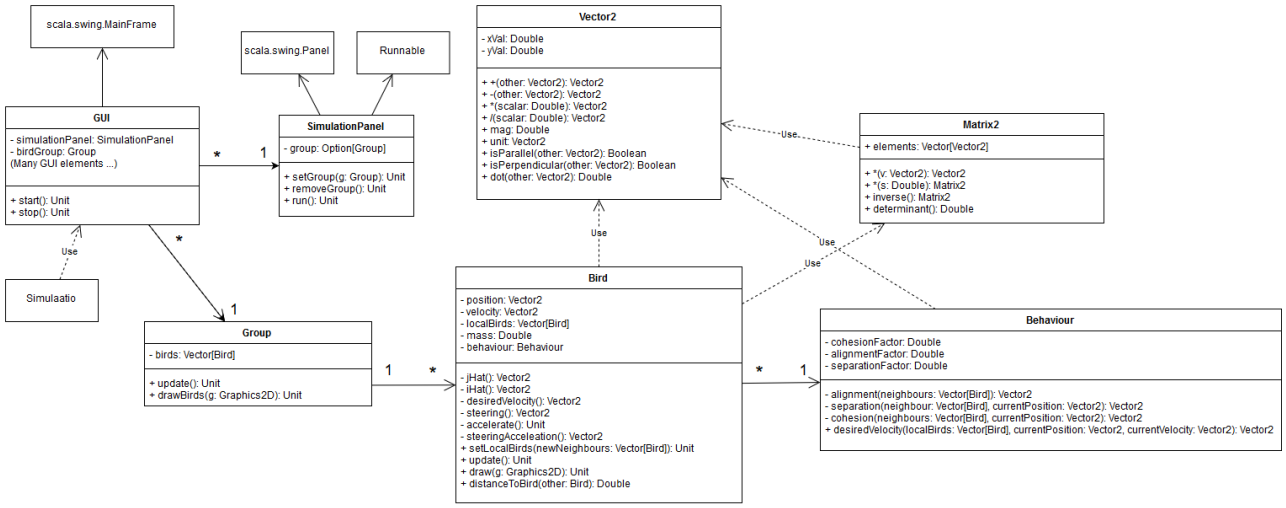
Käyttöohje

Kun ohjelman käynnistää, käyttöliittymä avautuu. Lintujen käyttäytymiseen vaikuttavia parametreja voi muuttaa käyttöliittymän avulla ja myös lintujen lukumäärää ja niiden näkökentän pituutta. Käyttäjä voi myös valita käyttöliittymästä muutamasta esivalitusta mallista. Simulaatio käynnistyy kun käyttäjä painaa nappia ”Start”. Simulaation parametreihin ei voi vaikuttaa sen ollessa käynnissä. Sen voi pysäyttää käyttämällä nappia ”Stop”. Liite (1) näyttää missä nämä napit ovat sinisellä. Samassa liitteessä on punaisella merkitty myös parametrien selite ja niiden tämänhetkinen arvo. Parametreja muutetaan käyttämällä liukusäätimiä, jotka ovat nappien yläpuolella. Alhaalla on valikko josta voi valita tiettyjä skenaarioita. Liite(2) näyttää miltä simulaationäytön pitäisi näkyä kun simulaatio on käynnissä.

Ohjelma käynnistyy suorittamalla GUI.scala tiedostossa olevan *Simulaatio* käynnistysolion.

Ohjelman rakenne

UML-kaavio (tärkeimmät osat):



GUI -luokka huolehtii ohjelman käyttöliittymän hallinnoimisesta. *SimulationPanel* -luokka huolehtii simulaation visualisoinnin piirtymisestä näytölle. *Bird* -luokka laskee yhden linnun liikkeeseen (paikka ja nopeus) liittyvät tiedot hyödyntämällä *Behaviour* -luokkaa. *Bird* -luokka myös pitää kirjaa linnun lähellä olevista linnuista. *Group* -luokka huolehtii kaikkien simulaation lintujen liikkeen ja naapurilintujen päivittämisestä. *Vector2* -luokka on 2-ulotteisen vektorin laskutoimitusten suorittava luokka. *Matrix2* -luokka kuvaa 2x2 matriisia. Se myös laskee matriisien ja vektoreiden tuloja, sekä käänteismatriiseja ja determinantteja. Sitä käytetään lintujen kääntymisen piirtämisen apuna. *Simulaatio* -olio toimii ohjelman käynnistysoliona ja se määrittää muutaman vakion, joita ohjelman muut osat hyödyntävät mm. käyttöliittymän ikkunan dimensiot.

Toinen vaihtoehtoinen luokkajako olisi että *Behaviour* -luokkaa ei olisi vaan se olisi toteutettu osana *Bird* -luokkaa. Tämä tosin hankaloittaisi ohjelman myöhempää kehittämistä ja tekemällä erillinen *Behaviour* -luokka saadaan koodi siistimmäksi. Samaa käyttötarkoitusta ajaa *SimulationPanel* -luokka, joka toimii osana *GUI* -luokkaa.

Algoritmit

Ohjelman käyttämät algoritmit ovat pääasiassa vektorien yhteen ja vähennyslaskua käsittelevät algoritmit. Aloitetaan niiden käsittely kuvaamalla ensin *cohesion*, *alignment* ja *separation* -vektoreiden laskenta.

Cohesion -vektori saadaan seuraavalla tavalla:

```
averageVector = Vector(0, 0)                    // Uusi vektori, jonka molemmat komponentit ovat 0.
for bird in localBirds:                         // Käydään jokainen lähellä oleva lintu läpi.
        averageVector += bird.position          // Lisätään niiden sijainnit averageVector vektoriin.
averageVector = averageVector / localBirds.size // Skaalataan averageVector vektoria lähellä olevien lintujen
                                                lukumäärällä jakamalla.
return averageVector – currentPosition          // Tuloksena saadaan averageVector – currentPosition, eli
                                                vähennetään keskiarvosta  linnun oma sijainti.
```

Separation vektori:

```
totalVector = Vector(0, 0)                                  // Uusi vektori, jonka molemmat komponentit ovat 0.
for bird in localBirds:                                     // Käydään jokainen lähellä oleva lintu läpi.
    deltaPosition = currentPosition – bird.position         // Lasketaan linnun oman ja sen naapurin sijaintien erotus.
    distance = deltaPosition.mag                            // Etäisyys.
    totalVector += deltaPosition.unit * (1.0 / distance)    // Lisätään kokonaisvektoriin sijaintien erotusvektori skaalattuna etäisyyden käänteisarvolla.
return totalVector                                          // Tuloksena saadaan vektori joka osoittaa poispäin kaikista naapuri linnuista.
```

Alignment vektori:

```
averageVector = Vector(0, 0)                // Uusi vektori, jonka molemmat komponentit ovat 0.
for bird in localBirds:                     // Käydään jokainen lähellä oleva lintu läpi.
    averageVector += bird.velocity          // Lisätään niiden nopeudet averageVector vektoriin.
return averageVector / localBirds.size      // Palautetaan keskiarvo kaikista nopeusvektoreista.
```

Nyt meillä on nämä kolme käyttäytymistä kuvaavat vektorit. Otetaan näistä vektoreista painotettu summa:

$$combinationVector = alignmentVector * alignmentFactor + separationVector * separationFactor + cohesionVector * cohesionFactor$$

Nyt meillä on vektori, joka kuvaa linnun haluttua nopeutta. Me saadaan meidän tarvittava muutosvektori vähentämällä nykyinen nopeus halutusta nopeudesta:

$$steeringVector = combinationVector - velocityVector$$

Skaalataan vielä muutosvektoria linnun ”massalla”:

$$accelerationVector = steeringVector/mass$$

Jokaisella päivityssyklillä lisätään nykyiseen nopeuteen skaalattu muutosvektori:

$$velocityVector_{t+1} = velocityVector_t + accelerationVector_t$$

Ja viimeiseksi päivitetään linnun paikkavektori:

$$positionVector_{t+1} = positionVector_t + velocityVector_t$$

Liike ja paikka ovat alussa satunnaisia.

Tietorakenteet

Ohjelma käyttää pääasiallisena listarakenteena scala.Vector -luokkaa. Sitä käytetään sillä se on muuttumaton ja ohjelma on ohjelmoitu sillä tavalla että niiden sisältöjä ei tarvita muuttaa. Esimerkiksi *Bird* luokan localBirds ilmentymämuuttuja on scala.Vector luokan ilmentymä. Kun lähellä olevat linnut päivitetään, rakennetaan uusi vektori ja korvataan vanha sillä. Vector luokkaa on käytetty paljon sillä se on nopea ja helppokäyttöinen tietorakenne. Muita vaihtoehtoja olisi ollut esimerkiksi *Buffer* -luokka mutta se on muuttuva, joten se ei tue yhtä hyvin funktionaalista ohjelmointitapaa. Emme tarvitse muuttuvia listarakenteita.

GUI -luokassa käytetään Map -tietorakennetta valikon ohjelmoimisessa, sillä on luontevaa säilöä ”dropdown” -valikon vaihtoehdot ja niiden tiedot toisiinsa.

Tiedostot & verkossa oleva tieto

Ohjelma ei käytä mitään valmiiksi olevaa tietoa. Ohjelman algoritmit ovat hyvin yksinkertaisia ja nopeita suorittaa, joten ei vaadita etukäteen laskemista.

Testaus

Ohjelma on hyvin visuaalinen, joten suurin testaustapa mitä ohjelman kehittäessä käytettiin oli silmämääräinen arviointi. Laskentaluokkia *Vector2* ja *Matrix2* testattiin yksikkötesteillä. Testit ovat *Vector2UnitTests* ja *Matrix2UnitTests* yksittäisolioissa. Testeihin ei käytetty varsinaista testauskirjastoa, mutta niiden tulokset tulostettiin konsoliin. Ohjelman testaaminen oli hyvin helppoa, sillä pystyi helposti visuaalisesti tarkastelemaan ohjelman kulkua ja esimerkiksi tulostamalla arvoja konsoliin sai tärkeitä lokitietoja ohjelman kulusta.

Tunnetut puutteet ja viat

Ohjelmassa ei suuria ongelmia ole. Aloitustilanteita, joista käyttäjä valitsee voisi olla enemmän ja GUI voisi olla hienomman näköinen. Linnut eivät törmää toisiinsa, mutta mielestäni tämä antaa simulaatiolle paremman ulkonäön. Parametrien säädöt ovat hyvin tekokeinoisia, minimit ja maksimit ovat määrätty vain silmämääräisesti. Oletusarvoinen käyttäytyminen on lähellä luontevaa, mutta ei varmastikaan aivan täydellistä. Linnut myös siirtyvät simulaation toiselta reunalta toiselle kun ne ylittävät ikkunan rajan.

Plussat ja miinukset

+ Ohjelma on mielestäni hyvin toteutettu ja GUI on selkeä ja dynaaminen. Simulaation visualisointi on selkeä ja se toimii sulavasti, säikeiden avulla toteutettuna. On myös paljon parametreja mitä käyttäjä voi muuttaa ja ne ovat selkeästi kerrottu.

- Alkutilanteita voisi olla enemmän ja GUI ei ole mikään kaikista nätein. Alkutilanteita voi tosin lisätä aika helposti ja GUI voidaan saada näyttämään paremmalta pienellä uudelleen suunnittelulla ja Swing-kirjaston paremmalla hyödyntämisellä.

- Lintujen kokoa ei ole mahdollista muuttaa käyttöliittymän avulla.

Poikkeamat suunnitelmasta ja työjärjestys

Mitään suuria luokkamuutoksia ei juuri ole. Linnun käyttäytymistä kuvaava *Behaviour* luokka tuli uutena ja myös matriisilaskentaluokka *Matrix2*.

Aloitin työn (20.2.2019) luomalla karkean version käyttöliittymästä ja sen jälkeen aloin toteuttamaan lintujen liikettä ja piirtymistä koskevat metodit. Toteutin myös tässä vaiheessa *Vector2* ja *Matrix2* luokat liikkeen ja piirtämisen avuksi. Kun ne olivat valmiita (3.3.2019), aloitin toteuttamaan lintujen käyttäytymisen. Tein sen ensin itse *Bird* luokkaan mutta myöhemmin siirsin sen omaksi *Behaviour* luokaksi. Tässä vaiheessa projektin ydin oli tehty (10.3.2019). Tämän jälkeen lisäsin käyttöliittymään elementtejä sekä tein yksikkötestaus oliot. Siitä asti projekti on kehittynyt hitaasti lisäämällä uusia käyttöliittymä elementtejä ja refaktoroimalla koodia. Viimeiset pienet hiomiset on tehty 23.4.2019

Työ sujui aikalailla suunnitellusti.

Kokonaisarvointi

Ohjelma on mielestäni hyvin toteutettu. Simulaation visualisointi on sujuvaa ja simulaatio toimii suunnitellusti. Käyttöliittymä on dynaaminen, mutta se voisi olla visuaalisesti näyttävämpikin. Mielestäni luokkajako toimii erityisen hyvin ja tietorakenteet toimivat hyvin. Osasin hyödyntää hyvin annettua materiaalia ja aihe oli kiinnostava. Käytin työssä paljon funktionaalista ohjelmointityyliä perinteisen olio-ohjelmoinnin rinnalla.

Ohjelmaan voi lisätä esimerkiksi uusia alkutilanteita suhteellisen helposti, tosin uudenlaisten käyttäytymismallien toteuttamisessa pitäisi muokata luokkajakoa merkittävästi. Lintujen kokoa käyttäjä voisi muuttaa jos tehtäisiin ohjelman käyttöliittymään sille jokin hallintoelementti ja tehtäisiin tarvittavat muutokset ohjelman piirtologiikkaan *Bird* -luokassa.

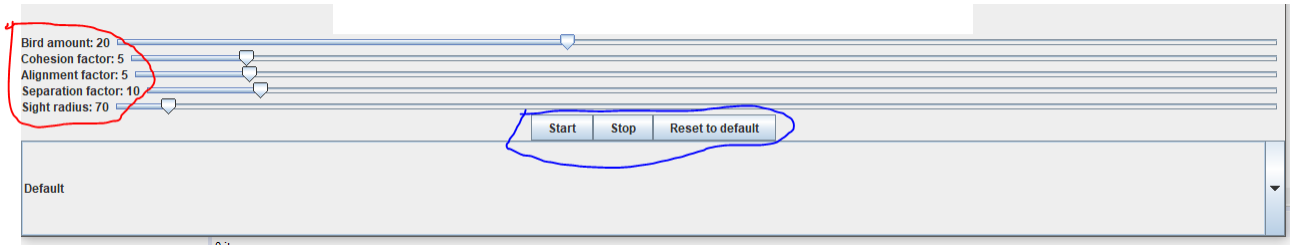
Jos aloittaisin nyt projektin uudelleen, niin laittaisin heti *Bird* luokan käyttäytymisen *Behaviour* luokkaan ja tekisin *Behaviour* luokalle jonkinlaisen yläluokan, joka mahdollistaisi uudenlaisten käyttäytymismallien tekemisen.

Viitteet

- http://www.red3d.com/cwr/boids/ Graig Reynoldsin materiaali aiheesta.
- https://www.math.hmc.edu/calculus/tutorials/changebasis/ Paikallisen ja globaalin avaruuden vektoriesityksien muutos keskenään.
- http://otfried.org/scala/gui.html Opas GUI-ohjelmointiin Scala Swing -kirjastolla.
- https://www.scala-lang.org/api/current/index.html Scala API (versio 2.12.8)

Liitteet

- Liite 1:



- Liite 2: