Projektityön dokumentti

T-106.1215 Ohjelmoinnin peruskurssi, osa 2

Unto Kuuranne

xxxxxx

TIK

1. vuosikurssi

25.04.13

Yleiskuvaus

Projektissa luotiin GUI-ohjelma joka visualisoi parveilevien lintujen liikettä graafisesti Craig W. Reynoldsin klassisen artikkelin ”Steering Behaviors For Autonomous Characters” [1] *A Simple Vehicle Model* -mallin mukaisesti. Mallintamisessa ja visualisoinnissa rajoitutaan 2D-maailmaan kuvaamalla lintujen liikkeitä ylhäältä päin.

Lintuparvia simuloidaan antamalla kullekin linnulle kolme sääntöä

- Separation: vältä törmäilyä muihin lintuihin
- Alignment: lennä samaa nopeutta kuin muu parvi keskimäärin
- Cohesion: pyri kohti parven keskipistettä

Näitä sääntöjä yhdistellään erilaisilla painokertoimilla joita käyttäjä voi säätää käyttöliittymästä sekä ohjelmaa käynnistäessä komentoliittymältä.

Projektityö on toteutettu ohjeistuksen vaativan tason viitoittamana, projektissa keskityttiin ohjeistuksen lisäksi kaiken normaalin toiminnallisuuden toteuttamiseksi käytetyn GUI-kirjastoon tutustumisen nimissä. Esimerkiksi ikkunan koon muuttaminen toimii odotetusti. Myös komentoliittymän käyttöön panostettiin.

Käyttöohje

Ohjelma on kehitetty ja testattu Pythonin versiolla 2.7.

Ohjelma käynnistyy lähdekoodihakemistossa komentoliittymältä käskytettäessä:

```
python2.7 boids.py run
```

Komentoliittymältä annettavissa olevat parametrit saa näkyviin komennolla:

```
python2.7 boids.py –help
```

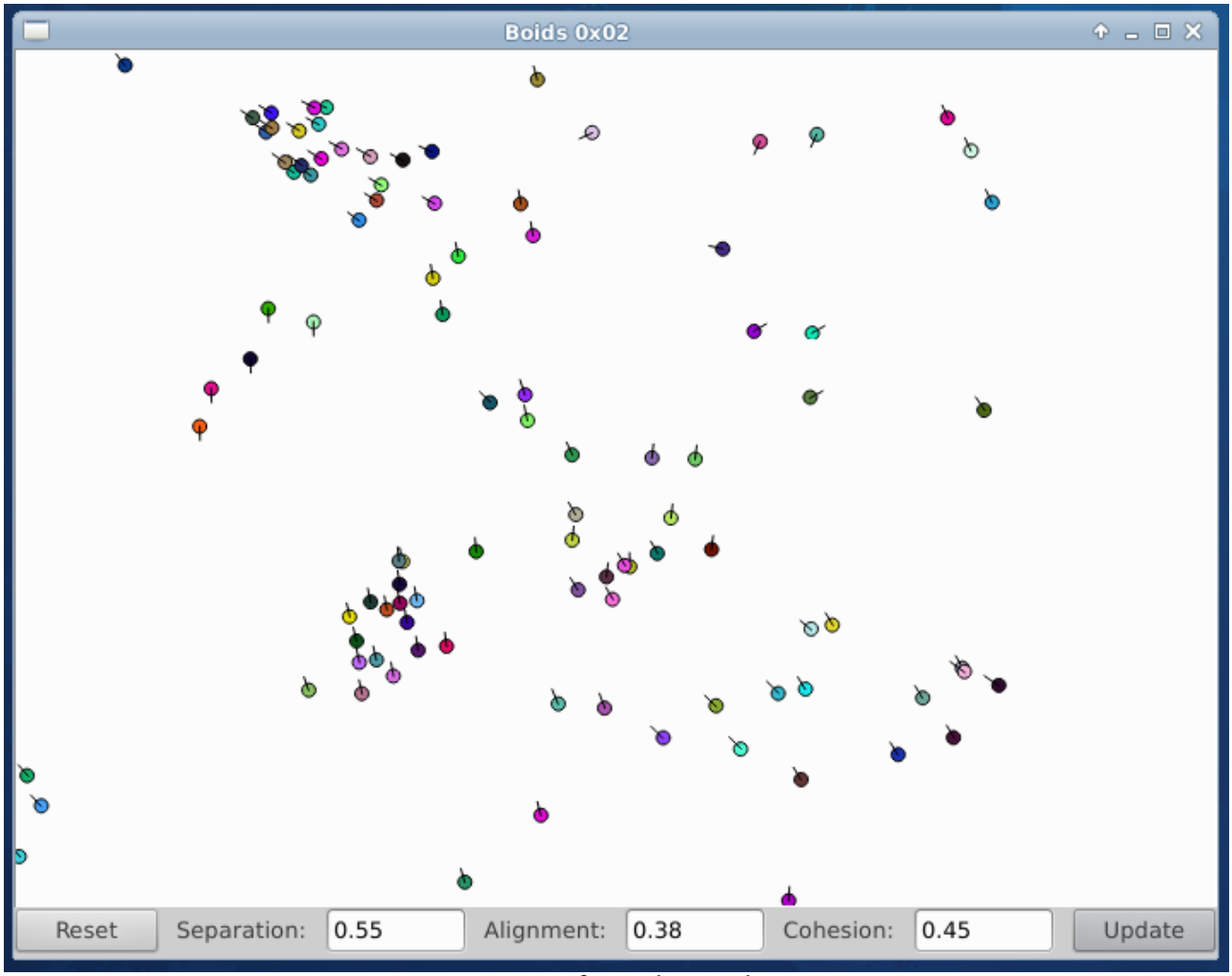
Listaus on annettuna myös seuraavassa:

```
Usage:
  boids.py run [--amount=<int>] [(--width=<int> --height=<int>)]
               [--numviews=<int>]
               [--separation=<float>] [--alignment=<float>] [--cohesion=<float>]
               [--boid-view-angle=<int>]
               [--boid-mass=<float>] [--boid-max-force=<float>]
               [--boid-normal-speed=<float>] [--boid-max-speed=<float>]
  boids.py preset (normal|wonky)
  boids.py --help
  boids.py --version

Options:
  --help                        Show this screen.
  --version                     Show version.
  -a --amount=<int>             Amount of boids
  -w --width=<int>              Window width
  -h --height=<int>             Window height
  -n --numviews=<int>           Number of views (gives n x n grid)
  -s --separation=<float>       Weight for the separation rule
  -l --alignment=<float>        Weight for the alignment rule
  -c --cohesion=<float>         Weight for the cohesion rule
  --boid-view-angle=<int>       Boid's view angle on both sides, 180 is full circle
  --boid-mass=<float>           Mass of the boid
  --boid-max-force=<float>      Force when applying rules
  --boid-normal-speed=<float>   Boids' target normal speed
  --boid-max-speed=<float>      Boids' target max speed
```

Listaus 1: Komentoliittymän käyttöliittymä

Käyttöliittymässä voidaan ajon aikana vaihtaa simulointisääntöjen painoja lennosta asettamalla uuden painon sen kenttään ja painamalla ”Update”. Koko näkymä voidaan alustaa uudelleen luoden uudet linnut satunnaisilla alkuarvoilla painamalla ”Reset”.



Kuva 1: Graafinen käyttöliittymä

Ohjelman rakenne

Ohjelma jakautuu luokkiin jotakuinkin teknisen suunnitelman mukaisesti, pääasialliset erot ovat assistentin toteamuksen johdattelemana laskentaa nopeuttavan spatiaalisen jaon toteuttavan *Grid*-luokan toteuttamatta jättäminen ja *Boid*-luokan kääriminen GUI-päivitykset ja toiminnallisuuden toteuttavaan *GuiBoid*-luokkaan. Pääasiallisen ohjelmalogiikan kannalta teknisen suunnitelman kuvaamat attribuutit ja metodit olivat kattavat. Ymmärrettävästi mm. käyttöliittymä vaatii huomattavan määrän erinäisien asioiden alustamista ja asettelua ja näitä metodeita ei teknisessä suunnitelmassa listattu. Kolmas suurempi eroavuus on ettei ohjelmassa käytetty eksplisiittiesti säikeitä vaan *QTimer*-luokan toteuttamaa ajastinta.

Ohjelma on toteutettu Qt-kirjaston [3] tarjoamien resurssien ympärille. Ohjelman

ydin on *Engine*-luokka joka on *QMainWidget*-luokan ilmentymä. Tämä luokka pitää sisällään käytännössä kaiken ohjelman ja käyttöliittymän alustamisen, käyttöliittymän kanssa viestityksen ja simulaation tilan edistämisen. Ohjelma etenee *QTimer*-ajastimen kutsuessa pääasiallista simulointisilmukkaa.

Enginessä sijaitseva pääasiallinen simulointisilmukka laskee etäisyydet *GuiBoid*-instanssien välillä ja luo *Neighborhood*-objektit näiden mukaisesti. Tämän jälkeen silmukka konsultoi ohjelmaan ladattuja sääntöjä, joiden toteutukset ovat jaettu abstraktin *Rule*-luokan toteuttaviin luokkiin, syöttäen säännöille *Boid*-instanssin ja sen naapuruston. Sääntöjen palauttamat ohjausvoimat normalisoidaan ja painotetaan senhetkisillä painokertoimilla. Painotetut ohjausvoimat yhdistetään ja *Boid*-instanssia käskytetään muuttamaan liikettään niiden mukaisesti. Lopuksi kaikkia *Boid*-instansseja liikutetaan eteenpäin sekä simulaatiossa että käyttöliittymässä.

Näiden pääluokkien lisäksi löytyvät luokat *Vect2d* ja *Orientation*. Näistä *Vect2d* on tuotu PyGame-kirjastosta [2] ja orientation toteuttaa Reynoldsin artikkelin mukaisesti linnun suunnan tallentavan *Vect2d*:ita sisältävän rakenteen.

Lopuksi ohjelmaan tuotiin docopt [8] projektin Python implementaatio komentoliittymän käyttöliittymän toteuttamiseksi.

Keskeiset metodit

Engine loop()

Kuvattu yllä.

Engine init*()

Alustavat käyttöliittymän ja *Boid*-instanssit.

Boid move(force, window_width, window_height)

Päivittää kiihtyvyyden ja uudelleensuuntaa linnun.

Boid step()

Liikuttaa lintua, eli päivittää sijainnin kiihtyvyyden perusteella.

Boid wrap_around(width, height)

Käärii linnun toroidiin, jotta se ilmaantuu kentän toiselta puolelta mennessään kentän rajojen yli.

Neighborhood _calculate_avgerages()

Laskee keskimääräisen sijainnin ja nopeuden naapuristolle.

Vec2d get_dist_sqrd_toroidal(other, wrap_x, wrap_y)

Laskee etäisyyden käärien koordinaatit toroidiin. Toteutettu itse lisäyksenä PyGamen *Vec2d*:hen.

Vec2d toroidal_sub(other, wrap_x, wrap_y)

Laskee kahden vektorin erotuksen käärien toroidiin. Toteutettu itse lisäyksenä PyGamen *Vec2d*:hen

Rule consult(boid, neighbors, width, height)

Eri toteutuksia eri säännöissä. Palauttavat lasketun ohjausvoiman.

Algoritmit

Itse simulointialgoritmi on käytännössä Reynoldsin [1] artikkelin sekä Conrad Parkerin [7] pseudokoodin mukaisesti toteutettu. Projektin luonteesta johtuen kaikki muu laskenta on teknisiä yksityiskohtia.

Naapurustolaskenta on toteutettu naiivisti, se tehdään kahdella sisäkkäisellä silmukalla kaikkien lintujen yli.

Lintujen naapurustot tarkistetaan, kuten Reynoldsin artikkelissa[1], r-säteisen ympyrän mukaisesti eli käytännössä laskemalla lintujen koordinaattien erotukset ja päättelemällä läheisyys epäyhtälöllä $diff\_x^2 + diff\_y^2 \leq r^2$ [7]. Tällä tavoin laskettaes-

sa ei tarvitse jurnuttaa neliöjuurta mikä nopeuttaa laskuja huomattavasti.

Testiversiona Numpyyn [9] tutustuessa/harhautuessa tein Grid-toteutuksen Numpyn matriiseilla. Tyylikäs oli ja laski kahden *Grid*in *Boid*ien väliset etäisyydet kerralla näppärästi. Valitettavasti tämä versio oli hitaampi kuin n^2 objektien yli silmukointi ja siten käytettyjen laskutoimitusten kuvaaminen tässä ei ole minkään arvoista.

Tietorakenteet

Projektissa ei ole toteutettu omia varsinaisia tietorakenteita, vain objekteja jotka olisi voinut kuvata jotakuinkin structeina. Sisäänrakennetuista tyypeistä käytetään pääasiassa listaa sekä muuttuvan että kiinteän kokoisille rakenteille (eli listat vs. taulukot), tämä on yksinkertaisesti siksi että Pythonin lista on paras sen tietorakenne näille molemmille.

Testaus

Projektin luonteesta johtuen testaus oli lähinnä silmämääräistä. *Rule*, *Boid*, ja *Neighborhood* laskennat tarkistettiin simppelisti Python konsolissa kehittäessä. Loput toiminnallisuudesta liittyi lähinnä hyvin yksinkertaisiin CRUD operaatioihin joita niitäkään ei ole oikeasti juuri nimeksikään, niinpä niille kirjoitettu yksikkötestejä.

Projektin aikana testausta tehtiin erilaisilla tarkemmin määritellyillä alkutilanteilla joilla saatiin kontrolloidut olosuhteet ja siten tarkisteltua simuloinnin käyttäytymisen oikeellisuutta.

Myös suunniteltu rämpytystesti tehtiin.

Tunnetut puutteet / viat

Toroidilaskutoimitukset ei ole aivan kohdillaan, joskus jotain väärää käyttäytymistä havaittavissa kentän reunilla.

Parhaat ja heikoimmat kohdat

Hyvää

Toteuttaa oikein kaikki ominaisuudet mitä moiselta GUI-viritykseltä nyt voisi odottaa, mukaan lukien ikkunan koon muuttamisen. Tämän lisäksi toteuttaa kattavat CLI-optiot, sekä grid viewit jos semmosia tykkää katsella.

Kenttä on toroidi.

Huonoa

Testattu vain Linuxilla.

Ikkunan koon muuttaminen toimii vain kun viewit eivät ole gridissä, view gridien tukeminen ei olisi missään määrin jakolaskua vaikeampaa, mutta tämän toteuttamista ei katsottu olennaiseksi.

Kaikkien hienojen Reynoldsin [1] kuvaamien lisäominaisuuksien puute? Tämän olisi voinut korjata implementoimalla ne.

*Engine*n ja *Boid*in staattisiin muuttujiin viitataan toisaalla, aiheuttaen luokkien riippuvuutta toisistaan, koitin välttää tätä ihan suosiolla passailemalla infoa funktiokutsuissa. Tähän voisi käyttää hienoja Design Patterneja ja joutua ojasta allikkoon (paitsi ehkä Dependency Injectionilla... hmm).

Poikkeamat suunnitelmasta

Kuvattu muissa osioissa.

Toteutunut työjärjestys ja aikataulu

Työ toteutettiin käytännössä neljässä sessiossa.

Ensimmäinen 6h sessio sisälsi projektisuunnitelmien teon ja suunnitelmissa mainittujen classien runkojen toteutuksen.

Toinen sessio oli ~12h pituinen pe 15.3. - la 16.3. ilta/yö jolloin toteutettiin GUI ja

säännöt. Checkpointissa työ oli hyvässä kunnossa erästä rajatapauskäyttäytymistä lukuunottamatta.

Kolmas sessio 22.4. oli ~16h harhailu Numpyn ihmeelliseen maailmaan ja *Grid*ien toteuttamiseen Numpy matriiseina.

Neljäs sessio 24-25.4. ~12h jonka aluksi Numpy-versiolla heitettiin vesilintua. Ongelmat poistettiin, angle check lisättiin, koodia siivottiin ja kommentoitiin. Lisäksi implementoitiin erinäistä lisäkivaa kuten ikkunoiden koon muuttaminen ja komentoliittymän käyttäliittymä.

Muutamat pikkufiksit tehtiin vielä tämän jälkeen.

Suunnitelmasta tämä poikkesi siten ettei yksikkötestaus ollut lopulta järkevää ja projekti ei ollut sen verran ajoissa että deadlinea edeltävänä yönä olisi saanut nukkua. Ajassa poikkeama tuli arviosta että Qt:n opettelussa menisi 48h, en lopulta perehtynyt Qt:hen ja sen signaalisysteemeihin syvällisemmin ja siten käytännössä tuo aika jäi vapaaksi.

Arvio lopputuloksesta

Mahtava softa.

Parantaa voisi repimällä suurimman osan koodista irti ja toteuttamalla kaiken matriiseilla mahdollisimman pitkälle. Numpy matriisit voi levittää suhteellisen helposti Pythonin multiprocessing modulin avulla ja siten ohjelman päivitykseen saisi useamman ytimen nykyisen sekventiaalisen laskutavan sijaan.

Parantaa voisi myös kokeilemalla Stackless Pythonia ja toteuttamalla jokainen *Boid* kevyenä säikeenä. Todennäköisesti tämä ei olisi erityisen tehokasta mutta jännää kylläkin ja hyvin mahdollisesti johtaisi erittäin idiomaattiseen koodiin kyseiseen ongelmaan.

Viitteet

[1] Craig W. Reynolds Steering Behaviors For Autonomous Characters <http://www.red3d.com/cwr/steer/gdc99/>

[2] pygame 2DVectorClass
<http://www.pygame.org/wiki/2DVectorClass>
[3] Qt Project Reference documentation
<http://qt-project.org/doc/qt-4.8/>
[4] Qt Project PySide tutorials
<http://qt-project.org/wiki/Category:LanguageBindings::PySide>
[5] PySide 1.0.7 Reference
<http://srinikom.github.io/pyside-docs/>
[6] Python 2.7 documentation
<http://docs.python.org/release/2.7/>
[7] Boids Pseudocode – Conrad Parker (2007)
<http://www.kfish.org/boids/pseudocode.html>
[8] Docopt
<http://docopt.org/>
[9] Numpy
<http://www.numpy.org/>

Liitteet

- Lähdekoodi hakemistossa boids/