Risto Koverola
479699 25.4.2018

Ohjelmointistudio 2: Projektityön dokumentti

Henkilötiedot

Risto Koverola
479699
Tietotekniikan kandidaatti 2015-
5.2.2018

Yleiskuvaus

Tehtävänantoni oli parvisimulaatio. Halusin toteuttaa kaksiulotteisen parvisimulaation, jossa antamalla erilaisia sääntöjä parven yksilöille voidaan vaikuttaa niiden liikkumiseen. Käytin työn perustana Craig Reynoldin artikkelia "Steering behaviors for autonomous characters". Artikkelissa kuvataan erilaisia sääntöjä yksilöiden liikuttamiseen. Yksinkertaisten alkeissääntöjen avulla voidaan toteuttaa kolme perussääntöä: seperation, alignment, ja cohesion. Näitä sääntöjä yhdistelemällä saadaan aikaan parville tunnusomaista liikettä. Myös muita sääntöjä on olemassa, mutta ne eivät kuulu tämän ohjelman minimi toteutukseen. Toteutin myös obstacle avoidance ja target seeking käyttäytymiset.

Lähdin liikkeelle graafisesta näkymästä, jotta yksilöiden liikkumisen saa näkymään ruudulle niin pian kuin mahdollista. Tämän jälkeen mahdollistin yksilöiden liikuttamisen. Sen jälkeen toteutin simple vehicle modelin ja aloin luomaan käyttäytymismalleja. Lopuksi lisäsin graafisen käyttöliittymän, jolla simulaatiota voidaan ohjata. Suoritin tehtävän vaativan tasoisena, sillä lisäsin esimerkiksi ominaisuuden lisätä yksilöitä parveen lennosta, vaihtaa käyttäytymismalleja lennosta sekä toteutin luokat esteille ja kohteille. Obstacle avoidance käyttäytymismalli ei myöskään kuulunut ohjelman minimitoteutukseen.

Käyttöohje

Ohjelma käynnistetään View-tiedostosta ajamalla se Scala applikaationa. Se avaa uuden ikkunan, jossa on uusi tyhjä simulaatio. Ruudun yläosassa on yläpalkki, josta voi valita erilaisia toimintoja ja asetuksia. Simulaatioon voi lisätä komponentteja klikkaamalla ruudulle. Components valikosta voi valita minkälaisia komponentteja simulaatioon lisätään. Boid-komponentit ovat parven yksilöitä, jotka liikkuvat eteenpäin käyttäytymismalliensa (Behaviors) mukaan. Obstacle-komponentit ovat esteitä, joita Boidit väistelevät, mikäli heidän käyttäytymismallinsa niin vaatii. Target-komponentteja voi olla simulaatiossa kerrallaan vain yksi. Boidit joilla on target seeking käyttäytymismalli hakeutuvat tätä kohdetta kohti. Components valikossa on myös nappi, jolla voi poistaa kaikki komponentit simulaatiosta.

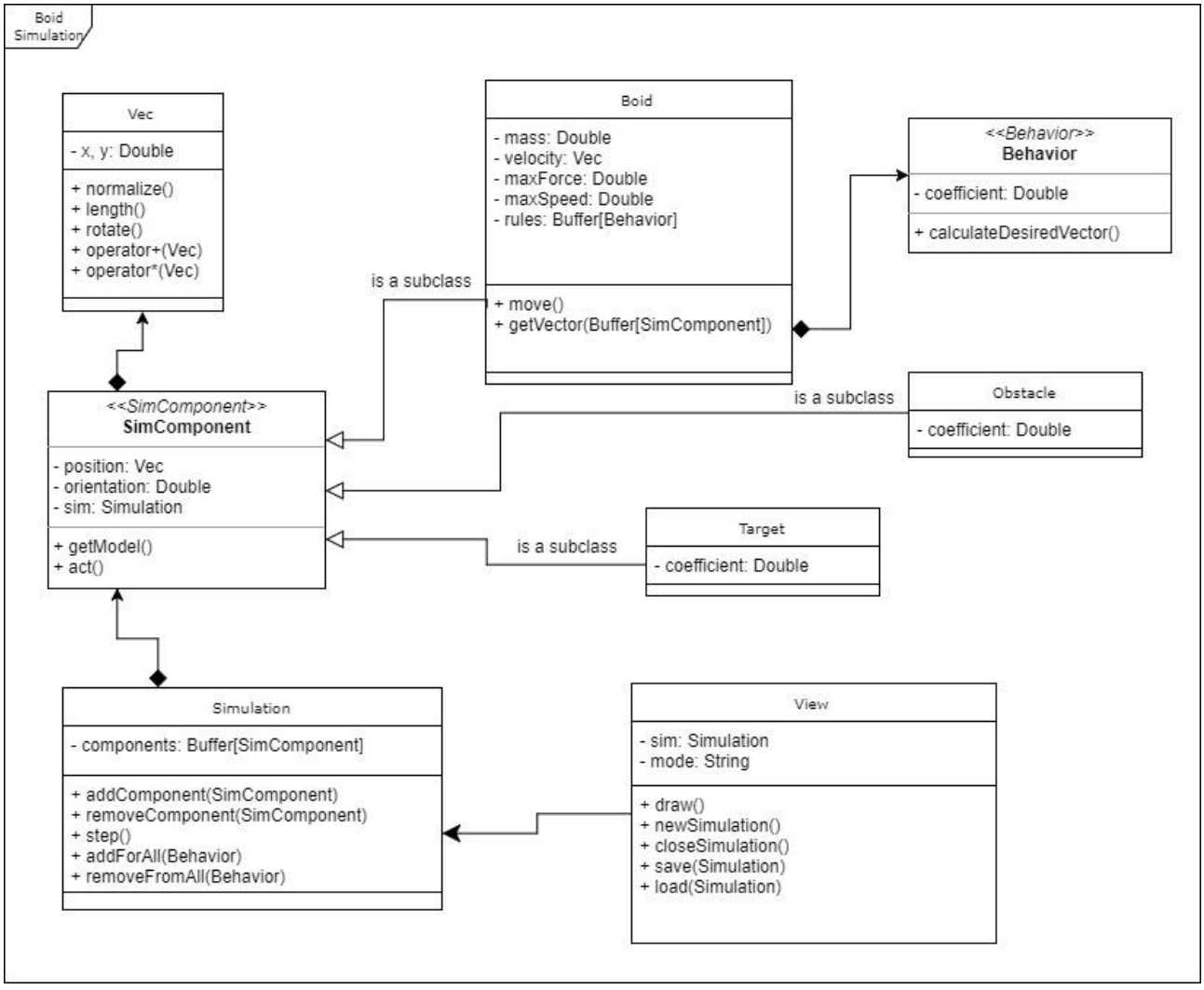
Risto Koverola
479699

Behaviors valikosta voi valita erilaisia käyttäytymismalleja. Valikosta voi valita yhden tai useamman käyttäytymismallin ja kun simulaatioon nyt lisätään uusi yksilö, sillä on nämä käyttäytymismallit ja se noudattaa niitä kaikkia yhtäaikaisesti. Simulaatiossa voi siis olla yksilöitä, joilla on keskenään erilaiset yhdistelmät käyttäytymismalleja ja ne siis liikkuvat eri tavoin. Behaviors-valikossa on myös nappi "Add to all", jolla voidaan laittaa jokaiselle nykyiselle yksilölle käyttäjän valitsemat käyttäytymismallit. Jokaisen yksilön käyttäytymismallit voidaan poistaa painamalla "Add to all", kun mitään malleja ei ole valittu.

Options valikosta voidaan vaihtaa simulaation parametreja. "Loop positions" parametri vaihtaa simulaation rajat niin, että rajan yli menevä komponentti tulee toiselta puoleta takaisin simulaatioon. Oletusarvoisesti simulaation ulkopuolelle menevät komponentit poistetaan. "Draw sectors" parametrilla ruudulle voidaan piirtää Obstacle avoidance käyttäytymiseen käytetyt sektorit, joita yksilöt yrittävät pitää vapaina esteistä. Valikosta voi myös valita voimavektorien piirtämisen näytölle.

Ohjelman rakenne

Kuva 1. Projekti suunnitelmassa esitetty luokkajako (ei ajan tasalla)



Risto Koverola
479699
25.4.2018

Simulation: Simulation on luokka, jonka tärkein osa ovat luettelot simulaation komponenteista. Sen tärkein metodi on act(), joka pyytää jokaista komponenttia toimimaan. Tämän jälkeen kutsutaan move() metodia, joka liikuttaa Boid-komponentteja eteenpäin.

SimComponent: Tämä on abstrakti yliluokka simulaation komponenteille. Tällä luokalla on abstraktit metodit act() ja getModel(), jotka toteutetaan aliluokissa. Kaikilla komponenteilla on lisäksi esimerkiksi paikka ja orientaatio. Abstraktista yliluokasta ei ollut loppujen lopuksi kovin paljoa hyötyä, koska Target, Boid ja Obstacle luokat olivat loppujen lopuksi hyvin erilaisia.

Boid: Tämä on tärkein simulaatio komponentti. Boid-luokassa on mallinnettu Graig Reynoldin artikkelin "simple vehicle model". Lisäksi jokaisella Boid-oliolla on lista käyttäytymisistä, joita se noudattaa. Käyttäytymiset voisivat myös olla globaaleja tiloja simulaatiossa. Kuitenkin se että jokaisella Boid-oliolla on oma listansa, on joustavampi ratkaisu, jos vaikka halutaan muuttaa yksittäisten yksilöiden käyttäytymistä. Tärkein metodi on act(), joka laskee steering-vektorin. Vasta sen jälkeen, kun jokaiselle Boidille on laskettu haluttu suunta, simulaatio kutsuu jokaiselle Boidille move() metodia.

Obstacle/Target: Nämä ovat passiivisia simulaatio komponentteja, joiden act() metodi ei oikeastaan tee yhtään mitään. Ne vaikuttavat paikallaan simulaatiossa Boid-olioiden käyttäytymiseen.

Behavior: Tämä on abstrakti luokka, jonka aliluokkina on suuri määrä erilaisia käyttäytymismalleja. Käyttäytymisiä voidaan yhdistellä ja ne kulkevat usein kokoelmassa Behavior-olioita. Tärkein metodi laskee yksittäiselle Boid-oliolle sen tämän käyttäytymisen osoittaman suunnan. Toteutin flee, seek, separation, alignment, cohesion, obstacle avoidance sekä target seeking aliluokat.

Vec: Tämä luokka kuvaa kaksiulotteista vektoria. Simulaatiossa tehdään paljon vektorilaskentaa, joten oma luokka, jossa on toteutettuna esimerkiksi vektoreiden summaaminen, pistetulo ja ristitulo on parempi vaihtoehto kuin esimerkiksi Tuple-tietotyyppien käyttö.

View: View-olio toteuttaa simulaation näkymän. Se kuuntelee käyttäjän komentoja ja ohjaa simulaatiota sen mukaan. Se myös hoitaa simulaation tilan piirtämisen ruudulle. View tulee toimimaan tiukassa yhteistyössä scala.swing kirjaston kanssa. Simulaatioiden lataamista ja vaihtamista ei toteutettu.

GUI: GUI on yksittäisolio, joka rakentaa käyttöliittymän ja sen käyttämät toiminnot, jonka jälkeen ne yhdistetään View-olioon. Tärkein tarkoitus on selkeyttää View-luokan koodia.

Algoritmit

Uuden paikan laskeminen Boid-oliolle:
steeringForce = truncate (steering_direction, maxForce)
acceleration = steeringForce / mass
velocity = truncate (velocity + acceleration, maxSpeed)
position = position + velocity

Risto Koverola
479699 25.4.2018

Jokaisella käyttäytymisellä on oma tapansa laskea uusi haluttu suunta. Tässä on niistä muutama.

Seek-käyttäytymisen antama vektori:
desiredVelocity = normalize (position - target) * maxSpeed

Arrival-käyttäytymisen antama vektori:
targetOffset = target – position
distance = length (targetOffset)
rampedSpeed = maxSpeed * (distance / slowingDistance)
clippedSpeed = minimum (rampedSpeed, maxSpeed)
desiredVelocity = (clippedSpeed / distance) * targetOffset

Käyttäytymisiä voi yhdistellä suoraan summaamalla yksittäisten käyttäytymisten antamat vektorit tai kertomalla ne ensin painokertoimilla. Toteutuksessani painokertoimet ovat käyttäytymismallien sisällä ja ne ovat siis sopivaksi asetettuja vakioita. Yleensä vektorit ovat niin pitkiä, että voidaan normalisoida haluttu suunta ja kertoa se maksiminopeudella.

Käyttäytymisten yhdistely:
v1 = separation. desiredVelocity
v2 = alignment. desiredVelocity
v3 = cohesion. desiredVelocity
desiredVelocity = normalize(a*v1 + b*v2 + c*v3) * maxSpeed

Tietorakenteet

Ohjelmassa tarvitaan useissa paikoissa muuttuvan kokoisia rakenteita. Esimerkiksi Behavior ja SimComponent oliot kannattaa säilöä Buffereihin, koska niitä lisätään ja poistetaan simulaation ollessa käynnissä. Scalan Tuple rakenteiden sijaan teen vektorilaskennalle oman luokan Vec. Näin saan vektorilaskutoimitukset suoraan tietorakenteen metodeiksi. Päädyin käyttämään simulaatiossa muuttuvatilaisia rakenteita, koska niitä on helppo käsitellä.

Tiedostot

Ohjelma ei käytä tiedostoja.

Risto Koverola
479699
25.4.2018

Testaus

Ohjelmaan ei kirjoitettu varsinaisia yksikkötestejä, kuten alun perin oli suunniteltu, sillä ScalaTest-kirjaston käyttöön ottaminen osoittautui työlääksi ja hyödyt pieniksi. Vec-luokka osoittautui odotettua yksinkertaisemmaksi ja helposti luettavaksi. Ohjelman testaus tehtiin pääosin graafisen näkymän kautta esimerkiksi piirtämällä yksilöön vaikuttavat voimavektorit ruudulle ja seuraamalla yksilöiden käyttäytymistä sekä tulostamalla simulaation aikana arvoja konsoliin. Graafisen käyttöliittymän testaaminen oli sivuutettu suunnitelmassa, mutta myös tämä olisi ollut huomionarvoista.

Eniten pidän koodisani Vec-luokan toteutuksesta, joka on lyhyt ja napakka. Vektorit eivät aiheuttaneet bugeja ja niitä oli helppo käsitellä. Pidän myös käyttäytymisten getSteeringVector() toteutuksista sekä simulaation act() metodissa näitten käyttäytymisten yhdistelemisestä. Koodissa pääsi käyttämään esimerkiksi for-yield silmukoita ja vektoreita summattiin yhteen fold-komennoilla.

Kuitenkin tästä yksinkertaisuudesta johtuen esimerkiksi separation, cohesion ja alignment metodien tehokkuudessa olisi parantamisen varaa ($O(n^2)$) ja ne ovat suurin syy simulaation hidastumiseen, kun ruudulla on suunnilleen sata yksilöä. Koodia myös pitäisi ajaa useammassa säikeessä, mutta tällä hetkellä ohjelma ajetaan kokonaan EDT:ssä. Muutenkin rinnakkaisuutta voisi olla enemmän. Graafinen käyttöliittymä näki useamman iteraation, mutta päädyin käyttämään yläpalkkia, jonka käytettävyys ei ole kaikkein paras. En testannut käyttöliittymää systemaattisesti, minkä takia siinä voi olla bugeja.

Puutteita ohjelmassa on, kun loopataan komponentteja simulaation puolelta toiselle. Käyttäytymismallit eivät ota huomioon toisella puolella olevia komponentteja ja ryhmä saattaa esimerkiksi hajota, kun se ylittää simulaation rajan toiselle puolelle. Obstacle avoidance käyttäytyminen toimii kohtuullisesti, mutta se ei tietenkään aina voi estää törmäysten tapahtumista. Koska mitään fyysistä törmäystä obstaclen ja boidin välillä ei tapahdu, boid siis kulkee obstaclen läpi tietyissä tilanteissa. Esimerkiksi jos suuri määrä boideja yritetään sulkea obstacleista tehtyyn aitaukseen, boidit voivat päästä sieltä ulos.

Poikkeamat suunnitelmasta

Ohjelma noudattaa pääpiirteissään suunniteltua luokkajakoa. Lisäsin olion nimeltä GUI, joka suorittaa graafisen käyttöliittymän rakentamisen. Suurin syy tähän oli ohjelma koodin selkeyttäminen, jotta View luokasta ei tulisi liian epäselvä. Joitain suunniteltuja metodeita (esim addComponent) ja puskureita (esim. simComponents) pilkoin pienempiin osiin, koska niiden pitäminen yhdessä ei tuonut mitään lisäarvoa. Pilkkominen helpotti ohjelmointia ilman, että koodista tuli turhan toisteista.

Merkitsin aikatauluun uudet esineet vihreällä ja toteuttamatta jääneet esineet punaisella. Tuntimäärän merkitsin vihreällä, mikäli esineeseen meni vähemmän aikaa kuin suunniteltu. Checkpointissa päätimme aikaistaa käyttäytymisten toteuttamista, joten ne ovat siirtyneet hieman aikasemmaksi, kuin alkuperäisessä suunnitelmassa.

Risto Koverola
479699
25.4.2018

19.2 - 4.3
-Gitin ja eclipsen yhteen laittaminen, kirjastojen linkittäminen (5h)
-View-luokka niin että ruudulle saa piirrettyä muotoja käyttäjän klikkaamiin kohtiin. (2h)
-Vec-luokka ja yksikkötestit luokalle. (3h)

5.3 – 18.3
-Simulation-luokan tynkä ja Obstacle ja Target luokat, niin että ne voidaan valita valikosta ja lisätä ruudulle. Käyttävät jo step ja act metodeita ja simulaatio pyörii säikeiden avulla. Poistaminen ruudulta onnistuu myös. (6h)
-Simple vehicle model toteutus Boid-luokkaan ja eteenpäin liikkuminen. (3h)

19.3 – 2.4
-Behavior-luokan käyttö Boid olioilla liikkumiseen. (4h)
-Wander, seek ja flee. Käyttäytymisten yhdistely. (6h)
-Säikeistä lukeminen (1h)
-Tehokkuusongelmien ratkaiseminen (8h)

2.4 - 16.4
-Lisää käyttäytymismalleja: separation, cohesion, alignment ja obstacle avoidance. (8h)
-Käyttäytymisten vaihto lennosta ja painokertoimet. (1h)

17.4 - 26.4
-Lisää käyttäytymismalleja ja asioiden tallentaminen ja lataaminen tiedostoon.
-Graafinen käyttöliittymä (9h)

Arvio lopputuloksesta

Kaiken kaikkiaan projekti onnistui todella hyvin. Saavutin tavoitteeni toimivasta parvisimulaatiosta, jossa on myös käyttöliittymä. Omasta mielestäni suurin puute oli wander-käyttäytymisen puuttuminen, joka olisi saanut boidit liikkumaan luonnollisemman näköisesti. Onneksi kuitenkin uusien käyttäytymisten lisääminen on todella helppoa. Vähemmän tärkeitä ominaisuuksia, kuten simulaatioiden tallentamista ja lataamista en saanut toteutettua, mutta se ei juuri jäänyt harmittamaan. Olisin myös halunnut ajaa ohjelmaa useammassa säikeessä ja parantaa tehokkuutta, mutta aika ei riittänyt siihen. Tämän lisäksi yksikkö testaamista olisi ollut kiva harjoitella.

Ohjelmaa voisi parantaa lisäämällä siihen uusia käyttäytymismalleja ja laittamalla se pyörimään useammassa säikeessä. Käyttöliittymää voisi muuttaa intuitiivisemmaksi, kuten suunnitelmassa alun perin oli. Ikkunan kokoa pitäisi voida muuttaa ja ajaa simulaatiota vaikkapa koko näytöllä. Ohjelman koodi on helposti muokattavissa ja lisäysten sekä muutosten toteuttamisen pitäisi olla helppoa, koska koodia ei ole kovin paljoa ja se on kommentoitu. Myös luokkajako onnistui hyvin ja luokkien toteutuksia voisi hyvin vaihtaa tai lisätä kokonaan uusia luokkia. Graafinen käyttöliittymä ja itse simulaatio ovat erillisiä kokonaisuuksia, joten simulaation esillepanoa voi muuttaa helposti.

Risto Koverola
479699

25.4.2018

Viitteet

"Steering Behaviors for Autonomous Characters", Graig W.Reynolds
http://www.red3d.com/cwr/steer/gdc99/

Scala documentation
https://www.scala-lang.org/api/current/

Scala swing menu example
https://github.com/ingoem/scala-swing/blob/master/scala/swing/test/UIDemo.scala

Concurrency in swing
https://docs.oracle.com/javase/tutorial/uiswing/concurrency/index.html