T-93.270: Projektiraportti

Ryhmä 41

15. huhtikuuta 2004

1 Ohjelmakoodi

http://www.niksula.cs.hut.fi/~koblomqv/bouncer/

2 Projektisuunnitelman pitävyys

Projektisuunnitelma piti paikkansa ohjelman yleisrakenteen osalta. Arkitehtuurin ydin controller ja sen riippuvuudet muihin komponentteihin pysyivät koko kehityksen ajan vakaina. Suunnitelmassa esitellyt arkitehtuuriratkaisut eivät olleet ongelma, ainoastaan toteutuksen vaatima työmäärä ylitti arvion. Laajennettavuuden rajaaminen olisi nopeuttanut kehitystä, mutta selvien rajapintojen puute olisi rikkonut projektin laajennettavus-vaatimuksen.

Tuukka: Fysiikan toteutuksen osalta työskentely eteni projektisuunnitelman mukaisesti. Ainoastaan loppupuolella esiintyneet fysiikkaosan bugit aiheuttivat sen, että suunnitelmassa olevaa tavoitetta pallojen välisistä törmäyksistä ei ehditty kokonaan täyttää tämän raportin kirjoittamiseen mennessä.

Aleksi: Vectorizerin ja GameMapin osalta suunnitelma onnistui mukavasti. Ensimmäiset versiot olivat triviaaleja, eli Vectorizer palautti 4 seinävektoria, jotka kiersivät kenttää ulkoreunoja pitkin. Samoin GameMap palautti kaikki mahdolliset vektorit. Lopullinen versio suoriutuu kiitettävästi vektorisoinnista.

Kim: Grafiikkan toteutus oli suoraviivaista ja nopeaa, suunnitelmassa esitelty arkkitehtuuri toimi mainiosti. Suorituskyky oli heti tavittavalla tasolla, joten mitään optimointia ei tarvittu.

Käyttöliittymän toteutusta ei suunnitelmassa oltu tarkasti määritelty. Se ei kuitenkaan haitannut, koska sen tekijällä on tarpeeksi kokemusta asiasta.

Jimmy: Suunnitelmassa ollut verkko-osan kuvaus toimi lähtökohtana varsinaiselle toteutukselle. Kuvaus oli kuitenkin suppeahko ja kuvasi verkkotoimintojen osat vain pääpiirteissään, joten siihen tarvittiin lisäyksiä ennen ohjelmoinnin aloittamista. Toteutuksessa käytettiin mm. useita eri kuuntelijoita (listeners), joita ei projektisuunnitelman arkkitehtuuriluonnoksessa oltu mainittu. Eri rajapintoja suunniteltiin muutenkin lisää, ja lopullinen arkkitehtuuri onkin huomattavasti monimutkaisempi kuin alkuperäisessä suunnitelmassa. Verkko-osan rungon perusarkkitehtuuri noudattee kuitenkin pääpiirteissään alkuperäistä suunitelmaa.

3 Arviointi saavutuksista

Yleensä ottaen yhdessä työskentelyyn ei löytynyt sopivaa aikaa, joten jouduttiin koodaamaan yksinään, eikä pystytty antamaan toisille neuvoja ja apua siinä määrin kuin sitä olisi ehkä tarvittu. Tämän ja henkilökohtaisten aikataulujen kireyden takia ei kaikkia projektin aiheessa mainittuja asioita saatu toteutettua ennen projektisuunnitelman palauttamista.

Vaikka suunnitelmassa todettiin, että tiukan aikataulun pitämiseen vaaditaan hyvä ryhmän sisäinen kommunikointi, ei tämä selvästikkään riittänyt, vaan vasta yhdessä täyspäiväinen koodaaminen alkoi tuottaa riittävää edistymistä.

Tuukka: Ohjelman fysiikan toteutuksessa tärkeimmät tavoitteet newtonilaisen fysiikan sekä seinätörmäysten implementoimiseen saavutettiin. Vaikka törmäyslaskentaan tutustuttiin etukäteen lukemalla aiheeseen liittyviä kirjoituksia internetistä, eivät valmiit metodit ja kaavat pudonneet syliin yhtä helposti hakukoneen avulla, vaan ne piti tehdä itse. Laskutoimitusten kehittäminen tyhjästä oli yllättävän haastavaa, mutta silti luultavasti harjoitustyön antoisin osa. Suunnitteluvaiheesta suurin osa ajasta kului erilaisten seinätörmäystarkistusten kehittämiseen ja vertailuun, mikä oli yllättävän miellyttävää, kaikkien toteustapojen vielä oltua avoinna. Tavoitteena oli saada aikaan mahdollisimman tehokas törmäystarkistus, ja siinä onnistuttiinkin, ottamalla laskuissa huomioon mahdollisimman vähän seiniä.

Matka teoriasta käytäntöön on pitkä, ja varsinaisen törmäystoteuksen koodaaminen osoittautui vaikeaksi, johtui se sitten vähäisestä ohjelmointikokemuksesta tai liian tiukasta aikataulusta. Ensimmäisissä testeissä, jotka toteutettiin vasta kun seinätörmäysfysiikka oli olevinaan valmiiksi koodattu, havaittiin lukuisia ongelmia (kiihtyvyyden puute, seinien havaitsematta jättäminen). Tämä ei ollut yllätys, sillä oli varauduttu siihen ettei kaikki ensimmäisellä kerralla toimi. Mutta debuggaus olikin oletettua raskaampaa, ja vasta parin täysipäiväisen debuggaus -session jälkeen saatiin seinintörmäystarkistukset siedettävään kuntoon. Tätä kirjoittaessa kaikkia seinätörmäyksiä ei havaita täydellisesti, ja pientä fysiikkamoottorin nikottelua esiintyy, mutta kokonaisuudessaan järjestelmä täyttää sille aseitetut tavoitteet. Ainoastaan partikkelien keskenäisien törmäysten tarkistusta ei ehditty toteuttaa debuggauksesta johtuneen aikapulan takia.

Aleksi: Vectorizer:ia tehdessä saavutettiin tavoitteista oikeastaan kaikki niille asetetut tavoitteet, eli koodin laajennettavuus, selkeys, erityisesti että selkeästi eri osat ovat eroteltuina eri metodeikseen ja ehkä dokumentointi. Vectorizerissa tehdessä tuotti ongelmia kordinaatiston vasenkätisyys, eli oikeakäsisääntö ei vektoreiden ristitulossa toiminut. Myös sopivan algoritmin kehittäminen tuotti ongelmia. GameMap:kin vaati pientä vääntöä virheiden ja algoritmin vuoksi, jotka saatiin onneksi korjattua.

Jimmy: Koska itselläni ei ollut aikaisempaa kokemusta verkko-ohjelmoinnista, eikä kovinkaan mittavaa ohjelmointikokemusta Java-kielellä, kului uusien asioiden opettelemiseen kohtalaisen paljon aikaa. Projektin loppuvaiheessa jouduttiinkin hieman tinkimään verkko-osan vaatimuksista, koska aika kävi vähiin. Suurimman osan ohjelmointityöstä tein pääsiäisloman aikana, ja jälkikäteen arvioituna olisi toteuttaminen ehkä ollut syytä aloittaa aikaisemmin. Raporttia kirjoitettaessa ei suunnitelmassa mainittua UDP-protokollaan perustuvaa tiedonsiirtoa oltu toteutettu toimivasti, lisäksi mm. dokumentoinnissa ja virhetilanteiden käsittelyssä olisi parantamisen varaa.

Sunnitelmassa mainitut riskit liittyen aikatauluun ja uusiin tekniikoihin kävivät siis omalla kohdallani toteen.

Alunperin ajateltu verkko-osan "itsenäisyys" eli käyttökelpoisuus jatkossa muissa ohjelmistoissa toteutui osittain, sillä tiedonsiirtoon käytettävät osat ja kuuntelijoihin perustuva arkkitehtuuri on siirrettävissä myös muunlaisiin ympäristöihin. Tiedonsiirtoluokkia käyttävä ja kuuntelijarajapinnat toteuttava verkon osa on lähemmässä vuorovaikutuksessa ohjelman kontrolleriosan kanssa, joten sen siirrettävyys suoraan muihin toteutuksiin on hieman huonompi. Toisaalta sen tarjoama yksinkertainen rajapinta voi tosin olla jopa hyödyllinen muissa projekteissa.

Kim: Grafiikan toteutus jäi ehkä liian myöhäiseen vaiheeseen, jolloin fysiikan ja vektorisoinnin alkuversioden testaus ei ollut helppoa. Tosin grafiikka oli melko nopeasti valmis kun sitä oikeasti tarvittiin. Grafiikan toteutuksessa oli pieni synkronointiongelma, joka tosin ratkesi nopeasti kun ongelmaan pureuduttiin.

Käyttöliittymään tuhrautui ihan liian paljon aikaa. Peli-ikkuna saatiin helposti koon pienentämistä kestäväksi, tosin muiden ikkunoiden optimointi ei ollut antoisaa eikä helppoa, joten siitä luovuttiin. Oikestaan JFC:n monipuolisuus oli sekä hyödyllistä että haitallista. Kaiken sai tehtyä, mutta se ei aina ollut niin helppoa kuin se voisi olla, vaikka siitä on jo kokemusta ja osasi odottaa ongelmia.

Controlleri ei oikeastaan ollut niin ongelmallinen kuin miltä tuntui. Tosin verkko-osan synkronisointi ei vielä ole käytössä, joten ehkä pahin on vielä edessä.

Voi kuitenkin todeta, että vaikka suunnitelmaa varten ei kaikki tullut valmiiksi on mahdollista,

että demoon mennessä saadaan kaikki perusvaatimukset täytettyä. Erityisesti, mielenkiintoisen peliloogiikan lisääminen ei pitäisi nykyisellä laajennettavuudella olla ongelma.

Työskentelytavoista voi mainita, että cvs-palvelimen käyttö, vaikka siitä koitui vähän lisää työtä, oli kyllä hyvin käytännöllinen. Useammin kuin kerran editoitiin eri kohtia samasta tiedostosta yhtä aikaan, mutta tästä ei koitunut ongelmia. Antin käyttö oli onnistunutta, tosin sen puuttuminen atk-keskuksen koneilta aiheutti hieman lisätyötä. Projektin aikana esittelin myös log4j-kirjaston, jonka käyttö oli suositeltavaa, koska eteenkin monisäikeisissä kohdissa se helpotti debuggausta.

4 Toteutunut ajankäyttö

Tuukka: Ajankäyttö fysiikan toteutuksessa toteutui muuten suunnitelman raameissa, mutta loppupuolella esiintyneet yllättävät ongelmat debuggauksineen veivät aikaa ylenmäärin.

Päivä	Tuntia	Aihe
ennen 17.3	6	suunnittelua
17.3	4	projektisuunnitelman tekoa
25.3	2	opiskelua
26.3	3	koodausta
27.3	5	suunnittelua
28.3	3	koodausta
30.3	2	koodausta
31.3	2	suunnittelua
1.4	2	suunnittelua
3.4	6	suunnittelua
4.4	5	suunnittelua ja koodausta
5.4	5	suunnittelua ja koodausta
8.4	8	koodausta
9.4	9	koodausta
10.4	10	koodausta
11.4	11	debuggausta
12.4	10	debuggausta
13.4	5	koodausta ja raportin tekoa
14.4	5	koodausta ja raportin tekoa
Yhteensä	91h	

Aleksi: Ajankäyttö Vectorizerin ja GameMapin toteutuksen puolesta toteutui, mutta näistä vastannut henkilö ei kerennyt auttaa muita.

Päivä	Tuntia	Aihe
25.2	1.0	Ensimmäinen kokous, aihe.
3.3	1.0	Toinen kokous, meta:mitä.
10.3	2.0	Kolmas kokous, meta:miten.
14.3	4.0	Alustavaa algoritmien suunnittelua.
18.3	3.0	Projektisuunnitelman tekemistä, alustavaa suunnittelua.
19.3	2.0	Lisää projektisuunnitelmaa, algoritmien vertailua.
20.3	1.0	Lisää projektisuunnitelmaa.
23.3	3.0	Alustavaa koodausta.
26.3	6.0	Koodausta. Algoritmien vertailua. Iteroimista.
3.4	5.0	Pääasiassa vectorizerin koodausta.
7-8.4	10.0	GameMapin koodausta. Vectorizerin debuggausta.
11.4	1.5	Koodausta. GameMapin debuggausta
12.4	4.0	Debuggausta.
13.4	10.0	Debuggausta. Koodausta.
14.4	7.0	Raportti. Koodin kommentointi.
Yhteensä	59h	

Kim: Projektin hallintaan, erityisesti rajapintojen työstämiseen, kului säännöllisesti arvioitua enemmän aikaa. Controllerin ja grafiikan toteutukset pysyivät aikaraameissaan, mutta kuten jo mainittiin Swing-kirjaston käyttö oli aikaa vievää. Tosin sekin oli jo arvattavissa.

Päivä	Tuntia	Aihe
25.3	1	Ideointia
3.3	1	Suunnittelua
10.3	1	Suunnittelua
18.3	2	GameData:n tukiluokkien toteutus
17.3	3	Projektin suunnittelua ja projektisuunnitelman laatimista
17.3	4	Suunnitelman kirjoittamista
18.3	3	Suunnitelman tekemistä
20.3	8	Suunnitelman tekemistä
21.3	1	Suunnitelman viimeistely
23.3	4	Suunnittelua ja rajapintojen toteutusta
25.3	2	Rajapintoja
26.3	4	Rajapintoja
29.3	2	Koodausta
30.3	6	Grafiikkaa
03.4	3	Grafiikkaa ja kontrolleri
04.4	5	Koodausta
06.4	3	Koodausta
08.4	4	Koodausta
10.4	10	Fysiikka ja grafiikan debuggaaminen toimintaan.
11.4	12	Käyttöliittymä ja debuggaus.
12.4	6	Käyttöliittymä
13.4	9	Käyttöliittymä ja kontrolleri
14.4	14	Raportti.
Yhteensä	108h	

Jimmy:

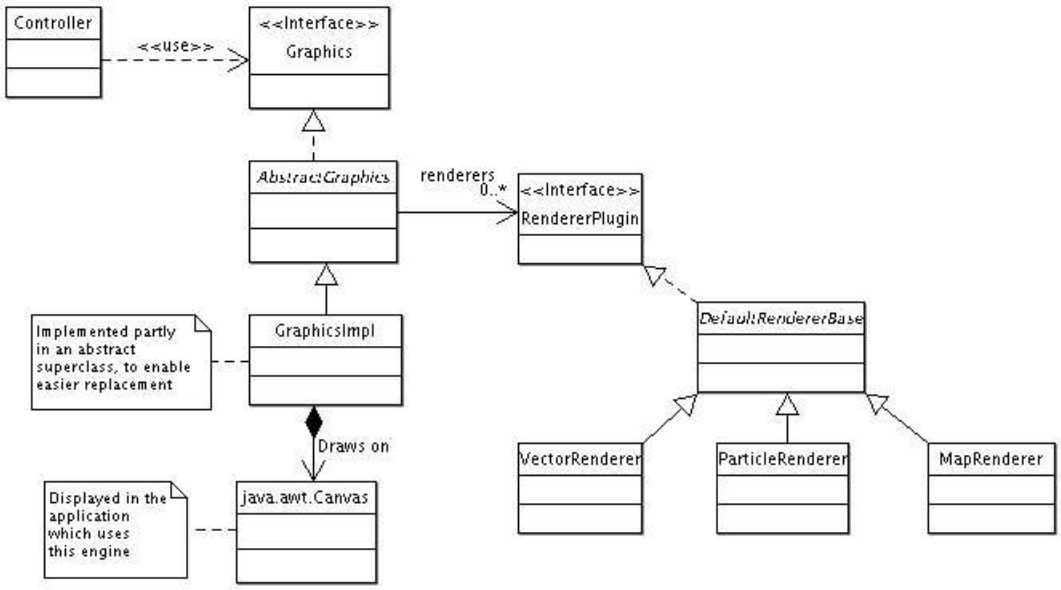
Viikko	Tuntia	Aihe
12-14	10	Suunnittelua ja toteutuksen aloittelua
15	25	Ohjelmointia (ja vähän suunnittelua)
16	25	Ohjelmointia, lisäksi raportin kirjoittamista
Yhteensä	60h	

5 Arkkitehtuurikuvaus

Ohjelma on toteutettu kolmeen periaatteelliseen kerrokseen. Ylimpänä, toteuttaen interaktion käyttäjän kanssa on käyttöliittymäkerros. Sen alla on peliä ohjavaa kerros, Controlleri ja sen dataluokat. Pelin kontrolleri hoitaa synkronoinnin peliin kytkettävien implementaatioden välillä. Alimpana ovat Controlleriin kytketyt implementaatiot eri osa-alueista.

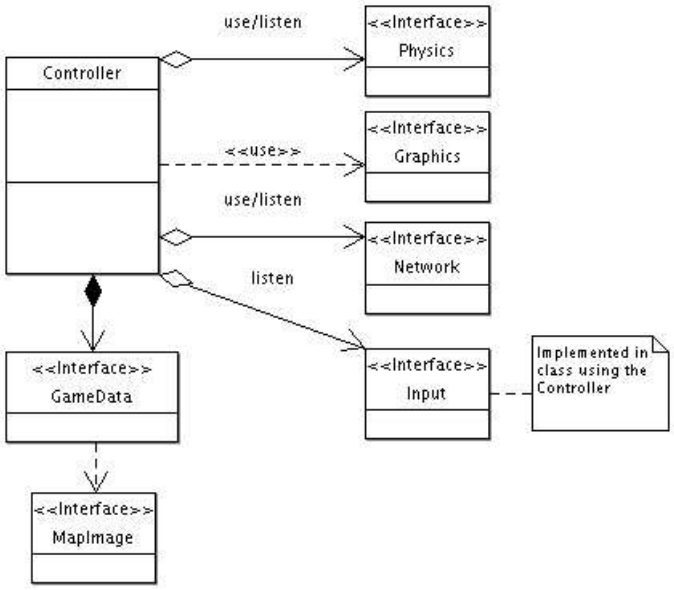
Kim: Käyttöliittymäkerros koostuu Launcher-luokasta (kuva 1), joka käyttää JFrame-luokkaa ikkunan näyttämiseksi käyttäjälle. AbstractWindow-luokka määrittelee vaihdettavan ikkunasisällön, joka näytetään JFrame:illä. MainWindow luo oman instanssin Controllerista näyttääkseen esikatselun. Launcher luo myös instanssin Controllerista, jonka grafiikka näytetään GameWindow:sissa. Tämä arkitehtuuri toimii mielestäni hyvin ja on ollut nopein Swing-käyttöliittymä tehdä. Tosin koska tapahtumienkäsittely on kuitenkin vielä Launcherissa, ei tämä ratkaisu kuuntelijan suhteen skaalaudu hyvin suurempaan määrään ikkunoita.

Controller on tilakone joka sitoo yhteen eri alemman tason komponentit. Samalle tasolle kuuluu myös GameData-rajapinta. Sen avulla määritellään yleisiä rajapintoja joita muut komponentit voivat vastaanottaa, katso kuva 2. Controllerilla on omat tähän peliin soveltuvat implementaatiot näistä rajapinnoista. Controller ajaa omassa säikeessään käyttöliittymästä riippumatta, näin saadaan esim. tapahtumat käsiteltyä nopeasti vaikka fysiikka laskee parhaillaan jotain operaatio-



Kuva 1: Käyttöliittymäluokat

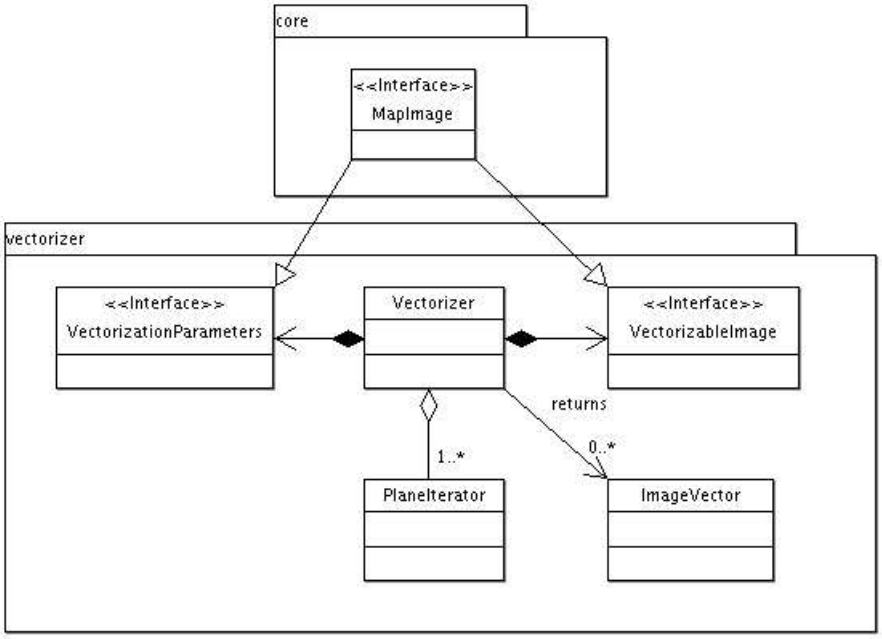
ta. Koska verkko osa ei ole valmis, on Input-rajapinnan implementaatio kytketty pelaajan pallon ohjaamiseen.



Kuva 2: Ydinluokat

Grafiikka on toteutettu täysin suunnitelman mukaan; AbstractGraphics-luokkalle voi rekisteröidä RendererPlugin:eja dynaamisesti. GraphicsImpl tarjoaa Controllerille kankaan jonne piirtopluginnien tulos päivitetään aina pyydettäessä. Koska tuplapuskuriin piirto tapahtuu Controllerin säikeessä ja ruudulle päivitys AWT:n Event-säikeessä, täytyy varmistaa ettei tuplapuskuria muuteta kun ruutua päivitetään.

Aleksi: Vektorisoija vektorisoi kuvan kahden rajapinnan avulla, VectrorizationParameters määrittelee kuinka VectorizableImage pitää vektorisoida. MapImage toteuttaa molemmat näistä rajapinnoista.



Kuva 3: Vektorisoija

Vectorizer käy läpi kaikki pikselit. Jos se löytää alueiden rajan, jolla ei olla käyty (tarkistaa sen haveBeen- taulukosta), kutsuu se PlaneIteratoria. PlaneIterator on olio, joka seuraa kahden eri alueen välistä rajaa. PlaneIteratorin luomisen yhteydessä laitetaan iterVector:in palauttamia vektoreita listaan imageVectors-listaan. IterVector kutsuu moveToNextCross:ia toistuvasti ja ylläpitää vektoria, jonka katkaisee sopivassa kohdassa. MoveToNextCross muuttaa iterpos-vektoria ja vector-vektoria niin, että uusi iterpos ilmaisee seuraavaa rajan risteystä. Tämä käyttää searchDots-metodia. SearchDots muuttaa parametreina annettuja MutableVectoreita siten, että ne ilmaisevat edessäolevien pikseleiden paikkoja. Kun näiden koordinaateilla kutsutaan moveToNextCrossissa (vectorizer:in) getPixelZone(x,y), saadaan vastaavat alueitten koodit. Näiden avulla selvitetään, onko seuraava risteys suoraan edessä vai tarvitaanko kääntyä. Samalla paikka merkitään läpikäydyksi.

GameMap:ssa on kaksi toimintoa: put* ja get*. (*=ImageVector tai Particle). GameMap:ssa koko kartta jakaantuu 100:n osaan, joiden sisällöstä huolehtii Area-oliot. Kun ImageVector laitetaan kartalle, selvitetään mihin Area-olioihin se kuuluu ja laitetaan se sinne. Samoin Particle:n kanssa. Kun taas kutsutaan GetImageVector kaikki annettua pistettä annetulla etäisyydellä olevat Area:t käydään läpi ja niistä otetaan niiden sisältävät ImageVector:it.

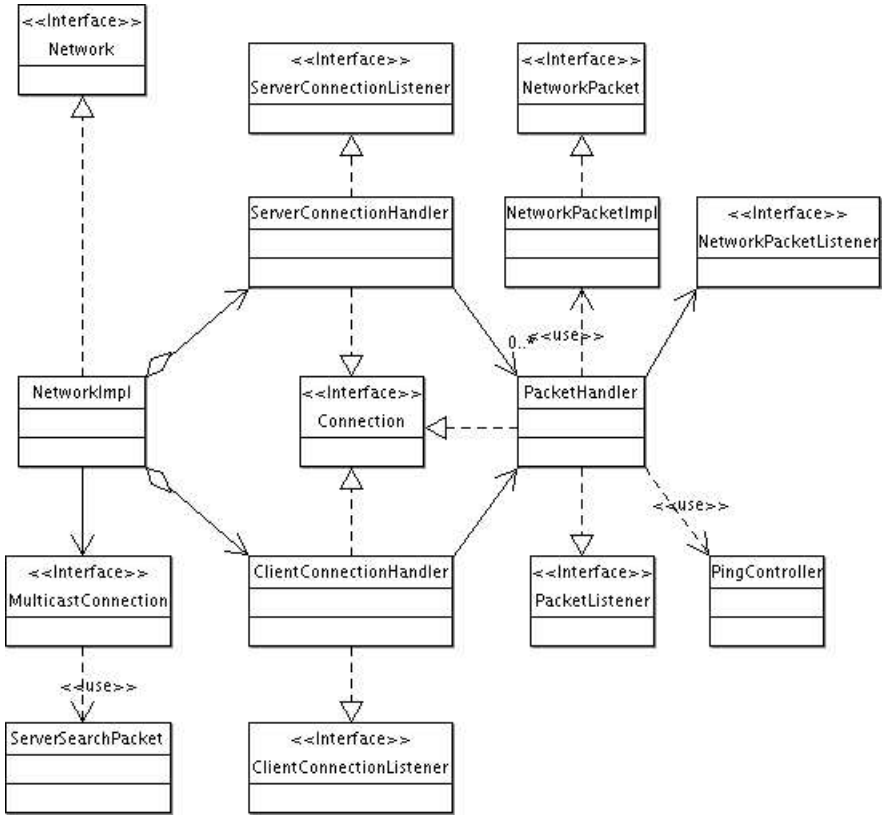
Jimmy: Ohjelmiston arkkitehtuurin sanallinen esitys

Verkko-osan "pääluokkana"toimii NetworkImpl-luokka, joka toteuttaa core-osan Network-rajapinnan. Sen avulla etsitään verkosta servereitä ja otetaan yhteys sellaiseen, tai käynnistetään oma serveri. Sen kautta myös lähetetään paketteja verkon yli kun yhteydet on asetettu. Pääluokka kommunikoi core-osan kanssa pääosin NetworkPacketListener-rajapinnan avulla.

Serverien etsiminen tapahtuu käyttämällä multicast-yhteyttä (MulticastConnection). Serverinä toimivat tietokoneet kuuntelevat UDP-protokollalla lähetettäviä multicast-viestejä (jotka ovat ServerSearchPacket-tyyppisiä olioita), ja ennen varsinaisen yhteyden muodostamista käytetään TCP-pohjaista tiedonsiirtoa kuvaustietojen siirtämiseksi.

Kun liitytään palvelimeen, on avattava uusi client-yhteys (Client), joka ottaa yhteyden valittuun serveriin (Server). Client-yhteydelle asetetaan ensin yhteyskuuntelija (ClientConnectionListener), joka valitussa toteutuksessa on ennalta luotu yhteyskäsittelijä (ClientConnectionHandler). Yhteyskäsittelijä vuorostaan asettaa uudelle client-yhteydelle pakettikuuntelijan (PacketListener), joka tässä toteutuksessa on pakettikäsittelijä (PacketHandler).

Pakettikäsittelijän avulla voi yhteyttä (Connection) pitkin lähettää paketteja. Lähetettävät



Kuva 4: Ytimeen liitetty verkko-osa

paketit ovat olioita, jotka toteuttavat Serializable-rajapinnan. Pakettikäsittelijä muokkaa oliot verkkopaketeiksi (NetworkPacketImpl), jotka sisältävät aikatietoa verkon viiveen laskemiseksi. Nämä paketit lähetetään sitten verkon yli.

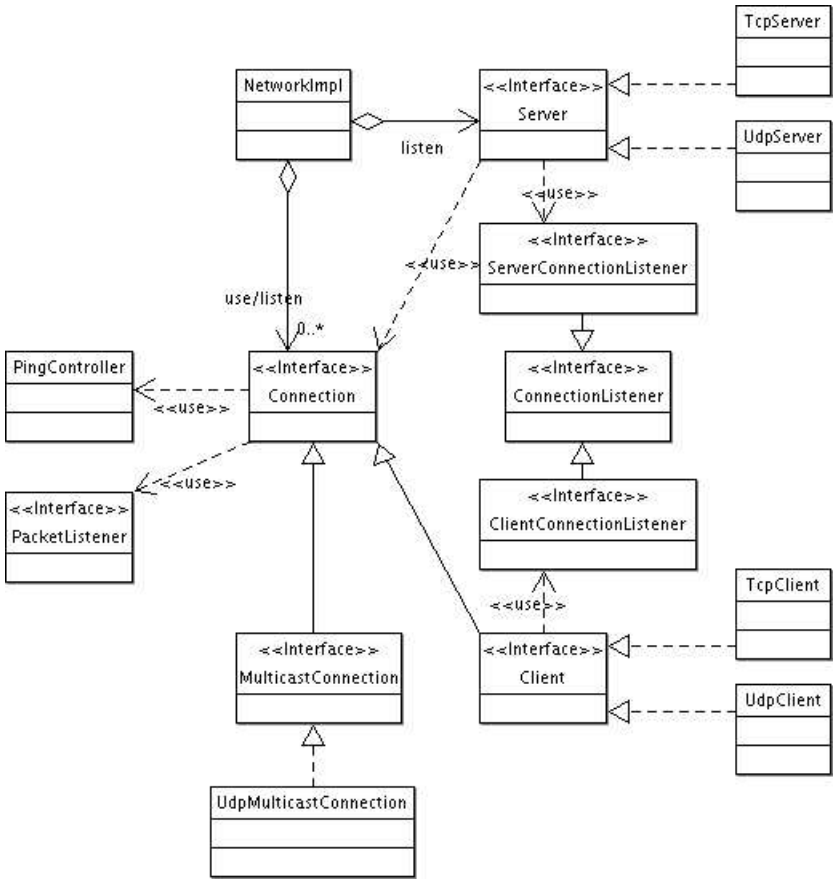
Pakettikäsittelijä hoitaa myös yhteyden kautta vastaanotettujen pakettien käsittelyn. Viivelaskijan (PingController) avulla asetetaan paketille viive, minkä jälkeen NetworkPacket-rajapinnan toteuttava paketti siirretään core-osaan NetworkPacketListener-tyyppiselle käsittelijälle. Pakettikäsittelijä voi myös katkaista yhteyden.

Serverinä toimittaessa vastaanotetaan clientien yhteyksiä, ja luodaan server-tyyppisiä "vastayhteyksiä". Serverille asetettu yhteyskuuntelija (ServerConnectionListener) on server-tyyppinen yhteyskäsittelijä (ServerConnectionHandler), joka asettaa saaduille yhteyksille pakettikäsittelijät, jotka se tallentaa. Näiden pakettikäsittelijöiden avulla voi kaikkien yhteyksien kautta lähettää paketteja, ja yhteydet voi myös katkaista (vastaavasti kuin client-tapauksessa, jolloin yhteyskäsittelijällä on vain yksi yhteys).

Verkon yli otettavat yhteydet voidaan jakaa paitsi client- ja server-tyyppeihin, myös TCP- ja UDP-pohjaisiin yhteyksiin. TCP-protokollaa käytetään luotettavaan tiedonsiirtoon, UDP:llä taas lähetetään pieniä paketteja joiden pitää päästä nopeasti perille ilman takuita. Käytännössä client-yhteyksille on siis erilliset TcpClient ja UdpClient -luokat, ja servereillä vastaavasti myös eri luokat. (Raporttia kirjoitettaessa UDP-tiedonsiirtoa ei tueta; lähiverkkoympäristössä TCP-yhteydet toimivat kuitenkin riittävän nopeasti).

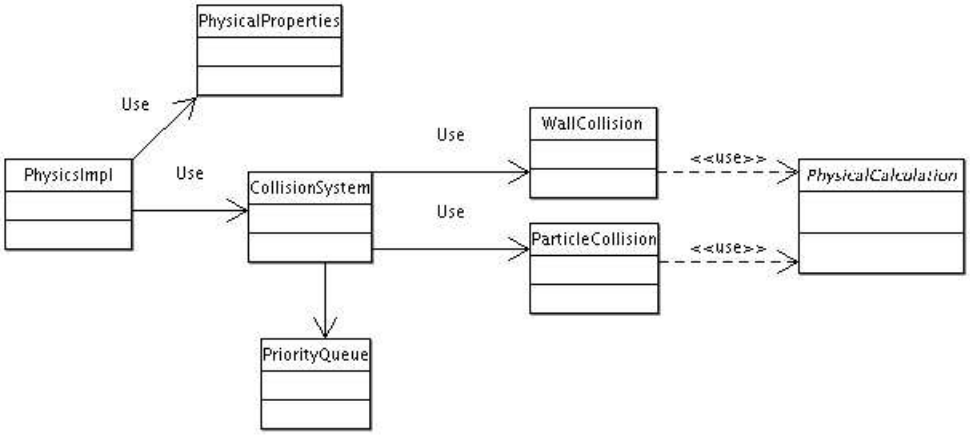
Arkkitehtuurin onnistuneisuus

Arkkitehtuuri ei eri kuuntelijoineen ja rajapintoineen ole kovin suoraviivainen. Tämä tuotti toteutusvaiheessa ongelmia, kun monia eri riippuvuuksia piti ottaa huomioon. Kuuntelijoihin perustuva rakenne on kuitenkin suhteellisen siisti, eikä tuhlaa tietokoneen resursseja. Verkon toteutuksen eri osat olisi vielä voinut eriyttää eri moduuleihin siten, että kunntelija-arkkitehtuuri olisi erillään pelille ominaiset toiminnot toteuttavista luokista. Tältä osin nykyinen rakenne kuitenkin noudattaa



Kuva 5: verkko-osan sisäinen toteutus

alkuperäistä suunnitelmaa.



Kuva 6: Fysiikka-osan luokat

Tuukka: MISSING

6 Mitä opimme

Tuukka: Fysiikan toteutuksen parissa työskentely havainnollisti hyvin, että metodien ja järjestelmien testaaminen kannattaa aloittaa mahdollisimman aikaisessa vaiheessa, sillä ongelmat yleensä propagoituvat alemmilta tasoilta ylemmille tasoille, jolloin ongelman aiheuttajan selvittäminen voi olla vaikeaa. Laskutoimituksia kehittäessä on syytä miettiä huolellisesti kaikki poikkeustapaukset ja tilanteet, joita saattaa syntyä. Lisäksi laskujen tarkistaminen muutamaan otteeseen on suotavaa. Myös koodi ja järjestelmän yleinen logiikka tulisi tarkistaa.

Ahertaessa deadlinen alla voi helposti ajautua rankkaan työskentelyputkeen, joka ei taukoja eikä virkistäytymishetkiä tunne. Tärkeä osa työn onnistumisessa (ja oman työkyvyn ylläpitämisessä) on asettaa rajat omalle työmäärälle ja päivittäiselle rehkimiselle. Päiviä jatkuva liki tauoton työskentely osoittautui huonoksi ideaksi.

Jimmy: Verkko-osaa tehdessäni opin ensinnäkin verkko-ohjelmointia Javalla, eli tietoliikenneominaisuudet toteuttavien luokkien (esim. Socket-luokkien) käyttämistä sekä client/server-mallin toteuttamista. Verkko-ohjelmointiin liityi myös säikeiden käyttö ja rinnakkaisohjelmointi, joiden opetteluun kului jonkin verran aikaa. Lisäksi opin verko-osassa käytetyn kuuntelija-arkkitehtuurin soveltamista. Suunnitelmaan kirjatut oppimistavoitteet siis toteutuivat osaltani.

Lisäksi huomasin, että asioiden opettelemiseen kului enemmän aikaa kuin olin ajatellut. Olisi siis syytä aloittaa toteuttaminen ajoissa, koska loppuvaiheessa ohjelmavirheiden korjaamisessakin kuluu oma aikansa.

Aleksi: Kannattaa ajatella ja keksiä keinoja debugata koodia jo alusta asti. Näin saadaan helposti myös myöhemmät muutokset nopeasti selville tarkistamalla koodi ensin ensimmäisillä debuggaustesteillä. Kannattaa myös testata vaikka käytännössä kaikki mahdollinen pariin otteeseen.

Kim: Mitään sudenkuoppia ei eteen tullut, mutta kyllä ongelmia riitti. Luulen, että Netbeansin ja cvs:n käytön aloittaminen osottautui muille luultua hankalammaksi. Perusasiat olisi varmaan kannattanut käydä läpi kahdesti, ja rauhallisessa ympäristössä.

Arkitehtuuri lyötiin lukkoon jo aikaisessa vaiheessa, mikä selvensi asiota, tosin jotkin rajapinnat eivät olleet varmoja tarpeeksi aikaisin. Suurin syy aikataulun venähtämiseen oli kuitenkin varmaan tarpeeksi tarkan luokkasuunnitelman puute. Arkitehtuurin lisäksi olisi jokaisesta osasta tehdä tarkka luokkakaavio ennen toteuttamista.