AS-0.3301 - projektidokumentti

Aihe - ownsubject5

Tekijät:

78713T Matti Pekkanen matti.pekkanen@tkk.fi

80391C Juho Salmi juho.salmi@aalto.fi

53761M Riku Luostarinen riku.luostarinen@tkk.fi

Viimeksi päivitetty:

14.12.2010

Ohjeet kääntämiseen ja käyttämiseen

Hyper Fabio Brothers Battle! Toimii Linux-käyttöjärjestelmillä, joissa on ClanLib-kirjasto asennettuna. Ohjelman hakemistosta löytyy yksinkertainen Makefile-tiedosto, jonka avulla voi kerralla kääntää kaikki ohjelmaan liittyvät tiedostot.

Siirry ensin hakemistoon, jossa koodi-tiedostot ovat:

```
horus ~ 1278 % cd ownsubject5/src
```

Ohjelma käännetään siis seuraavasti:

```
horus src 1279 % make
```

Kun ohjelma on käännetty, se ajetaan komennolla:

```
horus src 1280 % ./mario
```

Ohjelman käynnistyttyä siirrytään päävalikkoon (kuva 1.). Päävalikossa on kolme vaihtoehtoa:

1. New Game – aloittaa uuden pelin.
2. Options – asetusvalikko, josta voi vaihtaa pelaajien näppäimet.
3. Quit – lopettaa pelin.

Valikossa voi liikkua joko hiirellä tai numeronäppäimillä.

Kuva 1. Päävalikko

Options-valikossa (kuva 2.) löytyy lista pelaajien ohjaamiseksi käytettävistä näppäimistä. Vakiona ne ovat:

pelaaja 1: Oikea: Oikea nuoli, Vasen: Vasen nuoli, Hyppy: Ylänuoli

pelaaja 2: Oikea: D, Vasen: A, Hyppy: W

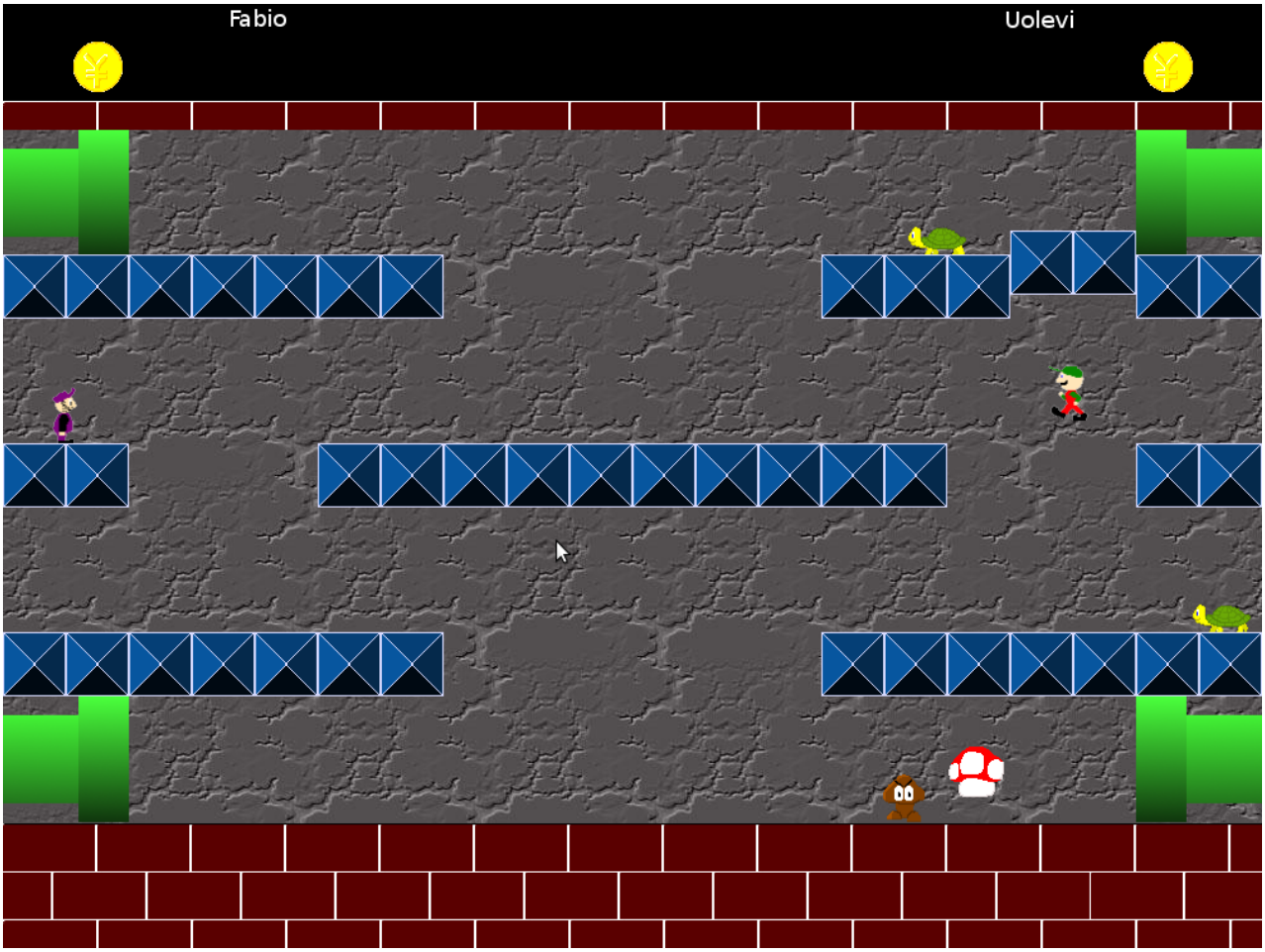
Valikossa napsauttamalla hiirellä mitä tahansa näppäintä, peli kysyy käyttäjältä uutta näppäintä. Käyttäjän painettua nappia se tallentuu halutuksi näppäimeksi. Valikosta pääsee pois valitsemalla Back.

Itse peliin pääsee päävalikosta valitsemalla vaihtoehdon 1. New game. Pelin alussa ruutuun aukeaa viemärimaisema (kuva 3.), jossa on kaksi hahmoa, Fabio (punavirheä hahmo, pelaaja 1) ja Uolevi (violetti hahmo, pelaaja 2). Pelin ajatuksena on kerätä kultajenejä kilpaa. Pelin aikana jaetaan viisi kolikkoa ja pelaajista se, joka kerää enemmän kolikoita, voittaa pelin. Pelaaja voittaa pelin myös siinä tilanteessa, että toinen pelaaja kuolee osuttuaan hirviöön.

Kuva 2. Asetusvalikko

Hirviöitä on kahdenlaisia, Koopia ja Goombia. Koopat ovat hitaita, mutta raskaasti panssaroituja, ja niiden tappaminen on vaikeampaa kuin Goombien. Goombat ovat taas nopeita, mutta heikompia kuin Koopat.

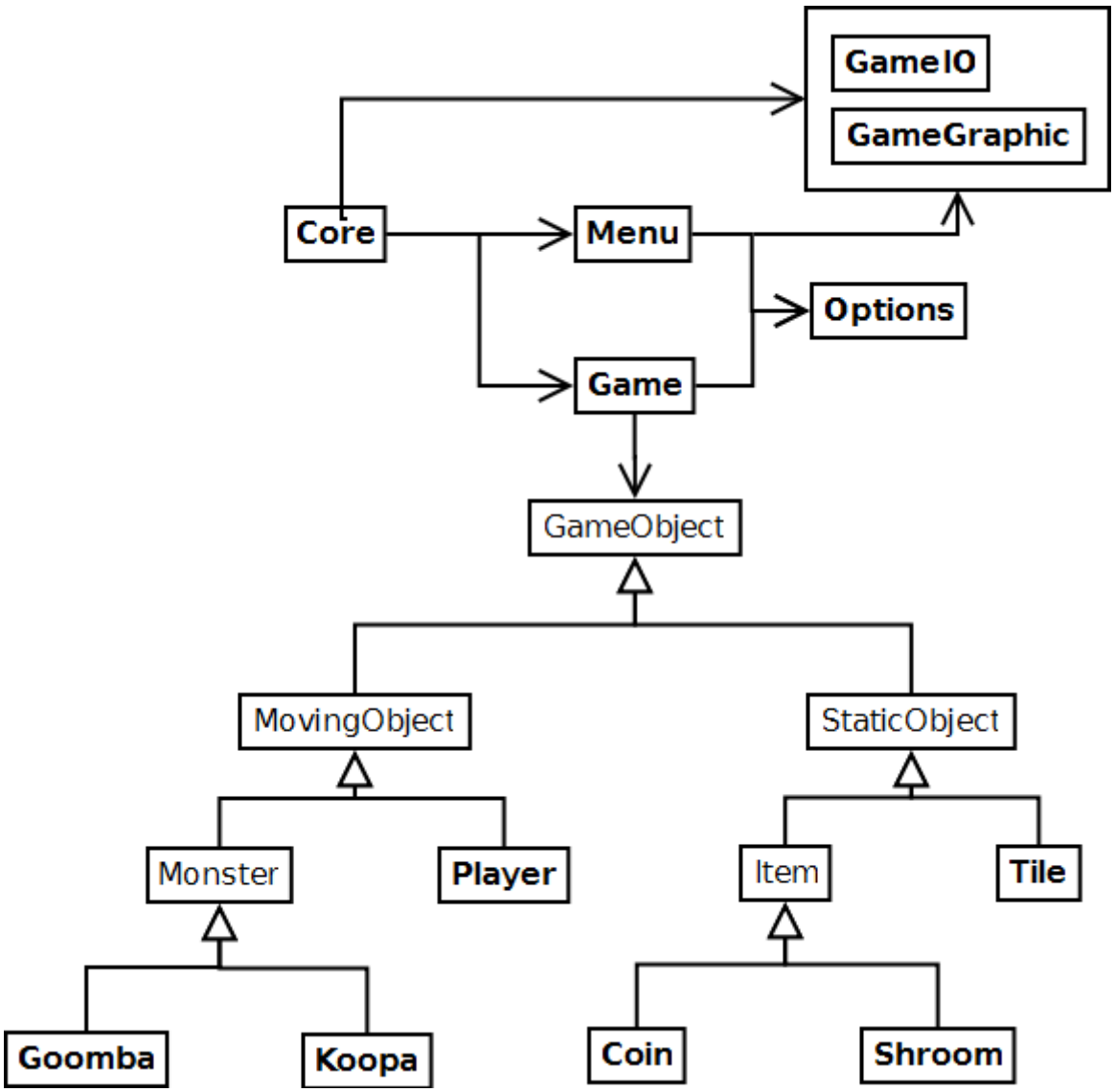
Kaikki hirviöt voi tappaa joko hyppäämällä niiden päälle tai osumalla tiiliskiveen niiden alapuolelta, jolloin tiilen liike tappaa hirviön. Kuolleen hirviön paikalle jää kultainen kolikko, joka kerätään kulkemalla sen päälle. Jos hirviön tappaa hyppäämällä itse sen päälle, on 50% mahdollisuus että kentälle generoituu satunnaiseen paikkaan supersieni, jolla pelaaja kasvaa Super Fabioksi. Super Fabio kestää yhden törmäykseen hirviöön kuolematta. Normaalisti törmätessään hirviöön pelaaja kuolee välittömästi.



Kuva 3. Pelitilanne

Ohjelman arkkitehtuuri

Ohjelma-arkkitehtuuri on lähes täysin projektisuunnitelman mukainen. Erona suunnitelman luokkajakoon verrattuna on liikkumattomille olioille toteutettu yläluokka StaticObject, jonka toteutusta harkitsimme jo projektisuunnitelmassa . Lopulliset luokkien väliset riippuvuussuhteet on esitetty alla olevassa kuvassa 4.



Kuva 4. Luokkarakenne

Ohjelma käynnistyy Core-luokasta, joka alustaa Menu- ja Game-luokat sekä kaikki pelissä tarvittavat grafiikka- ja ääniobjektit GameGraphic- ja GameIO-luokkiin. Ohjelman aloitusvalikko on sijoitettu luokkaan Menu, josta voidaan aloittaa uusi peli, jatkaa olemassa olevaa peliä, lopettaa ohjelma tai säätää pelin asetuksia. Kaikki asetukset talletetaan Options-luokkaan.

Itse pelilogiikka on luokassa Game. Core-luokka antaa Gamelle alustuksessa referenssin pelissä käytettäviin Options-, GameGraphic- ja GameIO-objekteihin. Game-objektin konstruktorissa alustetaan kaikki kyseenomaisella pelikerralla käytettävät pelaajat, hirviöt, kolikot ja muut kenttään sijoitetut esineet. Varatut resurssit vapautetaan destruktorissa (RAII).

Ohjelma-arkkitetuuri hyödyntää tehokkaasti perintää: Kaikilla kentässä esiintyvillä esineillä on abstrakti yläluokka GameObject, jonka perivät MovingObject- ja StaticObject-luokat. Nämä abstraktit yläluokat toimivat nimensä mukaisesti rajapintoina liikkuville ja stationaarisille

objekteille. Kentässä esiintyvät hirviöt (luokka Monster) ja pelaajat (luokka Player), perivät MovingObject-luokan, ja esineet (luokka Item) sekä kentän palat (luokka Tile) perivät StaticObject-luokan. Näistä luokat Monster sekä Item ovat edelleen abstrakteja, s.e. eri hirviötyypit Goomba ja Koopa sekä eri esineet Coin ja Shroom perivät nämä.

Perintä oli luonnollinen vaihtoehto peliin, jossa on paljon melko samanlaisia, mutta kuitenkin ominaisuuksiltaan eroavia objekteja. Kaikki pelin objektit saavat lähes kaikki toiminnallisuutensa periytyvästi, ja täten pystyimme välttämään koodin toistoa. Suunnitteludokumentista saamamme palautteen perusteella päätimme lisätä jo aiemmin harkitsemamme MovingObject- ja StaticObject-luokat GameObjectin alapuolelle, sillä se lisää ohjelman modulaarisuutta. Perinnän ansiosta monet luokat ovat hyvin kompakteja.

Toteutimme mahdollisimman monet luokat otsikkotiedostoiksi, vähentäen tiedostojen kokonaismäärää. Etenkin kaikki abstraktit luokat ovat pelkkiä otsikkotiedostoja. Luokat, joista tehdään useita instansseja, vaativat kuitenkin myös .cc-tiedoston, joten näille luokille on luotu kummatkin tiedostotyypit.

Tietorakenteet ja algoritmit

Ohjelman luonteesta johtuen, emme nähneet tarvetta monimutkaisten tietorakenteiden käyttämiselle toteutuksessa. Käytännössä kaikki tieto säilytetään yksinkertaisina luokkamuuttujina/referensseinä tai std::vector-tietorakenteessa.

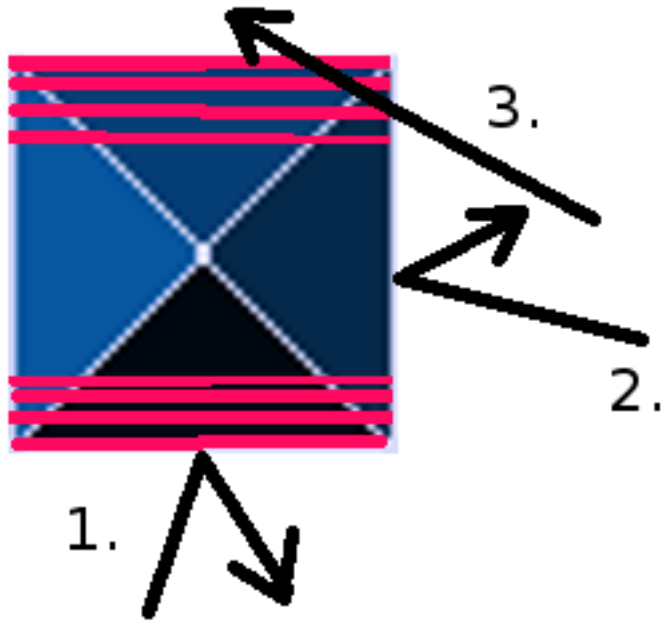
Viholliset säilytetään pointtereina std::vector<Monster*> tietorakenteessa. Vihollisille varataan muistia ja ne alustetaan Game-luokan konstruktorissa ja varatut resurssit vapautetaan vastaavasti luokan destruktorissa (RAII).

Törmäysten käsittely on tehty seuraavasti:

Jokaiselle viholliselle on luotu kaksi CollisionOutline:a: toinen sitä aluetta varten, johon törmäys aiheuttaa pelaajan kuoleman ja toinen, johon törmäys taas tappaa vihollisen. Vihollisen tappava CollisionOutline on luonnollisesti sijoitettu vihollisen yläosaan, jolloin pelaaja siis voi tappaa vihollisen hyppäämällä sen päälle. Muut osat vihollisista ovat sen sijaan pelaajan tappavan CollisionOutlinen peittämiä. Päädyimme edellä mainittuun ratkaisuun törmäyskäsittelyn helpottamiseksi: nyt törmäyksen suuntaa ei tarvitse erikseen huomioida, vaan osuma tiettyyn CollisionOutlineen kertoo suoraan törmäyksen tuloksen.

Törmäykset Tile-palikoihin on ohjelmoitu kiinteästi koordinaattien perusteella ohjelmakoodiin, sillä toisin kuin törmättäessä kolikoihin tai vihollisiin myös törmäyksen suunta on oltava tiedossa. Koska

Tile-palat ovat suorakaiteen muotoisia, oli törmäyslaskenta helpointa suorittaa toteuttamalla yksinkertaiset täysin kimmoisat törmäykset ohjelmakoodiin. Toisin sanottuna pelaaja kimpoaa seinästä samassa kulmassa kuin törmääkin siihen. Poikkeuksena sääntöön koodiin on myös lisätty pieni offset-arvo, jonka tarkoituksena on pehmentää pelaajan törmäyksiä Tile-palojen kulmiin. Tätä on havainnollistettu oheisessa kuvassa 5: Kun pelaaja törmään palaan alhaalta tai sivusta (tapaukset 1 ja 2), on törmäys kimmoisa, mutta pelaajan törmätessä sivusuunnasta aivan Tile-palan kulmaan (tapaus 3) pääsee pelaaja kiipeämään palan päälle (tai liukuu sen alle) sen sijaan että kimpoaisi poispäin palasta. Tämä helpottaa tasoilta toisille liikkumista huomattavasti ja parantaa pelattavuutta.



Kuva 5. Tile-palojen törmäyslogiikka

Myös Tile-palikoilla on kuitenkin lisäksi olemassa myös CollisionOutlinet, joiden avulla havaitaan tilanteet, joissa pelaaja tappaa vihollisen hyppäämällä alakautta Tileen, jonka päällä vihollinen kävelee. Lisäksi samaa CollisionOutlinea käytetään myös havaitsemaan tilanteet, joissa Tile-palikka kuuluu animoida liikkumaan ylöspäin alhaalta tulleen kosketuksen seurauksena. Koska itse pelaajien ja vihollisten kimpoamiset Tile-paloista on toteutettu aiemmin mainitulla tavalla ilman CollisionOutlineja, on Tile-palojen CollisionOutlinet määritelty hieman palojen yläreunaa (kävelytasoa) alemmas. Lisäksi vihollisten CollisionOutline on määritelty jatkumaan hiukan palojen sisälle, kun taas pelaajien Outlinet päättyvät kävelytasoon. Näin toteutettuna törmäys tarkastelu on helppo: vihollinen kuolee, mikäli sekä vihollinen että pelaaja koskettavat samaan Tileen, sillä CollisionOutlinet voivat kohdata vain jos pelaaja hyppää ja osuu alapuolelta Tileen, jonka päällä vihollinen kävelee. Ratkaisun haittapuolena on se, että vihollisen tippuessa pelaajan päälle, pelaaja

kuolee hiukan ennen kosketusta. Koska vihollisten Outlinejen ylimenevät osat ovat kuitenkin erittäin pieniä, ei tätä käytännössä pysty havaitsemaan eikä se siten huononna pelattavuutta. Ratkaisu kuitenkin helpotti toteutusta huomattavasti ja katsoimme sen siitä syystä järkeväksi.

Pelaajien ja vihollisten liikuttamiseen käytetään samoja funktiota, sillä molemmat luokat perivät MovingObject-luokan ja niiden liikuttaminen voi siten tapahtua samalla tavalla. Myös törmäysten tarkastelussa käytetään suurelta osin samoja funktiota. Poikkeuksena ovat törmäykset kolikoihin ja sieniin, jotka ovat ainoastaan pelaajille ominaisia asioita. Perinnän hyödyntäminen liikkeiden ja törmäysten laskennassa säästi meiltä huomattavasti vaivaa toteutusvaiheessa ja suunnittelun voidaan katsoa näiltä osin onnistuneen hyvin.

Bugit ja jatkokehitysehdotukset

Bugit:

Ohjelma segfaulttaa suljettaessa, mikäli näppäimiä on käyty vaihtamassa Options-valikossa. Tämä johtuu ilmeisesti siitä, että ClanLib:n signaalislottia (Slot-objektia) ei tuhota oikea-aikaisesti suhteessa siihen liittyviin CL_DisplayWindow- (peli-ikkuna) ja CL_InputDevice-objekteihin (näppäimistö ja hiiri). Objektia ei kuitenkaan voi tuhota manuaalisesti, joten emme löytäneet keinoa ongelman korjaamiseksi.

Toinen bugi on liittyy Tile-palikoiden reuniin. Joskus kun pelaaja hyppää tilen päätyyn, saattaa pelaaja yhtäkkisesti ilmestyä tilejen ylä- tai alapuolelle. Tämä johtuu tilejen kulmien pehmennyksestä (kts. luku Tietorakenteet ja Algoritmit, kuva 5) ja pelaajan koordinaateista, jotka lasketaan vasemmasta yläkulmasta. Vaikka laskenta tässä toimii halutusti, se aiheuttaa graafisesti epäintuitiivista käyttäytymistä. Vaikka tehty kulmien pehmennys parantaakin pelattavuutta, koodi vaatisi tässä vielä pientä hienosäätöä.

Kolmas bugi on, jos pelaaja seisoo puoliksi putken reunan päällä, pystyy pelaaja tappamaan hirviöitä virheellisesti. Tämä johtuu tarkistuslauseesta joka katsoo, onko hirviöt putkessa, jolloin niitä ei voi tappaa. Koska pelaajan ja hirviöiden koordinaatit määräytyvät spriten vasemmasta yläkulmasta, voi päästä paikkaan, jossa ehto mahdollistaa hirviöiden tappamisen, mutta ei toisinpäin. Tämän voisi korjata tekemällä putkille CollisionOutlinet, mutta emme ajan puutteen takia tahtoneet alkaa tekemään putkille uutta StaticObjectin perivää luokkaa, koska bugi on verrattain pieni.

Jatkokehitettävää:

- Pelaajien törmäykset toisiinsa, jolloin toisen pelaajan voi esim. tönäistä vihollista päin
- Peliin voisi helposti luoda uusia vihollisia tekemällä uusia Monster-olioita
- Uusien kiinteiden objektien luominen onnistuu helposti StaticObject-luokkaa periyttämällä

Työnjako ja aikataulu

Jäimme heti kärkeen tyylikkäästi jälkeen suunnitellusta aikataulusta, kuten tällaisissa projekteissa on tapana. Suunnitelmissa oli aloittaa täysverinen koodaus viikolla 45, mutta muiden kiireiden takia aloittaminen siirtyi viikolle 46. Viikolla 47 aloimme todenteolla vääntää koodia yötä päivää. Viikolla 48 työn hedelmätkin alkoivat näkyä, kun saatiin koko ohjelman kaikki luokat nivottua yhteen pelaajat juoksemaan näytölle. Viikko 49 kului ominaisuuksia lisätessä ja pikku bugeja korjaillessa. Dokumentti kirjoitettiin viikolla 50.

Työtä tehtiin pääosin yhdessä Maarintalolla. Lähtökohtana oli, että vähintään kaksi kolmesta ryhmän jäsenestä oli aina paikalla ohjelmoimassa. Jos joku ryhmästä oli poissa, häntä informoitiin työn edistymisestä sähköpostitse ja Facebookissa. Muuten kommunikointi hoidettiin kasvotusten.

Toteutus meni pääosin suunnitellusti, vaikkakin ratkaisimme ja osittain myös ohjelmoimme ongelmakohtia ”ristiin”. Työjako oli pääpiirteissään seuraava:

Juho toteutti ohjelman alkuvalikot ja koko pelin kasassa pitävän Mario-luokan.

Riku toteutti Game-luokkaan sijoitetun pelilogiikan.

Matti toteutti GameObject luokan, sekä sen perivät alaluokat sekä piirsi pelin grafiikat.

Arvioitu ajankäyttö:

Projektisuunnitelma: 6h/hlö

Toteutus: yhteensä n. 75h/hlö

Juho: Näppäimistösignaalien käsittely 15h

Valikoiden toteutus 10h

GameGraphic- ja GameIO-luokat 10h

Muut 40h

Riku: Rajapinnan suunnittelu ja toteutus GameObject-luokkaan ja sen aliluokkiin: 20h

Pelaajien ja vihollisten liikkeiden laskeminen: 10h

Tile-törmäykset: 10h

Muut 35h

Matti: CollisionOutline -törmäykset: 20h

Spritet: 5h

Luokkahierarkian ja rajapintojen toteutus Game-luokan käyttöön: 15h

Muut 35h

Loppudokumentti: 6h/hlö

Yllä olevassa Toteutus-osan kohta ”Muut” sisältää mm. setterien ja getterien sekä yleisen luokkarakenteen toteutusta, koodin kommentointia Doxygen-syntaksin mukaiseksi, testausta ja debuggausta sekä eri luokkien yhteensovittamista.

Testaus ja kehitys

Pyrimme testaamaan ohjelmaa vaiheittain sitä mukaa kun osia siitä valmistui. Koska alkuvalikko oli oma selkeä kokonaisuutensa, pystyi sitä testaamaan helposti ilman muita ohjelman osia. Game-luokan sekä sen käyttämien GameObject-luokan sekä sen aliluokkien testaaminen oli sen sijaan huomattavasti vaikeampaa, sillä molemmat luokat riippuvat toisistaan ja mahdollisissa vikatilanteissa vian alkuperän löytäminen ei aina ollut helppoa. Vaikka toteutimmekin sekä Game-luokkaa että sen käyttämiä luokkia pienissä osissa ja rinnakkain, oli kehitystyö erityisesti alussa haastavaa, sillä sekä Game-luokan että sen käyttämien luokkien piti olla jo melko pitkälle valmiita ennen kuin mitään konkreettista ja näkyvää pelitilannetta päästiin edes piirtämään näytölle. Alussa testaus perustuikin lähinnä erilaisiin Debug-printteihin, joilla rajapintojen läpi siirrettävien tietojen oikeellisuus voitiin varmistaa.

Kehitystyön haastavimmaksi osuudeksi osoittautui signaalien käyttö ja törmäyslogiikan suunnittelu ja toteuttaminen. Edellä mainittu Game-luokan riippuvuus GameObject-luokasta ja sen aliluokista vaikeutti myös kehitystä huomattavasti. Lukemalla ristiin toistemme tekemiä koodeja ja suunnittelemalla rajapinnat yhdessä saimme kuitenkin ongelmat ratkaistua.

Valgrind ei osoittautunut testauksessa kovinkaan hyödylliseksi, sillä ClanLib:stä aiheutuu suuri määrä virheitä ja omien virheiden löytäminen niiden joukosta oli lähes mahdotonta.

Erot projektisuunnitelmaan

Suunnitelma käsitteli suurelta osin vain ohjelman arkkitehtuuria ja luokkajakoa. Kuten luvussa Ohjelman arkkitehtuuri todettiin, lopullinen luokkajako on lähes täysin projektisuunnitelman mukainen. Suunnitelmassa ajattelimme että GameGraphic-luokasta luodaan kaksi instanssia, mutta käytännössä havaitsimme yhden laajemman structin toteuttamisen helpommaksi. Ajankäyttö poikkesi hieman suunnitellusta kuten aikaisemmin todettiin, mutta muuten voidaan todeta suunnitellun onnistuneen hyvin.

Lähteet

Käytimme apuna seuraavia lähteitä:

ClanLibin oma dokumentaatio:

http://clanlib.org/wiki/Documentation

Doxygenin dokumentaatio:

http://www.stack.nl/~dimitri/doxygen/manual.html

Muuta:

http://www.cplusplus.com/

luentokalvot

#tkk.as.c @ IRCNet

Liitteet

Doxygenillä luotu referenssidokumentaatio refman.pdf (löytyy svn:stä kansiosta /doc)