Parvisimulaatio

Eemi Kärkkäinen 909512

Tietotekniikka 2021

27.4.2022

Yleiskuvaus

Parvisimulaatio on kaksiulotteinen simulaatio, jossa yksilöt noudattavat tiettyjä sääntöjä ja näiden sääntöjen avulla yksilöiden käyttäytyminen saadaan näyttämään oikean lintu- tai esimerkiksi kalaparven liikehdintää. Jokaisella yksilöllä on siis tietyn kokoinen ympäristö (neighborhood), jonka säteeltä yksilöt voivat havaita muita simulaatiossa liikkuvia yksilöitä. Jokainen yksilö siis kerää lähistöllä oleviltaan yksilöiltä tietoja niiden käytöksestä ja mukauttaa oman käytöksensä näiden mukaiseksi.

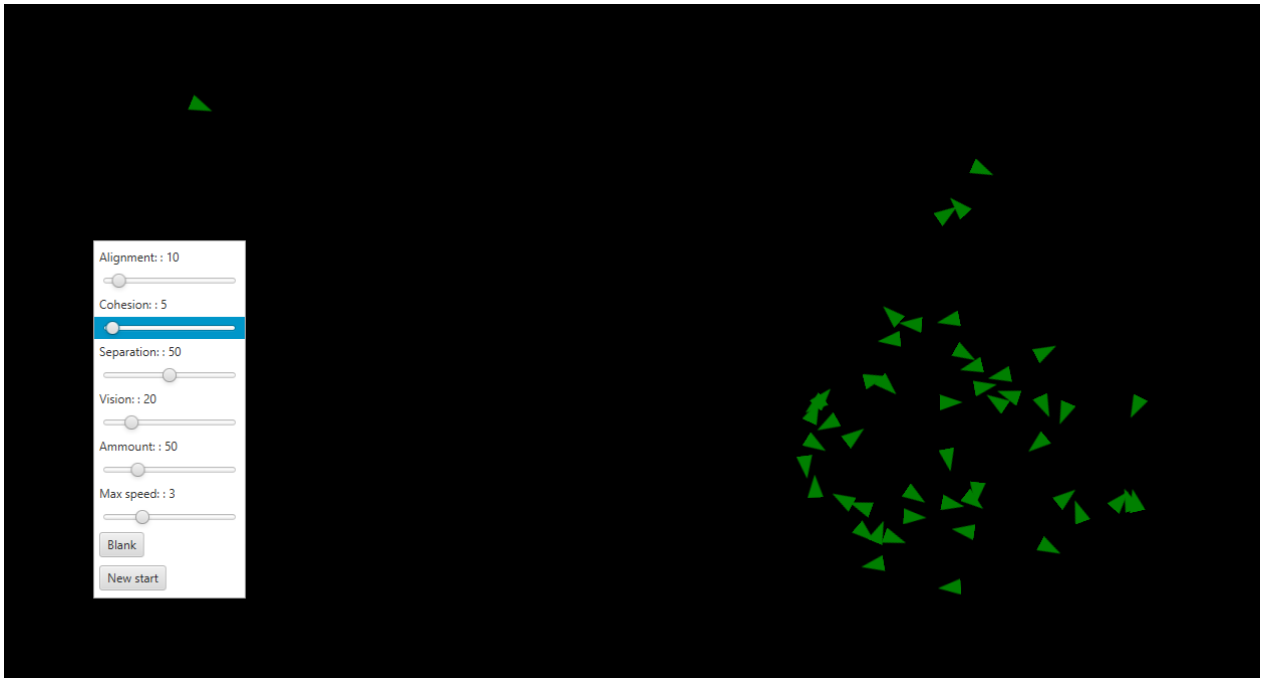
Ensimmäinen sääntö on törmäysten välttäminen (separation). Tämä sääntö saa yksilöt hakeutumaan poispäin muista lähellä olevista yksilöistä, jotta ne voivat välttää törmäilyä toisiinsa. Yksilö siis ohjaa tietyllä painokertoimella itseään poispäin lähellä olevista yksilöistä ja simulaatiossa tätä painokerrointa voi muuttaa tieyllä asteikolla haluamansa mukaan.

Toinen sääntö saa yksilöt ohjautumaan ympäristössä olevien yksilöiden keskimääräiseen suuntaan (alignement). Tämä sääntö saa yksilöt seuraamaan toisiaan, jolloin ne eivät ajaudu liian kauas toisistaan ja käyttäytyminen alkaa jo vaikuttamaan parven käyttäytymistä. Ohjautumisen vahvuutta voidaan myös säädellä erikseen simulaatiossa.

Kolmannen säännön (cohesion) tarkoituksena on pitää parvi kasassa, jolloin se alkaa vaikuttaa ikään kuin pallolta, jonka laidoilla yksilöt käyvät ja ohjautuvat takaisin parven keskustaa. Sääntö siis saa yksilöt ohjautumaan ympäristöllä olevien yksilöiden keskimääräiseen sijaintiin päin (center of gravity). Kyseistä ominaisuutta voidaan myös säädellä simulaation aikana.

Simulaatiossa on vakioarvoina ihanteelliset painokertoimet, joilla parven käyttäytyminen vaikuttaa mahdollisimman luonnolliselta. Näillä painokertoimilla muodostuu parvia, jotka liikkuvat muodossa, joka on pallomainen, mutta jonka muoto ja tilavuus muuttuu jatkuvasti. Simulaatiossa voi myös säädellä yksilön ympäristön sädettä, jolla yksilö huomaa lähettyvillä olevat muut yksilöt. Lisäksi voidaan säätää yksilöiden liikenopeutta, voidaan tyhjentää koko simulaatio ja aloittaa uusi simulaatio käyttäjän haluamalla yksilömäärällä.

Mielestäni projektin toteutus onnistui hyvin ja vaatimusten lisäksi simulaatiossa on ominaisuuksina esimerkiksi ympäristön säteen muuttaminen ja yksilöiden nopeuden muuttaminen, jotka osoittautuvat todella hauskoiksi ja hyödyllisiksi ominaisuuksiksi. Käyttöliittymän toteutuksessa on myös huomioita esimerkiksi kolmioiden suunnan muuttuminen yksilön suunnan muuttuessa ja erillinen valikko, joka tulee esiin hiiren oikealla painikkeella.



Kuvankaappaus käyttöliittymästä ja ohjausmahdollisuuksista

Käyttöohje

Ohjelma käynnistetään IDE:ssä avaamalla GUI-objekti ja painamalla objektin käynnistysnappulaa. Napin painamisen jälkeen simulaatio lataa hetken ja eteen aukeaa musta tyhjä simulaatioalue, jonka otsikkona on “Flocking simulator”. Tämän jälkeen voi painaa joko hiiren vasenta painiketta missä vain mustalla alueella, jolloin simulaatioon ilmestyy uusi yksittäinen yksilö tai painaa hiiren oikeaa painiketta, jolloin aukeaa valikko, jossa on erilaisia säätimiä. “Ammount” säätimellä voidaan valita haluttu yksilömäärä simulaation aloittamiseksi, jonka jälkeen “New start” nappia painamalla haluttu määrä yksilöitä ilmestyy alueelle, ja ne alkavat heti käyttäytymään alkuarvojen mukaisesti. Simulaation edessä voidaan eri painoarvoja muuttaa, jolloin yksilöiden käyttäytyminen muuttuu. Simulaatioon voi myös, milloin vain lisätä hiiren vasemmalla painikkeella uusia yksilöitä. “Blank” painikkeella voi tyhjentää koko simulaatio, ja halutessaan vaihtaa painokertoimia, muita ominaisuuksia ja vaikkapa yksilömäärää ja aloittaa uuden simulaation.

Ohjelman rakenne

Ohjelma koostuu kahdesta objektista ja kahdesta luokasta. Objekteja ovat “FlockingRules” ja “GUI” sekä luokkia ovat “Member” ja “Vector2D”.

“FlockingRules” objekti sisältää kaiken yksilöiden liikkumiseen liittyvän laskennan ja painoarvot sisältävät muuttujat. Jokaiselle painoarvolle on oma metodi, jolla arvoa voi muuttaa helposti ulkopuolelta käsin, jotta käyttöliittymässä olevat säätimet voivat kommunikoida reaaliajassa muuttujien kanssa. Lisäksi on

“neighbors” metodi, jolla voidaan määrittää tietyllä sääteellä yksilön lähellä olevat muut yksilöt *ListBuffer[Member]* kokoelmassa. Objekti sisältää myös “averageSpeed" metodin, joka määrittää “neighbors” metodin avulla yksilön lähistöllä olevien muiden yksilöiden ja kyseessä olevan yksilön keskinopeuden. Samaan tyyliin on toteutettu metodi “centerOfGravity”, joka taas laskee keskimääräisen nopeuden sijasta yksilön ja sen lähistöllä olevien yksilöiden keskimääräisen sijainnin. Muita metodeja ovat “sum”, jolla voidaan laskea kaikkien kokoelmassa olevien “Vector2D” olioiden summan sekä “streering” metodi, joka sisältää kaiken yksilöiden ohjaukseen liittyvät laskennan. Jokaisella simulaation tikillä lasketaan jokaiselle yksilölle uusi nopeus, joka määrittyy kolmen pääsäännön mukaisesti. Lisäksi metodi estää yksilöitä häviämästä alueen ulkopuolelle ja siirtää yksilön vastakkaiselle laidalle sekä lisää paikkaan nopeuden, jolloin yksilöt liikkuvat ja lopuksi myös kiertää käyttöliittymässä olevaa kolmiota yksilön suunnan mukaisesti ja vaihtaa kolmion paikkaa yksilön paikan muuttuessa.

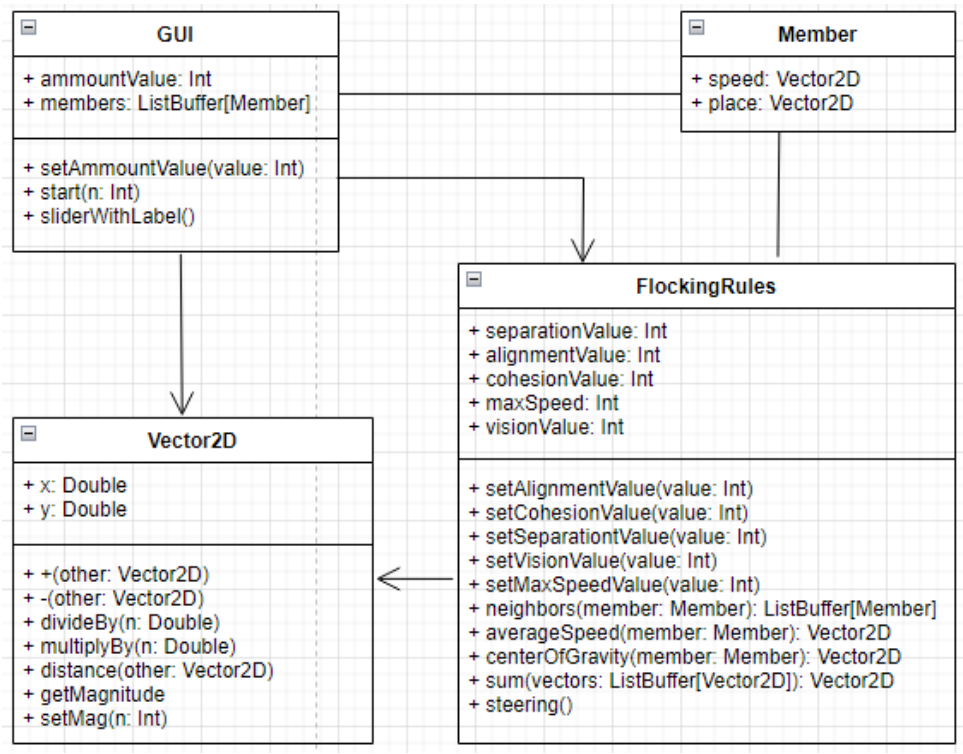
“GUI” objekti on simulaation käyttöliittymä, joka hoitaa simulaation käynnistymisen ja tiedostosta lukemisen. Objektissa määritellään kaikki käyttöliittymässä tarvittavat osat, muodostetaan simulaation satunnainen alkutilanne ja lopussa maalataan valmis käyttöliittymä ja käynnistetään simulaatio. Objekti sisältää myös apumetodin käyttöliittymän säätimien ja niiden otsikoiden muodostamiseen, jotta koodia saadaan hieman selkeämmäksi. Objekti kutsuu animaation ajastimessa “FlockingRules” objektin metodia “steering”, jolloin kaikki tarvittavat muutokset saadaan laskettua ja simulaatio toimii mutkitta.

“Member” luokka ottaa parametreinaan kaksi “Vector2D” oliota, jotka määrittävät yksilön nopeuden ja paikan. Nämä muuttujat ovat tärkeitä simulaation kannalta, koska ne varastoivat jokaisen yksilön henkilökohtaiset nopeus- ja paikkatiedot. “Member” luokassa määritellään jokaisen yksilön muoto ja väri, jotta ne saataisiin lisättyä käyttöliittymään.

“Vector2D” luokka ottaa parametreinaan x- ja y-koordinaatit, joiden avulla voidaan määrittää simulaatiossa esimerkiksi yksilön nopeuden suunta ja yksilön paikka käyttöliittymässä. Luokassa on hyödyllisiä apumetodeja kuten “+”, jolla voidaan summata kahden “Vector2D” olion koordinaatit yhteen, “-”, jolla voidaan taas vähentää oliot toisistaan, “divideBy”, jolla voidaan jakaa olion koordinaatit parametreina valitulla Double-arvolla. Lisäksi on metodi “multiplyBy”, jolla taas voidaan kertoa olion koordinaatit halutulla Double-arvolla sekä “getMagnitude”, jolla voidaan määrittää vektorin pituus koordinaatistossa ja metodi “setMag”, jolla voidaan normalisoida vektori halutun pituiseksi.

Luokkien välisistä riippuvuuksista oleellisimpia ovat “FlockingRules” ja “GUI” välinen riippuvuus sekä näiden riippuvuudet “Vector2D” ja “Member” luokkiin. Koska “FlockingRules” sisältää suurimman osan laskennasta, kutsutaan “GUI”:ssa melko paljon kyseistä objektia ja sen laskennallisia algoritmeja. Lisäksi “FlockingRules” objektissa kutsutaan paljon “Member” luokan olioita ja niiden sisältämiä arvoja simulaation etenemiseen liittyvien arvojen laskennassa ja “Vector2D” luokkaa, koska “Member” luokan olioiden sisältämät arvot ovat kyseisiä “Vector2D” luokan olioita ja niiden käsittelyyn tarvitaan luokalle ominaisia metodeja.

Toki ratkaisuja olisi varmasti ollut useita ja “FlockingRules” objektin olisi esimerkiksi voinut yhdistää “Member” luokkaan, jolloin paketista olisi tullut tiiviimpi. Myös monet yksilöiden ohjautuvuuteen liittyvät rakenteet olisi voinut toteuttaa monilla eri tavoilla, mutta loppujen lopuksi ratkaisujen idea olisi sama.



UML-kaavio luokkarakenteesta

Algoritmit

Keskeisimmät algoritmit sisältävät melko yksinkertaista vektorien välistä laskentaa. Esimerkiksi algoritmit määrittävät ympäristön keskinopeuksia $\frac{\sum_{l=1}^{n}(speed)}{n}$, ympäristön keskimääräistä sijaintia $\frac{\sum_{l=1}^{n}(place)}{n}$ ja esimerkiksi vain vektorien summia. Ehkäpä keskeisin algoritmi “steering” laskee jokaiselle jäsenelle uuden nopeuden joka tikillä. Ja koska nopeusvektori määrittää myös suuntaa, muuttuu näin myös yksilön suunta. Ensimmäiseksi metodissa lasketaan *alignment* eli vähennetään ympäristön keskimääräisestä nopeudesta yksilön nopeus ja lisätään yksilön nopeuteen painoarvon kohdalleen säätämisen jälkeen, niin saadaan tasainen ohjautuvuus ympäristön keskimääräiseen suuntaan.

Toiseksi lasketaan *cohesion* eli yksilön suhteellinen ohjautuvuus ympäristön keskimääräiseen sijaintiin päin. Kyseinen arvo saadaan vähentämällä ympäristön keksimääräisestä sijainnista yksilön nopeus ja jälleen oikean painoarvon löytyessä lisäämällä se yksilön nopeuteen.

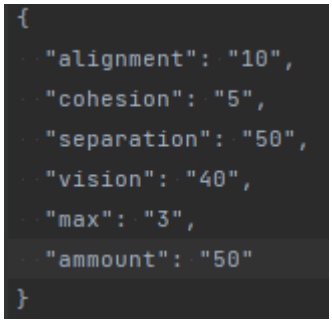
Kolmanneksi lasketaan *separation* eli yksilön ohjautuvuus pois muiden luota. Tämä saadaan laskettua summaamalla kyseessä olevan yksilön paikan ja muiden paikkojen erotetut vektorit, säätämällä painokerroin kohdalleen ja lisäämällä yksilön nopeuteen.

Tietorakenteet

Ohjelmassa käytettävä tieto varastoidaan suurimmaksi osaksi muuttujiin, mutta myös muutettavia kokoelmia tarvitaan esimerkiksi “Member” olioiden varastoimiseen ja käsittelyyn simulaation aikana. Koska simulaatiossa painoarvojen muuttaminen on olennaista, kannattaa nämä tiedot varastoida uudelleen kirjoitettaviin muuttujiin, jotta esimerkiksi käyttöliittymässä olevat säätimet on helppo yhdistää rakenteeseen. Toki löytyy myös yksilöiden muodot ja värit sekä esimerkiksi kaikki käyttöliittymän osat, jotka ovat sijoitettu muuttumattomiin muuttujiin.

Tiedostot ja verkossa oleva tieto

Ohjelma hakee simulaation alkuarvot JSON-tekstitiedostosta, jossa kaikki tarvittavat muuttujien arvot on esitetty selkeästi omilla nimillään alla olevaan tapaan.



Kuitenkaan alkuarvoja ei periaatteessa ole tarpeellista muuttaa, koska niiden muuttaminen ja simulaation alustaminen sen käynnissä olon aikana on todella paljon jouhevampaa, kuin simulaation sammuttaminen, alkuarvojen muuttaminen ja taas simulaation käynnistäminen.

Testaus

Ohjelman testaus onnistui suunnitelman mukaisesti suurimmalta osin juuri käyttöliittymän avulla. Kun käyttöliittymän sai toimimaan, muu toteuttaminen ja sen testaus oli melko suoraviivaista aloittamalla simulaatio ja tarkastelemalla yksilöiden käytöstä. Myös luonnollisten painoarvojen löytäminen oli mahdollisimman luonnollista tähän tapaan. Myös metodien yksikkötestaus oli tarkoituksen mukaista ja suurin osa metodeista eivät muuttuneet niiden luomisen jälkeen, koska ne luotiin spesifiin käyttötarkoitukseen.

Ohjelma läpäisee kaikki testit, koska kaikki testatut metodit ovat oleellisia simulaation toimintaan ja niiden toimimattomuus aiheuttaisi ongelmia joko simulaatiota käynnistäessä tai sen aikana. Testauksen suunnittelussa ei ollut olennaisia aukkoja, koska suunnitelma oli hyvä ja testaus oli melko suoraviivaista. Yksikkötestit toteutettiin yksinkertaisilla testeillä, joissa testattavan metodin tulosta verrattiin haluttuun tulokseen.

Ohjelman tunnetut puutteet ja viat

Yksi ohjelmaa toteuttaessa ilmennyt vika oli käyttöliittymässä. Kun yksilöt ajautuivat tarpeeksi lähelle reunaa tarpeeksi suurella nopeudella, ajautuivat ne jostain syystä alueen ulkopuolelle ja leijuivat poispäin kaukaisuuteen. Tämä korjattiin rajoittamalla yksilöiden maksiminopeus kymmeneen, jolloin ilmiö saatiin minimoitua. Lisäksi yksi parannuskohde olisi ohjelman tehokkuus, koska yksilöiden määrän kasvaessa liian suureksi, toiminta hidastuu todella paljon laskennan lisääntyessä. Nyt ohjelman ollessa valmis, en itse ainakaan löytänyt erikoisia puutteita tai korjattavia vikoja laskentatehon puitteissa. Toki metodeja voisi parannella ja toteutusta selventää, mutta lopullisessa tilassaan ohjelma toimii kuten tarkoitettu.

Kolme parasta ja kolme heikointa kohtaa

Aloitetaan kolmesta parhaasta. Mielestäni käyttöliittymä kokonaisuudessaan on oikein toimiva ja se on melko hyvin eristetty muusta ohjelmasta, jolloin muita osia muokatessa suuria ongelmia ei ilmennyt. Toinen hyvä asia on ehkäpä selkeys. Kaikki metodit ovat melko hyvin jäsennettyjä, jolloin rakenne on melko ilmeinen. Kolmas asia yleinen toimivuus. Olen yllättynyt kun kaikki toimii ainakin omasta mielestäni tarkoituksellisen mukaisesti.

Kolmena heikkona kohtana täytyy sanoa juuri tehokkuus, koska suurien parvien tarkkaileminen voisi olla myös mielenkiintoista, mutta ehkä tarpeetonta projektin skaala huomioon ottaen. Toinen kohta on yksilöiden ajautuminen pois alueelta, mitä saattaa todella harvoin tapahtua, mutta ei kuitenkaan ole huomattava. Viimeinen heikko kohta on ehkä rakenne, jota olisi vielä voinut pohtia. “Member” luokka jäi hieman erilliseksi ja ehkä hieman turhaksi, mutta kokonaisuus toimii ja projekti lähti hieman elämään toteuttamisen aikana, niin en nähnyt uudelleen suunnittelua niin tarpeelliseksi.

Poikkeamat suunnitelmasta, toteutunut työjärjestys ja aikataulu

Projekti poikkesi jokseenkin suunnitelmasta, koska vasta toteutuksen aikana oikeasti ymmärsi, kuinka ohjelma tulee toimimaan ja mitä toteutus tulee kaipaamaan. Ensimmäisten kahden viikon aikana pysyttiin aikataulussa ja suunnitelman kohdat saatiin suunnilleen toteutettua. Seuraavien kahden viikon aikana myös saatiin suunnitelman mukaisesti toteutettua oleelliset osat. Seuraavat kaksi viikkoa taas poikkesivat

suunnitelmasta kiireiden takia, mutta tämä ei oikeastaan vaikuttanut toteutukseen. Viimeisillä viikoilla kuitenkin tapahtui ehkä suurin edistys ja projekti soljui melko jouhevasti loppuun.

Kokonaisarvio lopputuloksesta

Ohjelma toimii tarkoituksen mukaisesti, sen rakenne on melkein suunnitelman mukainen ja se on toteutettu valmiiksi. Ajankäyttöä olisi voinut suunnitella paremmin, mutta projekti kuitenkin eteni mukavasti ja se vietiin melkein ongelmitta loppuun. Ohjelmaan voisi tulevaisuudessa lisätä uusia ominaisuuksia ja käyttöliittymästä voisi tehdä vielä miellyttävämmän ja käyttäjäläheisemmän. Ratkaisumenetelmä on melko yksinkertainen ja selkeä niin sitä en ehkä muuttaisi, sen toteuttamiseen kuitenkin kului melko paljon aikaa ja suunnittelua. Tietorakenteet ovat selkeät ja toimivat, joten niitäkään en ehkä muuttaisi, mutta luokkajakoa olisi voinut vielä projektia toteuttaessa suunnitella ja parannella. Rakenne soveltuu mielestäni muutoksiin hyvin, koska osat ovat eritelty ja lisämetodeja ja esimerkiksi lisälaskentaa simulaation kuluessa on melko helppo toteuttaa.

Jos nyt aloittaisin alusta, tekisin ehkä hieman enemmän taustatutkimusta ja tutustuisin muiden toteutukseen paremmin. Rakenne oli jokseenkin toimiva ensimmäisissä suunnitelmissa, mutta olisi kannattanut ottaa assarin suositukset huomioon. En kuitenkaan siinä vaiheessa ymmärtänyt niiden tarkoitusta ja koin itseopiskelun oikein hyödylliseksi ja kyseiset tiedot ja taidot voivat olla hyviä myös tulevaisuudessa.

Viitteet

http://www.red3d.com/cwr/steer/gdc99/

https://www.researchgate.net/figure/The-Flocking2D-class_fig1_228700404

https://www.jasss.org/13/2/8.html

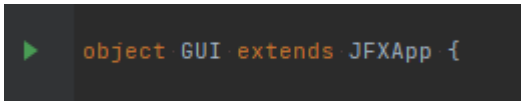
https://eater.net/boids

https://en.wikipedia.org/wiki/Boids

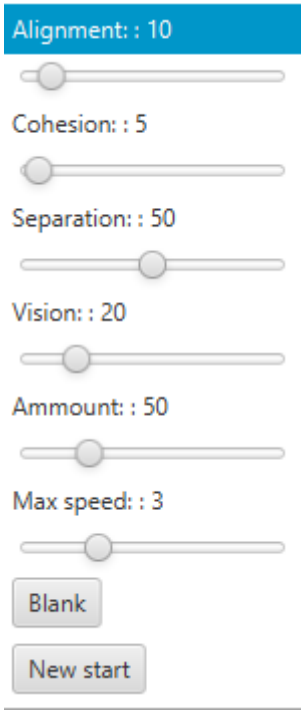
https://github.com/tofti/javafx-boids

https://www.youtube.com/watch?v=mhjuuHl6qHM&t=2230s&ab_channel=TheCodingTrain

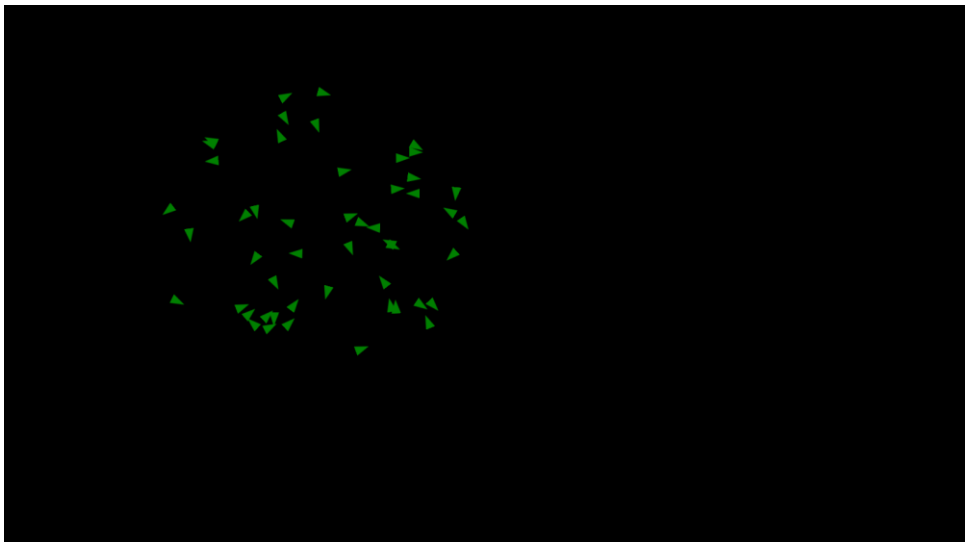
Liitteet



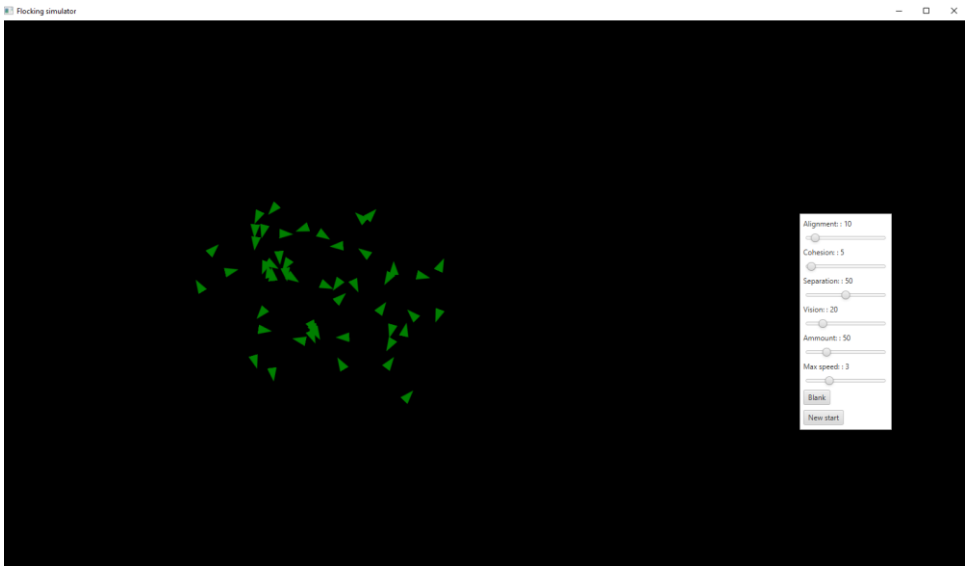
Ohjelman käynnistynappi



Käyttöliittymän säätimet



Kuva käyttöliittymästä toiminnassa



Hiiren oikealla painikkeella ilmestyvät valikko