jamk.fi

Unfold

Aleksi Tommila
Janne Möttölä

Harjoitustyö
Huhtikuu 2015

Ohjelmistotekniikka
Tieto- ja viestintätekniikka

JYVÄSKYLÄN AMMATTIKORKEAKOULU
JAMK UNIVERSITY OF APPLIED SCIENCES

Contents

1 Projektin kuvaus

1.1 Toimeksianto

Projekti tehtiin osana "olio-ohjelmointi 1" (IIO10110) opintojaksoa harjoitustyönä. Toimeksiannolla ei ollut määrättyä projektiaihetta, eikä työelämäyhteyttä. Sen sijaan toimeksiantoon kuului arvinointiperusteita, joita työn käytössä edellytettiin. Kirjastot, työvälineet ja teknologiat olivat vapaat. Lähtökohtana oli oliopohjainen ohjelmointi.

1.2 Tavoitteet

Tavoitteena oli tuottaa ohjelma, jonka pohjalta voidaan arvioida "olio-ohjelmointi 1"-kurssin osaamistavoitteita eli olio-ohjelmoinnin ja javan yleisimpien toiminnallisuuksien osaamista. Osallistujien henkilökohtaisena tavoitteena oli täyttää kaikki kiitettävään arvosanaan määritellyt kriteerit.

1.3 Aihe ja sen rajaus

Aiheeksi valittiin shakkilautaa muistuttava ruutupohjainen, hiekkalaatikkotyyppinen evoluutio- ja tekoälysimulaattori, jossa otukset liikkuvat luonnossa keräten tietoa ympäristöstään ja siellä liikkuvista muista otuksista omien fyysisten tarpeidensa ajamina, joka tulee johtamaan muutoksiin otusten käytöksessä. Ohjelma ei ole interaktiviinen, vaan tarkoituksena on tutkia emergenssiä simuloidussa hiekkalaatikossa ohjelman käynnistyksen alussa annettuihin parametreihin vaikuttamalla.

Pois on rajattu kaikki pelilliset elementit. Ohjelmalla ei tule olemaan varsinaista starttia eikä maalia, eikä liikutettavia pelaajahahmoja. Ainoastaan ympäristö ja ympäristön kanssa interaktoivia tekijöitä. Tekoälyn simulointiin paneudutaan ainoastaan käytännöllisellä tasolla, jolloin pois rajaantuu liian tarkka fysikaalisten edellytyksien simulointi, esim genetiikka.

1.4 Kohderyhmä

Aiheen valinnan perusteina on toiminut sen erinomainen soveltuvuus olio-ohjelmoinnin teemoihin ja henkilökohtainen kiinnostus tekoälyä kohtaan. Varsinaista projektiryhmän ulkopuolista kohderyhmää ei ole.

1.5 Saavutettu toiminnallisuus

Hiekkalaatikko simuloidaan käyttäen ruudukkoa. Ruudukon jokainen ruutu on olio kaksiulotteisessa taulukossa. Jokaisella ruutu-oliolla on dataa. Näistä esimerkkinä voi olla vaikka maastotyyppi, hedelmällisyys, suojan määrä, lämpötila jne. Jokaiselle ruudulle on myös maatyypin mukaan määräytyvä oma sprite. Käytännössä tämä tarkoittaa sitä, että vuoristomaatyppiä oleva ruutu tulostuu näytölle vuori-ikonilla, kun taas niitty tulostuu niittyikonilla ja metsä sille tarkoitetulla metsäikonilla.

Itse ruudukon generointi on satunnaismuuttujia hyväksikäyttävä algoritmi, jonka tarkoituksena on pakottaa ruudukon simuloiman luonnon erilaisuus jokaisen ohjelman ajon yhteydessä.

Hiekkalaatikossa liikkuvat oliot luodaan käyttäen satunnaismuuttujia, sekä käyttäjän antamia parametreja. Jokaisella otuksella on dataa, mm. ominaisuuksia millaisista lämpötiloista se pitää, onko se lihan- vai kasvinsyöjä, lauma- vai sooloeläin. Jokaisella otuksella on myös x- ja y-koordinaatit, joidenka mukaan otus tutkii koordinaatteja vastaavan ruudun tietoja ja joidenka mukaan se piirretään ruudulle oikeaan kohtaan.

Otukset toimivat aina tarpeidensa ajamina. Ruudussa on vain tietyn verran ruokaa kasvissyöjille, jonka kuluminen rajaa kasvissyöjän viipymisen mahdollisuudet ruudussa tiettyyn pisteeseen asti. Ruokaa generoituu ruutuihin itsestään hitaasti lisää, mutta mikäli ruokaa ei ole saatavilla ruudussa, niin otuksen stressitaso kasvaa ja se ajaa otuksen siirtymään toiseen ruutuun. Stressitaso on määräävä tekijä otuksen ruudussa pysymisen kannalta ja siihen vaikuttaa ruoan saatavuuden lisäksi useat tekijät, kuten ruudun tarjoama suoja, lämpötila ja ruudussa olevat uhkaavat saalistajat. Mikäli ruutu on sopiva, niin otus on tyytyväinen ja jää ruutuun.

Kasvissyöjät juoksevat ruudun itsensä tarjoaman ruoan perässä, kun taas lihansyöjät haistelevat ympäristöään ja pyrkivät hakeutumaan potentiaalisten saaliiden luokse. Löydettyään potentiaalisen saaliin vainun, saalistajat menevät katsomaan millainen vainuttu otus tarkalleen on ja sen perusteella sitten yrittävät syödä sen tai painavat muistiin kyseisen otuksen olevan liian kova pala purtavaksi.

Otukset kuolevat, jos ne eivät onnistu löytämään itselleen ruokaa, oli se sitten kasvissyöjille ruohoa tai lihansyöjille saalista.

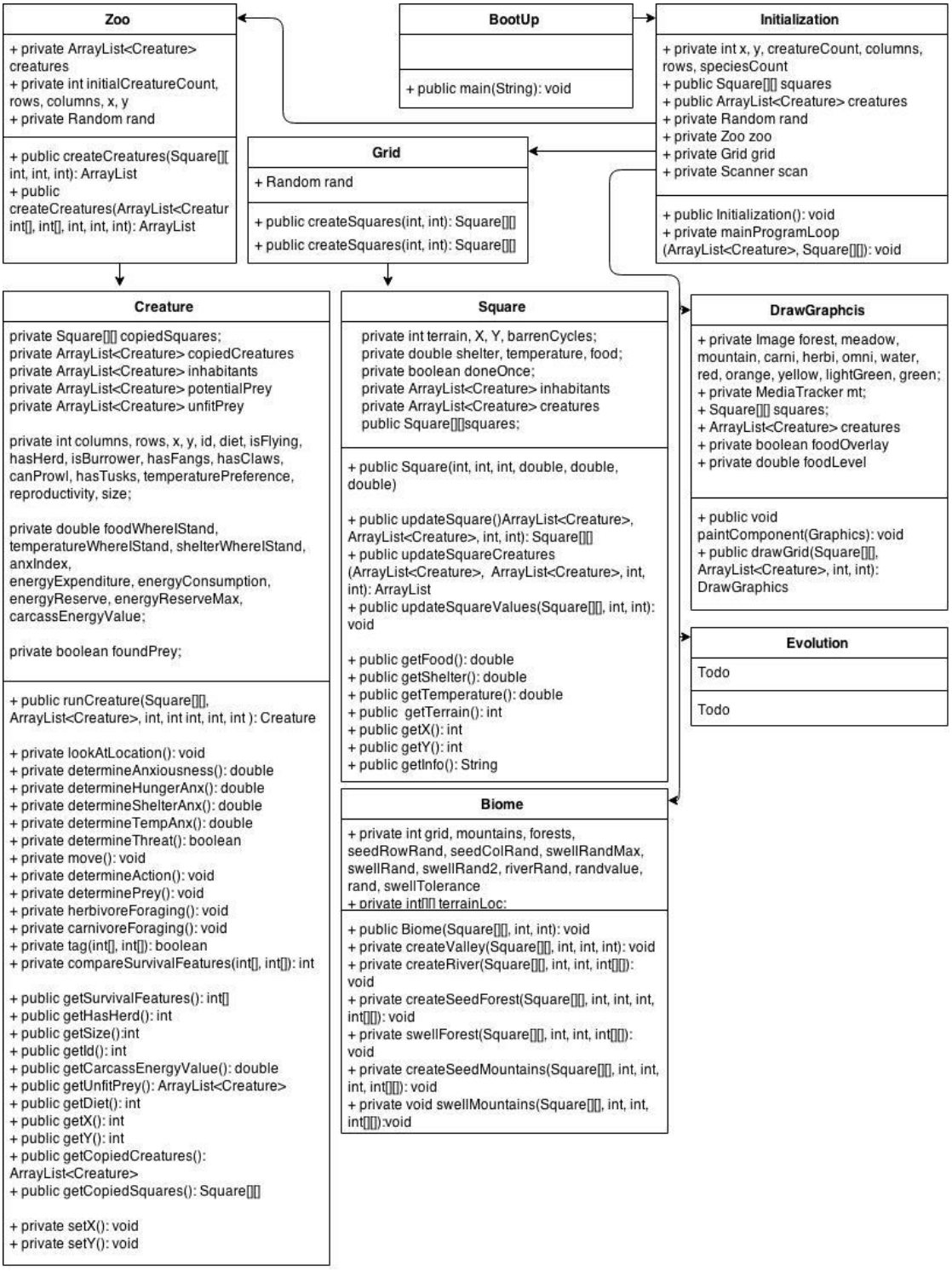
1.6 Pois jäänyt toiminnallisuus

Surullista kyllä, evoluutiosimulaattori jäi toistaiseksi ilman evoluutiota. Ohjelmaan suunniteltiin mm. lisääntyminen, ominaisuuksien periytyminen, muisti, menu ja simulaation pysäytystoiminto, jonka aikana olisi voinut tarkastella olioiden arvoja. Kuitenkaan näitä ei ehditty lisätä ennen deadlinen vääjäämätöntä saapumista ja on selvää, että simulaattori on projektina vielä kaukana valmiista tekeleestä.

1.7 Interaktiivisuus

Tarpeettomat interaktiot rajattiin pois hiekkalaatikon simulaation aikana. Ennen simulaation käynnistystä käyttäjän on mahdollista muokata hiekkalaatikon kokoa ja otusten määrää, joidenka asettamisen jälkeen ei voi enää vaikuttaa kulkevaan simulaatioon.

1.8 Luokkarakenne



Kuva 1, luokkakaavio

Oheinen luokkakaavio (kuva 1) on viimeisimmän simulaattorin version luokkakaavio. Se ei ole välttämättä lopullinen, sillä ohjelmaa saatetaan jatkokehittää.

1.8.1 Ohjelman looginen kulku

BootUp-pääluokassa main metodin sisällä tehdään initializationista olio, jota ajetaan. Instanssissa kutsutaan ensin kerran ajettavia luokkia, kuten Gridiä, Biomia ja Zoota, joiden vastuulla on rakentaa ruudukko ja lisätä sinne otukset käyttämällä luokkia Creature ja Square.

Kun kerran ajettavat toiminnot on suoritettu, siirrytään mainProgramLooppiin initializationin sisällä jossa pyörii ohjelman piirtäminen ja olioiden looginen päivittäminen. Kaikki oliot tekevät toimintonsa omalla vuorollaan ja kun otusten toiminnot on käyty läpi, DrawGraphics piirtää niiden uudet sijainnit ruudukolle kaikki kerralla. Tämä jatkuu kunnes ohjelma sammutetaan.

1.8.2 Luokkien erittelyn perustelu

Luokat ovat eritelty koodin korjaamisen, muokkaamisen ja työnjakamisen yksinkertaistamisen vuoksi loogisiksi kappaleiksi.

1.8.3 Luokkien toiminnallisuus ja ominaisuudet

BootUp-pääluokassa on main-metodi, jossa tehdään olio Initialization-luokasta, jota ajetaan.

Initialization-luokassa kysytään käyttäjältä, kuinka suuret x- ja y-akselit halutaan ja miten monta otusta peliin tehdään. Kun parametrit on annettu, kutsutaan kerran kutsuttavia luokkia joiden tehtävä on rakentaa olioruudukko, randomoida ruudukolle metsät, vuoret, niityt ja lisätä otukset ruudukolle.

Grid-luokka rakentaa käyttäjän antamien x- ja y-parametrien mukaan kaksiulotteisen taulukon, joka täytetään Square-tyyppisillä olioilla. Jokaiselle Square-oliolle on asetettu erinäisiä määriä suojaa, ruokaa ja lämpötilaa. Osa randomoidaan, osa ei. Kaikkien Square-olioiden terrain-tyyppi on alustuksessa plains, johon paneudutaan tarkemmin Biomessa.

Biome-luokka rakentaa varsinaisen maaston. Alustusvaiheessa ruudukko täytetään plains-tyyppisillä Squares-olioilla ja näiden olioiden terrain-parametriä aletaan eri algoritmeilla muuttamaan jotta saadaan ruudukolle metsiä, vuoriklustereita, yksi suuri niitty ja ylhäältä alas etenevä joki. Algoritmit ovat keskenään melko samankaltaisia, sillä kaikissa luodaan random sijainteihin ensin seedejä joita sitten turvotetaan suuremmiksi jotta saavutetaan luontevan näköisiä muodostelmia.

Zoo-luokka on vastuussa otusten lisäämisestä ruudukolle käyttäjän asettaman määrän, kun ruudukko on loogisella tasolla luotu. Se luo ArrayListin johon sijoitetaan Creature-luokasta luotuja olioita, joilla on jokaisella randomoituja ominaisuuksia.

DrawGraphics-luokka vastaa grafiikoiden piirtämisestä. Se luo konstruktoriaan käyttäen simulaattorin grafiikat, eli piirtää jokaisen ruudun sen oman grafiikan mukaan ja Creature-oliot niille kuuluviin koordinaatteihin ruudukolla.

Square-luokasta luodaan olioita Grid-luokassa ja nämä oliot tallennetaan kaksiulotteiseen taulukkoon, josta muodostetaan pelialue. Se sisältää x- ja y-koordinaatit Creature-olioiden sijainnin koordinointia varten, sekä itselleen määriteltyjä geografisia ominaisuuksia, kuten lämpötila, ravinto, maatyyppi jne.

Creature-luokka sisältää suuren joukon otuksille ominaisia piirteitä, jotka pyrkivät mallintamaan erilaisten elävien otusten ominaisuuksia ja toimintoja luonnossa. Nämä ominaisuudet vaikuttavat sen käytökseen ja selviytymiseen maailmassa.

1.8.4 Luokkien suhteet

BootUp-pääluokka käyttää Initialization-luokkaa ja luo siitä instanssin, jota ajetaan.

Initialization käyttää ajonsa aikana ohjelman alustamiseen Grid, Zoo ja DrawGraphics -luokkia. DrawGraphicsia käytetään myös ohjelmaloopissa.

Grid-luokka hyväksikäyttää Square-luokkaa ajonsa aikana ja Zoo-luokka vastaavasti Creature-luokkaa olioiden luomiseen.

1.9 Nykyisen version luokkien eroavaisuudet suunniteltuihin

Joitain luokkia jäi kokonaan luomatta tai hyödyntämättä. Menu- eikä Evolution-luokkiin rakennettu minkäänlaista toiminnallisuutta, joten ne jäivät turhiksi nykyisessä rakenteessa. Pääluokkana ei toimi Initialization, vaan BootUp joka tekee initializationista instanssin. Effects-luokka jäi myös kokonaan tekemättä, joten Creatures-luokka ei myöskään extendaa siitä. DeterminePrey-luokka muutettiin Creature-luokan metodiksi, sillä se oli selkeästi otuksen itsensä tekemä asia.

Vaikka työssä ei päädyttykään käyttämään samaa luokkarakennetta kuin mikä suunnitelmassa esiteltiin, niin se ei ole huono asia. Oli väistämätöntä että suunnitelmat elivät projektin kehittyessä.

Nykyinen luokkarakenne on looginen ja sieltä löytyy melko helposti se mitä halutaan tarkastella, joten siihen ollaan tyytyväisiä.

2 Osallistujat

Tekijöinä projektissa oli Aleksi Tommila ja Janne Möttölä joista molemmat ovat ohjelmistotekniikan ensimmäisen vuoden opiskelijoita. Asiakasta ei ollut, vaan projekti tehtiin ohjelmointikurssin harjoitustyönä. Projektin osallisena arvioivana osapuolena toimii olio-ohjelmointi 1 opintojakson opettaja Pasi Manninen.

Aikaisempaa ohjelmointikokemusta ja harrastuneisuutta on selkeästi enemmän Aleksilla, jolla on takana yksi HAMK:ssa suoritettu tutkinto tietojenkäsittelyn tradenomiksi. Jannella on takana se mitä on tämän kouluvuoden aikana opittu, eli käytännössä yksi C++ kurssi ja tämä kurssi johon harjoitustyö Unfold on tehty.

3 Vastuunjako

Työ pyrittiin jakamaan suunnilleen puoliksi tekijöiden A.Tommila ja J.Möttölä kesken ja työn kaikki osat käytiin läpi molempien toimesta. Molemmat tekijät olivat päätöksenteossa mukana.

Työnjako käytännössä tehtiin luokkakaavion pohjalta, jolloin kummallekin jäi vastuuksi eri luokkien tai niiden toiminnallisuuksien ohjelmointi.

4 Käyttöympäristö, välineet ja teknologia

Suunnitteluvaiheessa hyödynnettiin VioletUML ja www.draw.io UML-editoreja. Varsinaisessa ohjelmoinnissa käytettiin kehitysympäristönä NetBeansia ja paikoin Eclipseä. Ohjelmointikieli oli Java, eikä liitännäiskieliä tai ulkopuolisia kirjastoja ole käytetty.

5 Aikataulu ja tuntimäärät

Projektia alettiin tekemään pari viikkoa tehtävänannon jälkeen. Aikataulua ei alussa paljoa mietitty, vaan projektia vain tehtiin suurempia pohtimatta ja koodaus eteni hyvää vauhtia. Projektia ei ollut vaiheistettu, joten siitä ei myöskään tehty tarkennettua aikataulua. Projektin eteneminen tapahtui muiden opintojen ohessa molempien osallistujien aikataulun salliessa.

	Aleksi		Janne	
Viikko	Tunnit	Alue	Tunnit	Alue
6	6	Creature ja Square oliorakenteet		
7	6	Creature ja Square oliorakenteet		
8	7	Grafiikoiden piirtäminen, renderöinti		
9	7	Ruudukon renderöinti	12	Biomen luonti ja laajennus
10	10	Otusten renderöinti	15	Biomen laajennus
11	15	Creature ja Square luokkien laajennus	15	Biomen laajennus
12	15	Creature ja Square luokkien laajennus	18	Biomen laajennus
13	13	Yleistä luokkien säätämistä	22	Creaturen laajennus
14	10	Zoo & Grid luokat, uudelleen jäsennys	14	Creaturen laajennus
15	6	Renderöinnin uudelleen luomista	16	Squaren ja Creaturen laajennus
16	8	Renderöinnin uudelleen luomista	3	Yleistä luokkien optimointia
17	12	Rend. muutosten peruminen, esityksen valmistelu, loppudokumentaatio	6	Esityksen ja loppudokumentaation valmistelu
Yht.	115		121	

Tuntimäärä sisältää paljon eri toimintojen opiskelua,
debuggausta, läpän heittämistä ja pitsa/kebab paussit, mutta tuntimäärä on arvioitu alakanttiin.

6 Laadunvarmistus, testaus ja ongelmat

Tekniikkana oli koodaa, testaa, paikanna ongelma ja korjaa se.

Työn ohella harjoitettiin voimakasta itsekritiikkiä ja pyrittiin tekemään ohjelmasta optimoitu ja looginen. Vaihtoehtoisista toteutustavoista keskusteltiin usein ja vanhojen koodien pariin palattiin monta kertaa jotta se voitiin tehdä paremmaksi. Työtä myös välillä arvioitutettiin opettajalla jotta saatiin näkemystä erinäisiin toteutustapoihin ja ongelmiin.

Vakavimpia ongelmia koodauksessa tuotti renderöinnin rakentaminen, tiedon kuljettaminen halutuille metodeille ja biomin looginen suunnittelu ja toteuttaminen. Ideoita ratkaisuihin heiteltiin puolin ja toisin, eikä epäkohtiin pelätty käydä käsiksi. Internet tarjosi paljon tietoa jota hyödynnettiin.

Ohjelman varmistettiin toimivan ilman virhetilanteita ja ohjelmoinnin olevan sillä tasolla että siitä saa täydet pisteet, jonka jälkeen keskityttiin jo enemmän muiden kurssien töihin.

7 Tiedonvälitys

Etenemistä seurattiin melkein päivittäisellä keskustelulla koulussa tai internetin välityksellä. Samalla sovittiin työnjaosta, ohjelman ominaisuuksista ja tarkkailtiin aikataulussa pysymistä.

8 Projektin päätös

Harjoitustyö katsottiin valmiiksi kun sen todettiin kertovan olio-ohjelmoinnin osaamisesta riittävällä tasolla. Itse projektia ei katsota kuitenkaan päättyneeksi, sillä siihen on mahdollista lisätä vielä reilusti toiminnallisuutta ja kehitettävää ohjelmaa voi hyvin käyttää tulevaisuudessa työnhaun yhteydessä taidonnäytteenä. Projektiryhmän tavoite saavutettiin koska ohjelmointiosuudesta luvattiin täydet pisteet.

9 Ajatuksia jatkokehittämisestä

Ilmeisimmät jatkokehityksen aiheet ovat puuttuviksi jääneet ominaisuudet ja toiminnallisuus, joita suunnitteluvaiheessa haluttiin tehdä. Lisääntyminen, muisti ja ominaisuuksien periytyminen päälimmäisinä.
Simulaattoria voisi myös helposti viedä toiseen suuntaan. Tässä on hyvä rakennepohja jonkinlaiselle pelille, jossa pelaaja voisi liikutella hahmoa, taistella vihollisia vastaan ja katsoa kuinka pitkään selviää pelissä jossa on metsästettävä ruokaa; jatkokehitysmahdollisuuksia on valtavasti tämänkaltaiselle pohjalle. Ne mahdollisuudet saavat jäädä kuitenkin toistaiseksi odottamaan.

10 Itsearviointi

Kaiken kaikkiaan työstä jäi hyvä maku, vaikka töitä saikin tehdä paljon. Osaamisen tasoon suhteutettuna ohjelmaan suunniteltiin melko paljon toiminnallisuutta, jota lopulta jäi aika paljon pois palautettavasta versiosta.

Kirjoitusrutiini ja kyky etsiä ongelmakohdat koodista parantuivat huomattavasti ja muutenkin kehitystä osaamisessa syntyi työnteon aikana suuria määriä. Siihen on oltava tyytyväinen.