TAMPEREEN
AMMATTIKORKEAKOULU

TUTKINTOTYÖRAPORTTI

MOBIILIPELIOHJELMOINTI
Case: CyberPong

Tuukka Peltoniemi

Tietojenkäsittelyn koulutusohjelma
marraskuu 2005
Työn ohjaaja: Paula Hietala

TAMPERE 2005

TAMPEREEN
AMMATTIKORKEAKOULU

Tekijä(t): Tuukka Peltoniemi

Koulutusohjelma(t): Tietojenkäsittely

Tutkintotyön nimi: Mobiilipeliohjelmointi. Case: CyberPong

Title in English: Mobile game programming. Case: CyberPong

Työn valmistumis-kuukausi ja -vuosi: Marraskuu 2005

Työn ohjaaja: Paula Hietala **Sivumäärä:** 59

TIIVISTELMÄ

Tietokone- ja mobiilipelien suosio on kasvanut tasaisesti vuodesta toiseen. Tietokonepelien toteuttamista tutkittaessa on huomattavissa, että kaupallisen tietokonepelin toteuttaminen vaatii yleensä suuren kehittäjäryhmän. Kuitenkin mobiilipelejä on vielä mahdollista toteuttaa pienenkin ryhmän voimin.

Mobiili- ja tietokonepelien ohjelmointi eroaa joiltakin osin perinteisestä sovellusohjelmoinnista. Tämän työn yhtenä tarkoituksena on pyrkiä selvittämään lukijalle näitä eroavaisuuksia. Työssä lähdetään liikkeelle selvitellen yleisiä peli- ja sovellusohjelmoinnin eroja. Työn edetessä siirrytään peliohjelmoinnista käsittelemään mobiilipeliohjelmointia.

Toteutin tätä työtä varten CyberPong-mobiilipelin, jonka avulla kuvasin työssä erilaisia mobiilipelin toteuttamisen ongelmakohtia. Suurin osa työn teoreettisesta pohjasta perustuu CyberPongin toteuttamisesta esiin tulleisiin asioihin, mutta lisäksi työssä on käytetty pohjana myös aikaisempaa kokemustani mobiilipelien ohjelmoinnista.

Tämän työn tarkoituksena on tarjota yleiskuva peliohjelmoinnista siitä kiinnostuneille sekä palvella opaskirjana peliohjelmointia aloittaville.

.

Avainsanat: J2ME Java Mobiilipelit Peliohjelmointi Tietokonepelit

TAMPERE
POLYTECHNIC

Author(s): Tuukka Peltoniemi

Degree Programme(s): Business Information Systems

Title Mobile game programming. Case: CyberPong

Month and year November 2005

Supervisor Paula Hietala **Pages:** 59

ABSTRACT

Computer games and mobile games have increased their popularity from year to year. When production of commercial computer games is studied, it can easily be noticed that production of a single game requires an effort from a large team of professionals. At the moment mobile games are the only form of games that can still be produced with a smaller team.

Game programming, including mobile game and computer game programming, differs on some parts from a traditional application programming. One of this work's main purpose is to explain these differences. This thesis starts with explanations of differences between computer game programming and application programming. Then thesis continues to handle the development of mobile games.

I produced a mobile game to be used as a reference in this thesis. The game is named CyberPong and it's main purpose is to show problematic areas within mobile game development. Theory in this thesis is mostly based on things observed during production of the CyberPong. Theory is also based on a earlier knowledge of game programming.

The main purpose of this thesis is to offer a general view about game programming and to serve as a text-book to those who are just starting to learn game programming.

.

Keywords J2ME Java Mobilegames Game programming Computer games

Sisällysluettelo

1 Johdanto

Tietokonepeliala on varsin nuori sekä nopeasti kehittynyt. Alan nopea kehittyminen on vaatinut niin laitevalmistajia, ohjelmoijia kuin graafikoitakin omaksumaan nopeasti muuttuvia käytäntöjä ja työtapoja. Tietokonepelialalle on kuitenkin kehitetty erilaisia standardeja, jotka helpottavat pelien kehittämistä. Tämän vuoksi tietokonepelien kehittäminen on tullut mahdolliseksi myös yksityisille tahoille. Kuitenkin kaupallisen pelin toteuttaminen vaatii tällä hetkellä välttämättä ison kehittäjäryhmän taustalleen sekä yleensä ison julkaisijan tuen pelille.

Mobiilipeliala (mobiilipelillä tarkoitetaan kannettavalla pienikokoisilla laitteilla toimivaa peliä) on vielä huomattavasti tietokonepelialaa nuorempi ala. Alalla ollaan vasta pikkuhiljaa pääsemässä laitteistojen sekä standardien osalta sellaiselle tasolle, että kunnollisten pelien kehittäminen mobiililaitteille on järkevää ja yleensäkin mahdollista. Mobiilipelin toteuttaminen jollekin tietylle matkapuhelinmallille on ollut pitkään mahdollista, mutta saman pelin toteuttaminen useammalle puhelinmallille on ollut vaikeata. Erilaisten mobiililaitteiden ja näiden keskinäisten ristiintoimivuusongelmien vuoksi mobiilipelialalle on kehittynyt kaksi uutta toimialaa. Ensimmäinen toimiala keskittyy pelkästään valmiiden mobiilipelien sovittamiseen muihin mobiililaitteisiin sopiviksi (engl. to port, suom. porttaaminen), toinen toimiala keskittyy pelkästään testaamaan valmiita mobiilipelejä ja niiden eri laiteversioita. Pelijulkaisijoiden on käytännössä pakko käyttää molempien alojen palveluita ennen pelin varsinaista julkaisua.

Tavoitteeni on tutkintotyössä antaa yleiskuva pelialasta (peliala käsittää niin tietokone kuin mobiilipelitkin) ja sen toiminnasta. Peliohjelmointi eroaa jonkin verran perinteisestä sovellusohjelmoinnista ja tästä syystä tutkintotyöni päätarkoitus on käsitellä mahdollisimman kattavasti pelialan eroavaisuudet sovellusohjelmointiin verrattuna ohjelmoijan näkökulmasta. Käsittelen tutkintotyössäni nämä eroavaisuudet mahdollisimman havainnollisesti.

Toteutan lisäksi tutkintotyön puitteissa mobiilipelin, jonka avulla pyrin havainnollistamaan erilaisia hankalia kohtia mobiilipelin toteuttamisessa. Käsittelen tutkintotyön loppupuolella mobiilipelin toteuttamisessa esiin tulleet ongelmakohdat mahdollisimman kattavasti.

2 Tutkintotyön tavoite, rakenne ja rajaus

Ohjelmointi on kiinnostanut minua alusta asti suurimmaksi osaksi peliohjelmoinnin takia. Tästä syystä päätin jo opintojeni alkuaikoina, että tutkintotyöni tulisi käsittelemään ainakin jollakin tasolla peliohjelmointia. Lopulliseen tutkintotyön aiheeseen on vaikuttanut työpaikkani Universomossa sekä oma kiinnostukseni mobiilialaa kohtaan. Universomo on mobiilipelejä kehittävä noin 20 henkeä työllistävä yritys.

Peliohjelmointia käsittelevissä kirjoissa ja oppaissa mennään yleensä liian syvälle koodiin varsinaisesti selittämättä asioiden syitä, seurauksia tai merkityksiä. Ohjelmoinnissa, kuten monella muullakin elämän alueella on mielestäni tärkeätä ymmärtää asioiden syyt sekä seuraukset, jotta näihin asioihin saa vahvemman ja syvemmän ymmärryksen. Pyrinkin tällä tutkintotyöllä kattamaan peliohjelmoinnin alueelta omasta mielestäni tärkeimmät asiat siten, että niistä olisi mahdollista saada selkeä käsitys myös vähäisemmällä koodin tutkimisella.

Tutkintotyön tehtävä

Olen tutkinut peliohjelmointia laaja-alaisesti tätä tutkintotyötä varten. Toteutin tutkintotyötä varten myös mobiilipelin aina suunnitteluvaiheesta valmiiseen tuotteeseen asti. Pyrin löytämään pelin toteuttamisen jokaisessa vaiheessa asioita, jotka olivat itselleni ongelmallisia tai joihin itse koin tarvitsevani syvemmän ymmärryksen. Käsittelen nämä asiat tässä tutkintotyössä mahdollisimman perusteellisesti. Osaa asioista en itse kohdannut peliä toteuttaessa, mutta näiden asioiden tärkeys on tullut muista yhteyksistä selväksi, ja ne ovat tästä syystä saaneet omat kappaleensa tutkintotyöhön.

Pyrin muodostamaan tutkintotyön lukijalle (sinulle) kuvan siitä, millaisia asioita on mahdollista kohdata toteutettaessa mobiilipeliä tai tietokonepeliä. Pyrin kuitenkin pitämään pääpainon mobiilipeleissä. Toivonkin, että tämä tutkintotyö palvelisi peliohjelmoinnista kiinnostuneiden käytännöllisenä opaskirjana.

Pyrin selittämään myös peliohjelmointiin ja mobiilipeliohjelmointiin liittyvät termit mahdollisimman tarkasti ja kuvaavasti. Tämän tutkintotyön lopusta löytyy merkittävät käsitteet ja termit sisältävä sanasto. Sanastosta löytyvät sanat on kursivoitu tekstissä. Toivonkin, että aloittelija löytää tutkintotyötä lukemalla keinot välttää ainakin joitakin peliohjelmoinnin ”sudenkuoppia”.

Aiheen rajaus

Tutkintotyön aihe on rajattu siten, että tutkintotyössä tullaan käsittelemään mobiilipelien toteuttamista mahdollisimman laajasti ohjelmoijan näkökulmasta. Peliohjelmoinnin alueella on paljon erilaisia käytäntöjä, jotka pätevät myös mobiilipelipuolella. Nämä asiat käsitelläänkin tutkin-

totyön luvussa 4. Toteutan tutkintotyön puitteissa mobiilipelin, jota käytän mahdollisimman paljon pohjana tutkintotyön kirjallisessa osuudessa.

Sivuutan kuitenkin lähdekoodin optimoinnin kokonaan (koodista pyritään tekemään mahdollisimman tehokasta, jotta sovellusta voidaan käyttää myös hitaissa laitteissa), vaikka se on tärkeä osa peliohjelmointia. Optimoinnin jättäminen pois tutkintotyöstä johtuu siitä, että aihe on liian laaja käsiteltäväksi vain osana tutkintotyötä ja aiheen supistaminen ja vain osittainen käsitteleminen ei ole järkevää. Optimoinnista on useita hyviä oppaita sekä artikkeleita. Esimerkiksi Java ohjelmien optimointia käsittelevä artikkeli (http://websphere.sys-con.com/read/45001.htm) on melko lyhyt, vaikka hyvä artikkeli asiasta (Storey 2004).

Käsittelen pelien graafisuutta sekä graafisuuden suunnittelua mahdollisimman vähän, koska nämä asiat ovat myös liian laajoja käsiteltäväksi vain osana tätä tutkintotyötä. Kimmo Koivumäen Tampereen Ammattikorkeakoululle tekemä tutkintotyö ”Mobiilipeligrafiikka Series 60 -alustan matkapuhelimiin” on kattava kokonaisuus mobiilipelien grafiikoista (Koivumäki 2005).

Tutkintotyön sisältö

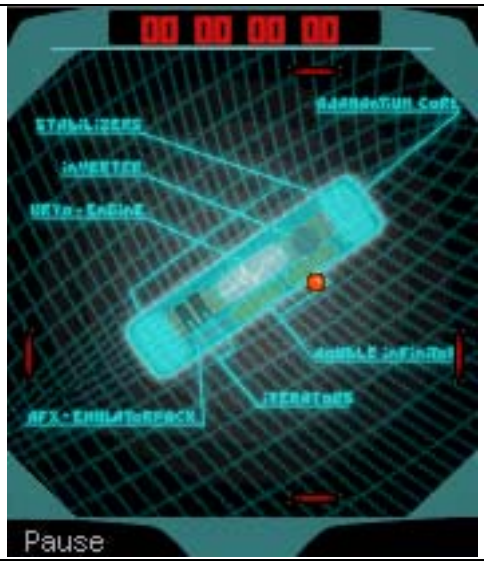
Peliohjelmoinnin kunnolliselle ymmärtämiselle on tärkeätä ymmärtää pelien historiaa pääpiirteissään. Seuraavassa pääluvussa käsitelläänkin lyhyesti tietokone-, konsoli- sekä mobiilipelien historiaa. Näkökulma pelien historiaan on markkinoinnillinen.

Luvussa 4 käsitellään asioita, jotka ovat yhteisiä niin tietokone-, konsolikuin mobiilipelienohjelmonnissakin. Tämän jälkeisessä luvussa käsitellään yksinomaan mobiilipeleille ominaisia asioita ohjelmoijan näkökulmasta. Kummassakin luvussa asiat käsitellään mahdollisimman pitkälti kronologisesti pelinkehitysprosessin mukaisesti.

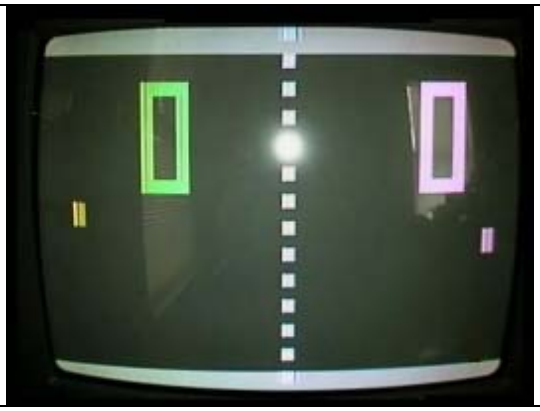
Luvussa 6 käsitellään mobiilipelin toteuttamisessa esiin tulleita ongelmia sekä sitä, että miten itse ratkaisin nämä ongelmat. Tämä kappale vaatii lukijalta eniten ohjelmoinnin ymmärtämistä. Pyrin kuitenkin selittämään asioita mahdollisimman selkeästi sen sijaan, että näyttäisin asioista vain koodiesimerkkejä.

Toteutettava mobiilipeli

Tämän tutkintotyön puitteissa toteuttava mobiilipeli on kehitetty yksinomaan tutkintotyötä varten. Kimmo Koivumäki toteutti peliin grafiikat, sekä osallistui peli-idean kehittämiseen. Peli kulkee tällä hetkellä työnimellä ”CyberPong” (Kuvio 1). Peli-idea yksinkertaistetusti on vanhan *PONG*:in neljän pelaajan versio aseiden kanssa. *PONG* on yksinkertainen kahdenpelaajan peli, jossa pelaajat yrittävät pitää pallon mahdollisimman pitkään ruudulla liikuttamalla omalla puolellaan olevaa mailaa (Kuvio 2).



Kuvio 1 Kuvakaappaus CyberPong:in työversiosta



Kuvio 2 *Alkuperäinen Atarin julkaisema PONG (PONG-Story)*

3 Katsaus kirjoittajan ja pelialan historiaan

3.1 Tietokonepelit: lyhyt oppimäärä

Peliteollisuuden kehittyminen

Tietokonepelit ovat varsin nuori ala, vaikka lähestulkoon 20 vuotta vanhempi ala kuin mobiilipelit. Ala on kehittynyt nykyiseen muotoonsa laskutavasta riippuen noin 30 vuodessa. Peliala sai alkunsa yksittäisten ohjelmoijien yksittäisistä ideoista. Vielä 80-luvun puolivälin tienoilla oli erittäin vähän henkilöitä, joilla oli tietokonepelien toteuttamiseen tarvittava laitteisto ja tietotaito. Ohjelmoijia olikin 80-luvulla huomattavasti vähemmän kuin nykypäivänä (Taulukko 1). Lisäksi ohjelmointi oli vaativampaa sekä hidastempoisempaa. Tässä vaiheessa tekniikka ei ollut vielä valmis pelialaa varten. Hiljalleen tekniikka lähti kehittymään pelialaa tukevaan suuntaan ensimmäisten kotitietokoneiden yleistymisen myötä.

Taulukko 1 Ohjelmoijien määrä Suomessa eri vuosikymmeninä (LABORSTA Internet)

Vuosikymmen	***Ohjelmoijia***
1970	2877
1980	7876
1990	28500

Taulukosta 1 on nähtävissä ohjelmoijien määrän kasvaminen vuosikymmenten kuluessa. Taulukossa esitetyt luvut muodostuvat vain ohjelmoijista, jotka ovat tehneet ohjelmointia työkseen. Uskonkin, että vuosikymmenten välillä olevat erot olisivat vielä suuremmat, jos mukaan laskettaisiin harrastuksenaan ohjelmoivat henkilöt.

Peliala on kokonaisuudessaan ollut viimeisten viidentoista vuoden ajan voimakkaassa kasvussa. Tämä näkyy suoraan pelien erittäin suurina budjetteina, sekä pelien yleisesti korkeana laatuna. Tämä ei kuitenkaan tarkoita sitä, että kaikki pelit olisivat omasta mielestäni hyviä, tai että suuri budjetti takaisi hyvän tuoton pelille. Pelien budjetteihin vaikuttavat pelien jatkuvasti kohoavat tuotantokustannukset, sekä jatkuvasti kasvavat markkinointikustannukset.

Aloittelevan yrityksen onkin vaikea päästä tietokone- ja konsolipelimarkkinoille kasvaneiden kehityskustannusten takia. Hyvänä esimerkkinä kasvaneista henkilöstötarpeista on vuonna 2005 ilmestynyt konsolipeli Gran Turismo 4. Gran Turismo on Sony Playstationeille ilmestyvä autopelisarja, josta tämä neljäs osa on uusin. Gran Turismo 4:ssä on eri automalleja yli 700 kappaletta (Heinonen 2005). Konsolipelien

erittäin kauniit ja fotorealistiset grafiikat nostavat tuotantokustannuksia vielä entisestään. Pelkästään autojen 3D-mallintamiseen kuluneet henkilötyötunnit ovat tähtitieteelliset. Peliä on kehitetty neljä vuotta ja pelin tekemisessä on ollut mukana 20 ohjelmoijaa, 40 graafikkoa sekä 20 avustajaa (Fitzgerald 2005). Henkilötyötunnit jotka kuluvat pelkästään autojen näkyvän osan mallintamiseen ovat suuret. Tämä on kuitenkin vasta pieni osa siitä, mitä muuta peliin pitää tehdä ennen kuin se on valmis markkinoille.

Pelien kehittyminen

Pelit ovat kehittyneet voimakkaasti koko niiden olemassaolon ajan. Pelien kehittyminen on pakottanut myös tekniikkaa kehittymään samassa tahdissa pelien kanssa, ja toisaalta tekniikan kehittyminen on taas pakottanut pelejä hyödyntämään uusinta tekniikkaa. Lopputuloksena myös pelaajan on pysyttävä tekniikan mukana ja ostettava uusinta tai vähintäänkin uudehkoa tekniikkaa voidakseen pelata pelejä, joita on löydettävissä kauppojen hyllyiltä.

Pelien alkutaipaleilla keskivertopelissä oli yleensä vain yksi toiminnallisuus, jonka ympärille oli lisätty mahdollisesti yksinkertainen alkuvalikko, josta pääsi siirtymään varsinaiseen peliin ja pelistä pois. Juonellisuus tuli peleihin mukaan 80-luvun loppupuolella. Nykyään uusimmissa peleissä pelin juoni on yleensä käsikirjoittajan kirjoittama eikä vain ohjelmoijan ohjelmoinnin lomassa sanailema. Muutenkin kuin vain juonen osalta pelien sisältö on monipuolistunut vuosien saatossa. Nykypeleissä voidaan nähdä trendinä elävän, realistisen ja vapaan maailman mallintaminen. Pelaajalle pyritään antamaan kuva vapaasta maailmasta, jossa pelaaja voi toimia mielensä mukaisesti. Tämä asettaa haasteita niin pelienkehittäjille, kuin laitteistovalmistajillekin.

Pelaajien odotusten muuttuminen

Tietokoneiden ja pelikoneiden nopea kehittyminen viimeisen kahden vuosikymmenen aikana on kaikessa hiljaisuudessa totuttanut kaiken ikäisiä pelaajia jatkuvaan, ja vieläpä nopeaan pelien paranemiseen. Kahdeksankymmentäluvun loppupuolen pelit ovat eri tasolla nykypeleihin verrattuna, varsinkin latausaikojen, grafiikoiden ja sisällön osalta.

Tekniikan nopea kehittyminen on poistanut käytännössä kaikki tietokonepeleissä esiintyneet latausajat. Vielä 80-luvun loppupuolella oli aivan tavallista kahden minuutin latausajat peliä käynnistettäessä. Tällaiset latausajat olivat tosin laitekohtaisia.

3.2 Mobiilipelit

Käytän tutkintotyössä käsitettä mobiilipeli vain matkapuhelinten peleistä. Yleisemmän tulkinnan mukaan kämmentietokoneiden, laskukoneiden ja käsikonsoleiden (Nintendo GameBoy, Sony PSP...) pelit luetaan myös mobiilipeleiksi. Oman tulkintani mukaan näiden laitteiden välillä on niin

suuri ero mm. pelien budjettien suhteen, etten halua laskea näitä kaikkia saman käsitteen alle.

Edelleenkin yksi kaikkien aikojen tunnetuimmista mobiilipeleistä on matopeli. Matopeli on ollut valmiiksi asennettuna ainakin useissa Nokian valmistamissa matkapuhelinmalleissa. Nykypäivän markkinoilla mobiilipelit ovat paljon muutakin kuin vain valmiiksi asennettuja pelejä (tällaisista peleistä käytetään termiä esiasennettu). Jälkikäteen matkapuhelimiin ladattavien pelien markkinat ovat syntyneet viimeisten vuosien aikana.

Tekniikan kehittyminen

Matkapuhelimet kehittyvät tällä hetkellä todella nopeasti, ja useimmat matkapuhelimet ovatkin tehokkaampia kuin useimmat kahdeksankymmentäluvun loppupuolen tietokoneet/pelikoneet. Vaikka matkapuhelimet ovat kehittyneet suuresti, eivät ne siltikään ikävä kyllä sovi vielä kovin hyvin pelejä pyörittäviksi laitteiksi. Tämä johtuu useissa tapauksissa siitä, ettei matkapuhelinta ole suunniteltu pelejä varten, vaan mm. virrankulutus on suuremmassa arvossa kuin pelien toimivuus.

Suurin osa matkapuhelinvalmistajista on kehittänyt erillisen tai erillisiä pelipuhelimia. Esimerkiksi Nokia on julkaissut kaksi pääasiallisesti pelikäyttöön suunniteltua matkapuhelinta. Nokian N-Gage julkaistiin lokakuussa 2003 ja N-Gage QD julkaistiin heinäkuussa 2004. Nämä niin kutsutut pelipuhelimet soveltuvat hyvin pelien pelaamiseen. Lisäksi matkapuhelinvalmistajien kalliimmat puhelinmallit soveltuvat yleensä vähintäänkin hyvin pelien pelaamiseen.

Pelien kehittyminen

Mobiilipelialalla on pelien suhteen useita ongelmia. Yksi suurimmista ongelmista on pakotettu laitetuki. Jos pelinkehittäjä haluaa saada pelinsä myytyä julkaisijoille, on pelin toimittava valtaosassa sillä hetkellä markkinoilla olevista puhelimista. Tämä tarkoittaa sitä, että pelistä on pystyttävä tekemään helposti monia erilaisia versioita toimiviksi niin heikompitasoisissa (engl. low-end), kuin korkeatasoisissakin puhelimissa (engl. high-end). Tämä asettaa väkisinkin vaatimuksia pelien laajuudelle ja yleiselle tasolle.

Mobiilipelejä testaava yritys QA (www.softwareqatest.com) pitää mobiilipeliä testatessaan yli 15 sekunnin latausaikaa pelissä olevana ongelmana (bugina), ja palauttaa pelin takaisin pelinkehittäjälle korjattavaksi. Tämä tarkoittaa sitä, että peli pitää toteuttaa siten, ettei pelissä synny (ainakaan peliä käynnistettäessä) pitkiä latausaikoja. Ikävä kyllä tämä tarkoittaa useissa tapauksissa pelin sisällön karsimista.

3.3 Kirjoittajan historia pelimaailmassa

Olen ollut osa tietokone- ja konsoli-pelialaa jo kaksi vuosikymmentä. Olen viettänyt nämä vuodet kuluttajan sekä innokkaan ja aktiivisen pe-

laajan rooleissa. Nyt viimeisten kahden vuoden aikana kiinnostukseni pelien tuottamiseen sekä toteuttamiseen on lisääntynyt voimakkaasti samalla kun innostukseni pelaamista kohtaan on hitaasti vähentynyt. Oman kokemukseni mukaan pelialaa on melko vaikeata lähestyä muutoin kuin edellä mainitsemieni roolien kautta. Alalla on vain vähän erilaisia työtehtäviä, joista käytännössä jokainen vaatii jonkinlaisen koulutuksen kyseiseen tehtävään sekä lisäksi hyvän yleissivistyksen peleistä ja pelialasta yleensä. Suorittamani opinnot Tampereen Ammattikorkeakoulussa ovat antaneet itselleni tarvittavan tietotaidon ohjelmoinnista, jonka avulla uskon selviytyväni pelialan mukanaan tuomista haasteista toimiessani ohjelmoijana. Tämän tutkintotyön toteuttaminen avasi itselleni hyvän mahdollisuuden päästä tutustumaan pelialaan niin tuottajan kuin toteuttajankin rooleissa.

Työskentelen tutkintotyön kirjoittamishetkellä Tampereella sijaitsevassa mobiilipelejä valmistavassa yrityksessä Universomossa ohjelmoijana. Universomo on melko pieni yritys ja se työllistääkin tällä hetkellä noin kaksikymmentä henkilöä. Olen työskennellyt yrityksessä vasta kahdeksan kuukautta. Kuitenkin jo tänä aikana olen saanut hyvän kuvan pelialalla toimivan yrityksen toiminnasta sekä toimintatavoista. Olen saanut työsuhteestani kokemusta mobiilipelien ohjelmoinnista sekä kokemusta yleisesti peliohjelmoinnista ja sen perusteista.

Ohjelmointi on ollut harrastukseni neljän vuoden ajan. Aloitin ohjelmoinnin opiskelun harrastusmielessä jo hieman ennen pääsyäni opiskelemaan Tampereen Ammattikorkeakouluun tietojenkäsittelyn koulutusohjelmaan. Opiskelin ohjelmointia pääasiassa kahden ohjelmointikirjan avulla, sekä internetistä löytyvien ohjelmointioppaiden avulla. Varsinkin ohjelmoinnin perusteiden ymmärtäminen oli todella vaikeata kirjojen avulla. Toivonkin, että pystyn esittämään asiat lukemieni kirjojen esitystapaa yksinkertaisemmin. Tämän tutkintotyön ymmärtäminen sekä hyödyntäminen vaatiikin uskoakseni saman verran kokemusta ohjelmoinnista, kuin mitä itselläni on, tai vaihtoehtoisesti erittäin suuren kiinnostuksen peliohjelmointia kohtaan.

4 Peliohjelmointi

Tässä luvussa käsitellään yleisesti peliohjelmoinnin eroavaisuuksia tavalliseen sovellusohjelmointiin. Pääpiirteissään kaikki käsiteltävät asiat soveltuvat ainakin periaatteessa käytettäväksi kaikilla ohjelmointikielillä. Alaluvuissa ei ole montaa koodiesimerkkiä ja nämä esimerkit on toteutettu käyttäen Java-ohjelmointikielen kirjoitustyyliä (syntaksia). Jos tarkoituksena ei ole missään tapauksessa ohjelmoida pelejä käyttäen Javaa ohjelmointikielenä, on kannattavaa lukea koodiesimerkit vain pintapuolisesti. Tällaisessa tapauksessa on hyödyllistä pyrkiä ymmärtämään vain esimerkin logiikka ja sivuuttaa varsinainen kirjoitusasuun liittyvät asiat.

Alaluvut eivät ole parhaassa mahdollisessa järjestyksessä pelinkehittämisprosessin kannalta. Alalukujen järjestys määräytyi siten, ettei yksikään alaluku vaadi esitietoa myöhemmin esitettävästä alaluvusta.

4.1 Ohjelmointikielet

Pelejä voidaan ohjelmoida käytännöllisesti katsoen kaikilla olemassa olevilla ohjelmointikielillä. Kuitenkin suurin osa kaupallisista peleistä on toteutettu käyttäen tiettyjä ohjelmointikieliä. Tietokonepelien puolella hallitsevat ohjelmointikielet ovat tällä hetkellä *C++* yhdessä DirectX tai OpenGL rajapinnan kanssa. DirectX ja OpenGL rajapinnat ovat yksinkertaistetusti iso joukko funktioita, joilla saadaan pelejä varten tietokoneista mahdollisimman paljon tehoa. Mobiilipelien puolella hallitsevat ohjelmointikielet käsitellään seuraavassa luvussa.

Juuri ohjelmointia aloittelevan kannattaa tutkia tarkoin ohjelmointikielten tilannetta ennen varsinaisen ohjelmointikielen valintaa. Ohjelmointikielen valinnalla on kauaskantoisia vaikutuksia, joita on vaikea ennakoida alussa. Jos tietokonepelien ohjelmointi on pääprioriteettina ohjelmoinnin opiskelulle, näkisin tällä hetkellä kolme hyvää vaihtoehtoa ohjelmointikieleksi. *C++* on vielä tällä hetkellä paras vaihtoehto yleisyytensä vuoksi. Uskon kuitenkin, että lähitulevaisuudessa *C++* tulee korvautumaan jollakin toisella hieman helpommalla ja oliopohjaisella kielellä. Yhtenä korvaavana vaihtoehtona näen C# kielen, joka oman tulkintani mukaan on ottanut hyvät puolet niin Javasta kuin *C++:kin*. C# kieli on kuitenkin vielä, varsinkin DirectX rajapinnan osalta, osittain keskeneräinen. Kolmantena vaihtoehtona on Java. En kuitenkaan näe Javaa vahvana haastajana kaupallisissa peleissä, mutta Javalla peliohjelmoinnin opiskelu sujuu helpommin kuin muilla ohjelmointikielillä.

4.2 Pelisilmukka

Pelisilmukalla tarkoitetaan pelin uudelleentoistettavien toimintojen yhtä suorituskierrosta. Helpoin tapa ajatella pelisilmukkaa on ajatella jonossa olevia tehtäviä, joita suoritetaan jonon alusta loppuun. Kun tehtäviä on suoritettu siten, että päästään jonon viimeiseen tehtävään aloitetaan jonon suoritus uudelleen jonon alusta. Yleensä tällaisissa jonoissa on tehtävinä ainakin ruudun uudelleenpiirto, syötteenkäsittely sekä jonon viimeisenä tehtävänä pelin tahdistaminen. Näitä kolmea tehtävää käsitellään yksityiskohtaisemmin luvuissa 4.4, 4.7 ja 4.12.

Kun peli käynnistetään ja pelaaja pääsee varsinaiseen pelitilaan, käynnistetään pelisilmukka. Tämä silmukka pyörii pelin käynnistyksestä aina pelin sulkemiseen saakka tai niin kauan kuin varsinainen peli on käynnissä. Yleinen tapa on se, että kun pelaaja menee pelivalikoihin, pelisilmukka pysäytetään. Kun pelaaja tulee takaisin pelitilaan, aloitetaan pelisilmukan suorittaminen uudestaan silmukan alusta. Silmukka pysäytetään siksi, ettei pelin haluta etenevän pelaajan ollessa valikoissa.

Laajemmista ja kaupallisistakin peleistä löytyy vastaava pelisilmukka. Pelin toiminnasta ja suunnittelusta riippuen pelissä saattaa kuitenkin olla useita rinnakkaisia pelisilmukoita. Tällaisissa peleissä saattaa olla oma silmukkansa esimerkiksi syötteenlukemiselle, tietokoneen ohjaamien vihollisten liikuttamiselle ja musiikin soittamiselle.

Jos pelissä on vain yksi pelisilmukka, silmukan suoritusaika vaikuttaa suoraan siihen, kuinka nopeasti peli pyörii. Kuvitellaan, että pelisilmukkaa ei mitenkään hidasteta ja, että pelisilmukassa on vain kaksi toimenpidettä: ruudun uudelleenpiirto sekä syötteenlukeminen. Jos ruudun uudelleenpiirto kestää vaikkapa 10 millisekuntia ja syötteen lukeminen esimerkiksi 5 millisekuntia, tarkoittaa tämä, että yhden pelisilmukan kierroksen suoritusaika on 15 millisekuntia. Tuhat millisekuntia vastaa yhtä sekuntia. Nämä asiat tietämällä saadaan laskettua, montako kertaa pelisilmukka ehtii kiertää alusta loppuun yhden sekunnin aikana. Sekunnin aikana tapahtuvista ruudunpäivityksistä käytetään yleisesti lyhennettä *fps* (frames per second, suom. ruutua sekunnissa). Tässä esimerkissä *fps*-luku saadaan laskettua kaavalla 1000ms / 15ms = 66,7. Tämä tarkoittaa, että jos yhden pelisilmukan suorituskierroksen kesto on 15 millisekuntia, ruutu päivitettäisiin noin 67 kertaa sekunnissa, samoin myös syötelaitteen nappien tilat luettaisiin noin 67 kertaa sekunnissa.

Pelisilmukan toteuttaminen vaihtelee pitkälti ohjelmointikielestä riippuen. Toimintaidea pysyy kuitenkin kaikissa kielissä samanlaisena. Hyvän esimerkin sekä mallin hyvin toteutetusta pelisilmukasta C++ / DirectX kielellä saa Jason Olsonin kirjoittamasta artikkelista ”The One With The Game Loop” (Olson 2004).

4.3 Syötteenkäsittely

Syötelaitteella tarkoitetaan ohjelmoinnista puhuttaessa mitä tahansa laitetta, jolla luetaan käyttäjän toimia. Syötelaitteita on nykypäivänä kymmeniä erilaisia. Näppäimistö, hiiri ja perinteinen joystick ovat jo vanhoja keksintöjä. Uudempia innovaatioita ovat mm. ratti-, rumpu-, kitara-, miekka-, tanssimatto-, mikrofoni- ja lumilautaohjaimet. Jokaisella pelikonsolilla on omat peliohjaimensa ja lisäksi markkinoilla on useita valmistajia, jotka valmistajat useisiin laitteisiin sopivia ohjaimia.

Kohdelaitteet, joille peliä ollaan toteuttamassa ratkaisevat sen, mitä ohjaimia pelin tulisi tukea. Tämä ei kuitenkaan tarkoita sitä, että kaikkia pelejä pitäisi pystyä pelaamaan kaikilla erilaisilla ohjaimilla. Pelin tuottaja voikin määrätä ohjaimet, joita pelin ilmoitetaan tukevan. Loppuja ohjaimia ei testata pelin kanssa ollenkaan. Todennäköisesti muutkin ohjaimet toimivat pelissä, mutta koska peliä ei ole suunniteltu pelattavaksi muilla ohjaimilla, ei niillä luultavasti ole erityisen hyvä pelata peliä. DirectX:llä ohjelmoitaessa pelissä tehdään vain yksi syötteenkäsittely ja DirectX hoitaa varsinaisen laitepuolen syötteenlukemisen.

Mobiililaitteiden kanssa ei yleensä käytetä erillistä ohjainta, vaan pelin syötteen antaminen tapahtuu matkapuhelimen omalta näppäimistöltä. Mobiililaitteiden näppäimistöt eroavat toisistaan erittäin paljon ja tästä syystä mobiililaitteille joudutaan tekemään erilaisia versioita riippuen näppäimistön mallista. Standardit ovat helpottaneet tilannetta, vaikkakaan ne eivät ole täysin onnistuneet pelastamaan tilannetta.

Syötteenkäsittely on mahdollista toteuttaa usealla eri tavalla. Kuitenkin ymmärtämällä kaksi päätapaa, ymmärtää pääidean syötteen lukemisesta. Nämä kaksi tapaa ovat syötelaitteen tilantarkistus (engl. State checking) ja tapahtumankäsittely (engl. Event handling). Mahdolliset syötteenkäsittelymenetelmät määräytyvät osittain ohjelmointikielen mukaan. Esimerkiksi Java ohjelmointikieli mahdollistaa molempien syötteenkäsittelymenetelmien käyttämisen. Käsittelemme näistä ensimmäisenä tilantarkistuksen.

Tilantarkistaminen

Syötelaite koostuu tavallisesti mikrokytkimistä, jotka voivat joko olla päällä (= nappi on painettuna) tai ne voivat olla pois päältä (= nappi ei ole painettuna pohjaan). Tilantarkistamisella tarkoitetaan menetelmää, jossa tietyin väliajoin luetaan syötelaitetta ja tutkitaan ovatko jotkin syötelaitteen napit/näppäimet painettuna pohjaan.

Tilantarkistus kiinnitetään usein jonkin säikeen suoritukseen, jolloin voidaan helposti määritellä sovellukselle, kuinka usein nappien tilat tarkistetaan. Säikeitä käsitellään tarkemmin luvussa 4.12. Ongelmana tilantarkistamisessa on kuitenkin se, että jos nappien tilat tarkistetaan esimerkiksi kymmenen kertaa sekunnissa, pelaaja voi ehtiä painaa nappia siten, ettei napin painallus rekisteröidy ollenkaan. Tämän ongelman voisi ratkaista

siten, että näppäinten tilat tarkistettaisiin useammin, mutta tämä olisi turhaa prosessointiajan tuhlaamista. Toinen ongelma tilantarkistamisessa on se, että pelissä ei voida mitenkään tietää nappien painamisjärjestystä. Ratkaisun molempiin edellä mainittuihin ongelmiin tarjoaa tapahtumankäsittely.

Tapahtumankäsittely

Tapahtumankäsittelyn tarkoitus on se, että kaikki nappien painallukset rekisteröidään jonomaiseen rakenteeseen, josta ne käsitellään järjestyksessä. Prosessointiaikaa annetaan koko ajan näppäinten käsittelyyn. Tämän menetelmän avulla tiedetään tarkalleen, mitä nappeja on painettu ja missä järjestyksessä niitä painettiin.

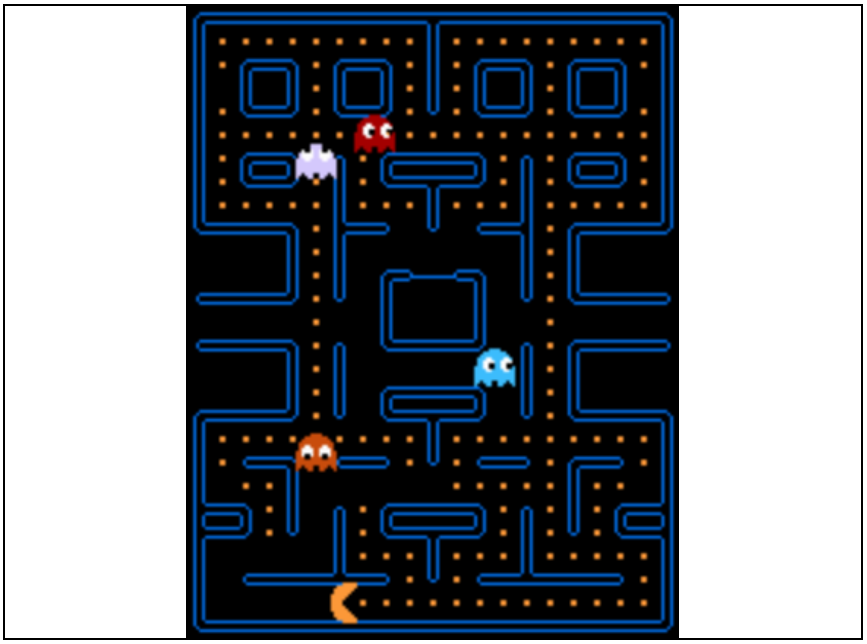
Sillä hetkellä kun käyttäjä painaa näppäimistöltä jonkin näppäimen pohjaan aletaan pinoon asettaa näppäinpainallus tapahtumia. Tässä saattaa tulla ongelmaksi vain yhden näppäinpainalluksen lukeminen. Tästä syystä tapahtumankäsittely toteutetaan yleensä siten, että käytetyistä napeista/näppäimistä tehdään muuttujia, joihin tallennetaan napin tila. Tavallisesti muuttujatyyppinä käytetään boolean tyyppiä (= kyllä/ei). Tämän jälkeen muuttujien arvoja tutkimalla päätetään, miten näppäimiä käytetään.

Edellä mainitulla tavalla pystytään toteuttamaan tilanne, jossa halutaan liikuttaa esimerkiksi laatikkoa niin kauan kuin nappi on painettuna pohjaan. Laatikon liikutusnopeuteen ei vaikuta näppäintenpainallustenmäärä tai prosessointiaika, vaan esimerkiksi säikeen suoritusaika. Tätä menetelmää käyttämällä onnistutaan saamaan tilantarkistamisen tuomat hyödyt myös tapahtumankäsittelyyn.

4.4 Tekoäly

Tekoäly (engl. Artificial Intelligence – A.I.) käsitteellä tarkoitetaan pelimaailmassa tietokoneen hallitsemien asioiden järkevältä tuntuvaa toimintaa. Viholliset ja muut peleissä itsenäisesti toimivat hahmot omaavat kaikki tekoälyn. Tekoälyn laatua mitattaessa käytetään tavallisesti vertailupohjana realistisuutta eli yhtenevyyttä omaan maailmaamme. Autopeleissä hyvin toimiva tekoälyn ohjaama auto ei törmäile jokaisessa mutkassa pelaajan autoon tai radan reunoihin, vaan toimii mahdollisimman realistisesti kaikissa mahdollisissa (ja mahdottomissakin) tilanteissa. Hyvin toteutetuissa autopeleissä tekoälylle voidaan asettaa mm. erilaisia tunnetiloja, joiden mukaan tekoälykuljettaja tekee päätöksiä toimiessaan.

Tekoäly on ollut osa tietokonepelejä jo pelien alkuajoista lähtien. Hyvänä esimerkkinä tekoälyn esiintymisestä peleissä on vuonna 1980 Namco:n julkaisema peli Pac-Man. Pac-Man on erittäin kuuluisa ylhäältäpäin kuvattu toimintapeli, jossa pyritään keräämään kentästä kaikki reiteillä olevat pallot siten, etteivät neljä eriväristä vihollista saa pelaajaa kiinni (Kuvio 3).



Kuvio 3 *Pelitilannekuva vuonna 1980 julkaistusta Pac-Manista*

Pac-Manissa jokaiselle neljälle viholliselle on ohjelmoitu erilainen tekoälymalli. Punainen vihollinen (Blinky) pyrkii seuraamaan pelaajan reittiä mahdollisimman suoraan, kun taas vaaleanpunainen vihollinen (Pinky) pyrkii katkaisemaan pelaajan reitin kääntymällä mutkista eri suuntaan punaisen vihollisen kanssa. Vaaleansininen vihollinen (Inky) vaihtelee toimintaansa radikaalisti, välillä jahdaten ja välillä jopa paeten pelaajaa. Oranssi vihollinen (Clyde) vaeltelee omia arvottuja reittejään. Yhdessä nämä neljä erilaista tekoälymallia tekevät Pac-Manista erittäin haastavan pelin. (Pac-Man2005)

Aikojen saatossa peleissä tekoälyn oikeaoppinen toiminta on tullut koko ajan tärkeämmäksi. Mitä realistisempaa tekoälyä pelissä pyritään toteuttamaan, sitä hankalammaksi tekoälyn ohjelmointi muodostuu. Tekoälyn ohjelmoinnin helpottamiseksi on pyritty löytämään erilaisia keinoja. Yksi yleisesti käytetty keino on tilakoneen hyödyntäminen, tästä lisää seuraavassa luvussa.

Reitinhaku

Reitinhaku on yksi tärkeimmistä tekoälyn toimista, jonka avulla pystytään vaikuttamaan siihen, miten realistisen kuvan peli välittää pelaajalle. Reitinhaulla tarkoitetaan sitä, että tietokoneen ohjaamat hahmot kulkevat järkeviä reittejä pitkin paikasta toiseen törmäilemättä pelissä oleviin esineisiin tai jäämättä niihin jumiin. Reitinhaku on osa miltei kaikentyylisiä pelejä. Järkevän reitinhaun toteuttaminen ei usein ole kuitenkaan kovin helppoa.

Reitinhaun toteuttamisessa käytetään usein niin kutsuttua A* algoritmiä (algoritmi tarkoittaa sarjaa vaiheita, jotka suorittamalla on mahdollista

saada ratkaistua määritelty ongelma). A* algoritmi soveltuu hyvin reitinhakuun, jossa pitää löytää sopiva reitti kahden etäällä olevan pisteen välillä. Algoritmin toiminta perustuu siihen, että kentässä on etukäteen määritelty esteet sekä pisteet, joihin tietokoneen ohjaamat hahmot voivat kulkea. Lisäksi kenttiin pitää määritellä näiden pisteiden keskinäiset kulkumahdollisuudet (Nareyek 2004). Reitin alkupisteestä aletaan etsiä mahdollisia reittejä kohti päätepistettä. Algoritmin mukaisesti kaikkia reittejä ei käydä läpi, vaan ainoastaan lupaavimmat reitit tutkitaan. Lupaavimpia reittejä ovat ne, joissa kahden välipisteen välinen etäisyys on mahdollisimman pitkä. A* algoritmin mukainen reitti löytyy, kun lupaavimmat mahdollisuudet on käyty läpi ja lyhyin mahdollinen reitti löydetty.

Reitinhaku saattaa tuntua aluksi erittäin vaikeasti lähestyttävältä asialta. Tässä kappaleessa käsiteltiin reitinhaun osalta vain kehittynyt reitinhakualgoritmi pintapuolisesti. Kannattaakin ottaa huomioon, että reitinhaun toteuttamiseen on myös yksinkertaisempia mahdollisuuksia. Esimerkkinä yksinkertaisemmasta reitinhakualgoritmistä on algoritmi, joka käy systemaattisesti läpi mahdollista reittiä kunnes löytää ensimmäisen reitin, jonka kautta on mahdollista päästä alkupisteestä loppupisteeseen. Tällaisen algoritmin käyttö saattaa joissakin tilanteissa näyttää järjettömältä, mutta varsinkin ensimmäistä reitinhaku algoritmiä rakennettaessa kannattaa mielestäni lähteä jonkin itselle selvän reitinhakumenetelmän rakentamisesta, eikä heti pyrkiä täydellisen reitinhakualgoritmin rakentamiseen.

4.5 Tilakone

Tilakone (engl. state machine) on yleisesti käytetty käsite peliohjelmoinnissa. Tilakoneita voidaan hyödyntää peleissä useissa eri yhteyksissä, vaikka useimmiten tilakonetta hyödynnetään vain tekoälyn ohjaamien hahmojen/vihollisten toteuttamisessa. Pelissä asiat, joiden käyttäytyminen perustuu erilaisiin tiloihin tai vaiheisiin, voidaan toteuttaa niin kutsutun tilakone-mallin mukaisesti. Tilakoneen tuomat suurimmat hyödyt tulevat esille lähdekoodin helppolukuisuutena sekä erilaisten tilojen helppona hallintana.

Tekoäly ja tilakoneet

Tietokoneen ohjaamien hahmojen tekoälyn hallinnointia pyritään yleensä helpottamaan tilakoneen avulla. Hahmoille määritellään erilaisia tiloja, joiden mukaisesti hahmon on mahdollista toimia. Nämä tilat voidaan ohjelmoida erikseen ja eri tilojen välisille siirtymille voidaan helposti tehdä erilaisia esivaatimuksia. Esivaatimuksilla tarkoitetaan esimerkiksi sitä, ettei tilaa voida muuttaa suoraan tilasta A tilaan C vaan, että tilasta A voidaan siirtyä tilaan C vain tilan B kautta. Tietokoneen ohjaamalla vihollisella voisi olla perinteisessä ammuskelupelissä seuraavanlaisia tiloja:

- Herää henkiin (Alustetaan muuttujat…)
- Tutki ympäristöä (Kerätään tietoa ympäristöstä…)
- Etsi pelaajaa (Lähdetään etsimään pelaajaa…)
- Havaitse pelaaja (Kun pelaaja havaitaan lasketaan pelaajan sijainti…)
- Kutsu apuvoimia (Pyydetään muita hahmoja paikalle…)
- Vaihda asetta (Vaihdetaan sopiva ase pelaajaa varten…)
- Hyökkää pelaajan kimppuun (Hyökätään pelaajan kimppuun…)
- Peräänny (Kun energia on vähissä peräännytään…)
- Etsi piilopaikkaa (Yritetään löytää paikka, josta pelaaja ei löydä…)
- Odota (Pysähdytään odottamaan ja katsellaan samalla ympärille…)
- Kuole (Kun energia loppuu, tuhotaan hahmo…)

Jokaisen tilan toiminta tilakoneessa ohjelmoidaan erikseen, vaikka yhteisiä funktioita voidaan tietysti käyttää. Tämä saattaa aluksi kuulostaa työläältä tehtävältä, mutta tilakonetta oikein hyödynnettynä saatetaan ohjelmoinnin myöhemmässä vaiheessa säästyä isoiltakin ongelmilta helpommin seurattavan lähdekoodin ansiosta. Varsinaista tilakonetta varten kirjoitettavan lähdekoodin määrä on yleensä varsin pieni.

Toteutus

Tilakoneita voidaan hyödyntää kaikilla ohjelmointikielillä. Tilakoneen oikeanlainen toiminta vaatii tiedon siitä, mikä tila on aktiivinen sekä lisäksi tarvitaan yleensä tieto edellisestä aktiivisesta tilasta. Asialla, jota tilakoneen avulla pyritään rakentamaan, on tavallisesti jokin funktio, jota kutsutaan esimerkiksi viisi kertaa sekunnissa. Kuvio 4 on yksinkertainen esimerkki siitä, kuinka tilakonetta voidaan hyödyntää tietokoneen ohjaaman hahmon ohjelmoinnissa.

```
final int HERAAHENKIIN          = 100;
final int TUTKIYMPARISTOA       = 101;
final int ETSIPELAAJAA          = 102;

int edellinenTila, nykyinenTila;

public void Toimi(){

        switch(nykyinenTila){

                case HERAAHENKIIN:
                        alustaMuuttujat();
                        arvoPaikka();
                        break;

                case TUTKIYMPARISTOA:
                        kaannyYmpari();
                        keraaTietoa();
                        break;

                case ETSIPELAAJAA:
                        valitseSuunta();
                        katseleYmparillesi();
                        kuljeEteenpain();
                        break;

                default:
                        break;
        }
}
```

Kuvio 4 Koodiesimerkki yksinkertaisen tilakoneen toteuttamisesta

Kuvion 4 tarkoituksena on esittää tapa, jolla tilakone voidaan rakentaa hyödyttämään ja helpottamaan tietokoneen ohjaamien hahmojen hallintaa. Tärkeintä esimerkistä on huomata jokaista tilaa vastaavan vakion määrittely sekä switch-case lauseen käyttäminen. Jokaista tilaa kohti toteutetaan yksi case-haara. Tilakoneet toteutetaankin suurimmassa osassa tapauksista juuri näitä kahta asiaa hyödyntämällä. Switch-case lauseita hyödyntämällä muissakin funktioissa voidaan toteuttaa erilaista toimintaa

helpommin kuin tekemällä iso määrä if-else if lauserakenteita. Switch-case lauseet pitävätkin tilakoneen toiminnan helpommin seurattavana.

4.6 Näytön uudelleenpiirto sekä kaksoispuskurointi

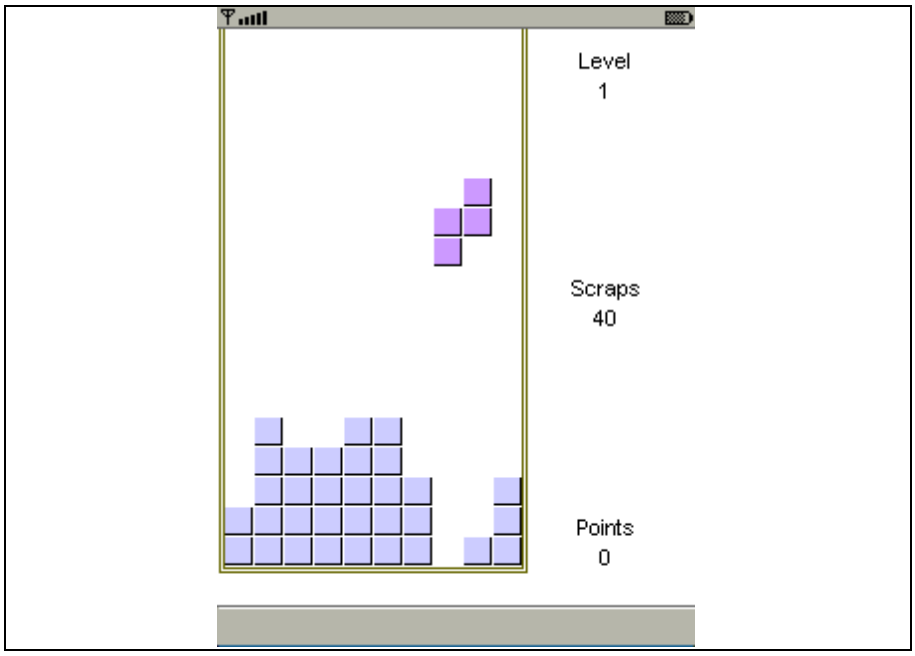
Uudelleenpiirtämisellä (engl. repaint) tarkoitetaan tapaa jonka avulla mm. pelit saadaan liikkumaan kohdelaitteen näytöllä. Luvussa 4.3 käsiteltiin pelisilmukkaa ja kappaleesta kävikin ilmi, että ruutu pitää piirtää uudelleen tietyin väliajoin. Tavallisesti peleissä pyritäänkin piirtämään ruutu niin useasti, kuin se on vain laitteiston kannalta mahdollista.

Tavallinen televisio piirtää kuvan ruudulle 25 kertaa sekunnissa. Tällä tavalla televisioon saadaan ns. liikkuva kuva. Normaalitilanteessa ihminen ei näe ruudun piirtämistä ollenkaan, vaan hän näkee vain tasaisesti liikkuvan kuvan. Jos televisiossa kameraa käännetään johonkin suuntaan sopivalla nopeudella, ihminen saattaa nähdä kuvan piirtymisen pätkivänä. Tietokonepeleissä uudelleenpiirtomääränä pyritään pitämään nykyään vähintään 40 piirtokertaa sekunnissa. Suuremman piirtomäärän tarve johtuu siitä, että televisio pehmentää jokaista kuvaa (engl. Motion blur), tietokoneella tällaista ei tehdä. Matkapuhelinten peleissä uudelleenpiirto määränä pyritään yleensä pitämään 15 kertaa sekunnissa.

Kohdelaitteesta riippuen uudelleenpiirtonopeus voi olla ongelma. Siinä vaiheessa kun laite ei enää pysty piirtämään ruutua tarpeeksi nopeasti, ihminen alkaa nähdä ruudulla ns. pätkimistä. Tämän pätkimisen havaitsee vain, kun on kyse liikkuvasta kuvasta. Näytön uudelleenpiirtäminen saattaa olla niin raskas operaatio, että joissakin tapauksissa näytöstä halutaan piirtää ainoastaan muuttuneet kohdat.

Näytön osittainen uudelleenpiirtäminen

Joissakin peleissä näyttöä ei ole järkevää piirtää kokonaan uudestaan. Tällaisia tapauksia on yleisimmin peleissä, jossa ruudulla on suhteellisen vähän liikettä, tai pelinäkymä pysyy koko ajan samassa kohdassa. Kuviosta 5 on nähtävissä Tampereen ammattikorkeakoulun mobiiliohjelmointi kurssille toteuttamastani Scraps nimisestä pelistä kuvakaappaus. Peli-idea on ohjata putoavat palat ruudun alareunaan siten, että ne muodostavat täysinäisiä poikittaissuuntaisia rivejä. Peli mukailee vanhaa klassikkopeliä Tetristä.



Kuvio 5 Scraps pelitilanne

Scraps on hyvä esimerkki siitä, miten ruutua on turha uudelleenpiirtää kokonaan jokaisella uudelleenpiirtokierroksella. Scrapsin pelitilanteessa ainoa muuttuva osa ruudulla on laskeutuva pala, ja tästä syystä tällaisessa tilanteessa kannattaakin piirtää ainoastaan putoavan palan ympärillä oleva alue uudestaan. Siinä vaiheessa kun pala törmää johonkin, tehdään ruudun täydellinen uudelleenpiirto. Tässä vaiheessa ruudun oikeassa laidassa olevat tekstit muuttuvat ja mahdollisesti poistetaan rivejä.

Scraps:ssä ruudun osittainen uudelleenpiirto oli helppo toteuttaa, koska pelissä ei ole liikkuvaa taustagrafiikkaa tai muuta asiaa hankaloittavaa tekijää. Toteutin Scraps:issä ruudun osittaisen uudelleenpiirtämisen siten, että ruudulla piirretään putoavan palan päälle valkoinen laatikko ja tämän jälkeen putoava pala piirretään uudestaan oikeaan kohtaan. Näin säästyttiin koko ruudun uudelleenpiirtämiseltä.

Peleissä, joissa käytetään taustagrafiikkaa, pitää piirtämisessä käyttää rajausta piirtoalueen suhteen, jos näyttöä ei haluta piirtää kokonaan uudestaan. Ohjelmointikielissä tämän tekniikan englanninkielinen nimi on clipping. Tämän tekniikan idea on se, että ruudulta määritetään alue, joka halutaan piirtää uudestaan. Tämän jälkeen piirtäminen voidaan tehdä aivan kuin se vaikuttaisi koko näytön alueelle, vaikka todellisuudessa ainoastaan sellaiset piirtokäskyt toteutetaan, jotka mahtuvat tämän määritellyn alueen sisäpuolelle. Tällä tekniikalla saatava tehohyöty on huomattava.

Kaksoispuskurointi

Kaksoispuskuroinnin avulla pyritään helpottamaan koneen piirtämistaakkaa sekä pyritään näyttämään käyttäjälle onnistunut kuva ilman kuvan vilkkumista tai vääristymistä. Ilman kaksoispuskurointia kaikki peleissä

tehtävät piirtotoimenpiteet tehtäisiin suoraan näkyville. Tätä ei yleensä haluta tehdä, sillä suoraan näytölle piirrettäessä käyttäjä alkaa nähdä kuvan vilkkumista. Yleensä pelialue maalataan täysin tyhjäksi esimerkiksi valkoisella värillä ja tällaisessa tapauksessa pelaaja näkee nopean vilahduksen täysin valkoisesta ruudusta. Yleensä valkoinen ruutu on näkyvissä vain niin vähän aikaa, että pelaaja ei ehdi havaitsemaan sitä muuten kuin vain eräänlaisena silmiä rasittavana vilkkumisena.

Tuplapuskurointi toteutetaan siten, että varsinaista piirtämistä ei koskaan tehdä suoraan näytölle vaan piirto toimet tehdään puskuriin, joka ei ole näkyvissä. Kun puskuriin piirtäminen on valmis, puskuri siirretään suoraan näytölle kokonaisena. Joissakin ohjelmointikielissä tuplapuskurointia ei tarvitse itse ollenkaan tehdä, vaan kaikki ruudulle piirrettävät asiat tuplapuskuroidaan automaattisesti. (Powell 2004).

4.7 Kaksiulotteisuus ja kolmiulotteisuus

Kaksiulotteisuudella tarkoitetaan sitä, että ruudulla näkyvät asiat omaavat vain kaksi ulottuvuutta. Nämä ulottuvuudet ovat vaaka- ja pystysuunta, eli X- ja Y-akselit. Yleensä kaksiulotteisen pelin tunnistaa syvyysvaikutelman puuttumisesta.

Kuvio 6 *Kuvakaappaus Nintendon Super Mario Bros. 3:sta*

Kuviosta 6 on nähtävissä kuuluisa ja samalla tyyliltään perinteinen kaksiulotteinen ”hyppelypeli”. Peli julkaistiin jo vuonna 1990 Nintendo Entertainment Systemille (Piqna2004). Pelin tavoitteena on pelastaa prin-

sessa ilkeältä Koopalta. Prinsessan pelastamista varten pelattavan pelihahmon on kuljettava kymmenien kenttien läpi välillä väistellen ja välillä taistellen erilaisia vihollisia vastaan.

Kolmiulotteisessa pelissä on kaksiulotteiseen peliin nähden yksi uusi ulottuvuus, syvyys (Z-akseli). Kolmiulotteisissa peleissä kameraa pystytään liikuttamaan täysin vapaasti kaikkiin kolmeen suuntaan. Kuviossa 7 on nähtävissä erittäin hyvin kolmiulotteisuutta hyödyntävä peli.

Kuvio 7 *Kuvaruutukaappaus Rockstar Gamesin vuonna 2005 julkaisemasta "Grand Theft Auto: San Andreas" -nimisestä pelistä*

Pelitoteutuksen kannalta pelityypit eroavat melko paljon toisistaan, vaikka suurin osa tässä tutkintotyössä käsiteltävistä asioista pätee molempiin pelityyppeihin. Yleensä kolmiulotteisuus yhdistetään suoraan kaupallisiin, ja budjetiltaan kalliisiin peleihin. Tämä ei onneksi pidä täysin paikkaansa ja tälläkin hetkellä toteutetaan kaupallisia kaksiulotteisia pelejä.

Kolmiulotteisten pelien toteuttaminen vaatii tavallisesti huomattavasti enemmän matematiikkaa kuin perinteisen kaksiulotteisen pelin toteuttaminen. Itse näen asian siten, että peliohjelmointia opiskelevan kannattaisi valita ohjelmointikieleksi sellainen kieli, jolla on mahdollista ohjelmoida kolmiulotteisia pelejä, mutta kuitenkin peliohjelmoinnin opiskelu kannattaa aloittaa kaksiulotteisista peleistä.

4.8 Kenttäeditori ja muut pelin toteuttamisen aputyökalut

Peliä toteutettaessa kaikki ohjelmointityö ei liity pelkästään pelin ohjelmointiin. Tavallisesti peliä toteutettaessa tarvitaan vähintään yhtä ja yleensä useampaakin aputyökalua pelin ohjelmoinnin ja toteuttamisen helpottamiseksi. Yksi keskeisimmistä aputyökaluista, jota tarvitaan lähestulkoon jokaisen pelin kehittämisessä, on kenttäeditori. Tämä alaluku painottuukin suurimmaksi osaksi kenttäeditoreihin sekä niiden toimintaan. Kenttäeditoreiden painottaminen johtuu siitä, että kenttäeditorit ovat ainoa aputyökalu peliohjelmoinnissa, jota tarvitaan sellaisenaan kaikentyyppisten pelien toteuttamisessa. Muiden aputyökalujen tarve on yleensä peli- tai pelityyppi-kohtaista. Muiden aputyökalujen tarve tulee yleensä selvästi esille peliä toteutettaessa.

Kenttäeditorit

Kenttäeditorilla tarkoitetaan aputyökalua, jonka avulla voidaan suunnitella ja toteuttaa kenttiä toteutettavaan peliin. Kenttäeditoreita on löydettävissä internetistä useita ilmaisia sekä maksullisia.

Suurimmassa osassa tapauksista valmiita kenttäeditoreja ei voida suoraan käyttää pelin kenttien suunnitteluun ja toteuttamiseen, vaan valmista kenttäeditoria joudutaan joko muokkaamaan tai luomaan kokonaan uusi editori, joka sopii paremmin toteutettavan pelin aputyökaluksi. Yksi syy tähän on se, että pelit ohjelmoidaan yleensä melko pitkälle ennen kenttäeditorin toteuttamista tai valitsemista. Tilanne onkin yleensä se, että valmiista kenttäeditoreista ei löydy valmiina kaikkia ominaisuuksia, joita pelissä tarvittaisiin kenttien toteuttamiseen. Toisena syynä on reaaliaikaisen kenttäeditoinnin puuttuminen, joka on itsessään yksi tärkeimmistä kenttäeditoreiden ominaisuuksista (Rouse 2000:378).

Reaaliaikaisella kenttäeditoinnilla tarkoitetaan mahdollisuutta, jossa kenttäsuunnittelija voi tehdä suoraan muutoksia kenttiin ja nähdä samalla muutosten vaikutukset pelissä. Yleensä tämä on toteutettu siten, että kenttäeditorissa on pieni ikkuna pelinäkymälle, jossa kenttäsuunnittelija pystyy siirtämään pelinäkymää kentässä esimerkiksi nuolinäppäimiä käyttäen ja näin näkemään, miltä kenttä todellisuudessa näyttää pelissä.

Kenttäeditorilla on useita eri tehtäviä. Yksi tärkeimmistä kenttäeditorin tehtävistä on auttaa kentän graafisen yleisilmeen suunnittelussa. Tämä ei suinkaan tarkoita sitä, että kenttäeditorilla toteutettaisiin pelissä käytettävät grafiikat. Kenttäeditorin tarkoitus kaksiulotteisissa peleissä on se, että kenttäsuunnittelijalla on käytettävissä tietty määrä erilaisia pieniä graafisia paloja (engl. tile). Näitä graafisia paloja asettelemalla kenttäsuunnittelija rakentaa kenttiä, jotka hänen mielestään sopivat ulkoasullisesti pelin tyyliin. Kaksiulotteisten pelien kentät koostuvat yleensä pysty- ja vaakasuunnassa vierekkäin olevista grafiikkapalasista.

Kokonaisvaltaisella kenttäeditorilla pystytään myös määrittelemään mahdollisten tekoälyn ohjaamien hahmojen sekä pelissä olevien tavaroi-

den sijainnit. Laajemmissa peleissä pitäisi pystyä myös määrittelemään jokaiselle tekoälyn ohjaamalle hahmolle käyttäytymismallit, jos sellaisia on ohjelmoitu peliin. Kenttäeditorin toimintaa kokonaisuudessaan pohdittaessa pitäisikin etsiä vastausta kysymykseen: ”Mitä kaikkia ominaisuuksia tarvitsen kenttäeditoriin, jotta voin helposti suunnitella ja toteuttaa hyviä kenttiä?”. Kun kenttäeditorin toteuttaja osaa vastata tähän kysymykseen, pystyy hän todennäköisesti toteuttamaan peliin hyvin soveltuvan kenttäeditorin.

Kaksiulotteisen pelin kenttäeditori voidaan toteuttaa millä tahansa ohjelmointikielellä, jossa voidaan lukea ja piirtää kuvia, sekä kirjoittaa tiedostoon. Kenttäeditorin on kyettävä lukemaan kuvia siinä muodossa jossa niitä pelissäkin tullaan käyttämään. Kuvat ladataan kuvasarjoina (engl. tileset), joista jokainen pieni kuva luetaan erikseen liikuteltavaksi objektiksi. Jokaista kuvaa voi olla yhdessä kentässä määrittelemätön määrä, poikkeuksena suunnitellut rajoitukset. Tällainen rajoitus on esimerkiksi sääntö, että kentässä voi olla vain yksi kentän läpäisykohta.

Kenttäeditorin toiminnalle on ratkaisevaa, että sen avulla voidaan tallentaa suunniteltu kenttä hyvässä muodossa peliä varten. Lisäksi kenttäeditorilla pitää pystyä siirtämään kenttiä helposti valmiiseen peliin. Kenttää ei koskaan siirretä peliin suoraan kenttäsuunnittelijan tekemänä kuvana, vaan suunnittelijan suunnittelema kenttä tallennetaan numero- tai tekstimuotoisena tiedostona pelihakemistoon.

4.9 Lokalisointi

Lokalisoinnilla tarkoitetaan eri kieliryhmiä tai eri valtioita varten tehtäviä erillisiä versioita pelistä. Lokalisointi tulee ottaa huomioon jo peliä suunniteltaessa, mutta yksityiskohtaisemmin lokalisointi tarvitsee ottaa huomioon vasta peliä toteutettaessa. Yleisimpiä lokalisointikohteita ovat pelissä olevat tekstit, mutta usein lokalisoidaan myös mm. ääniraitoja (musiikit / äänet / puheet) sekä pelin sisältöä.

Pahimmillaan huonosti suunniteltu lokalisointi, tekee pelin kääntämisen muille kielille mahdottomaksi kustannustehottomuutensa takia. Lokalisointia on vaikeata, jos ei jopa mahdotonta toteuttaa onnistuneesti ja kannattavasti peliprojektin loppuvaiheessa. Tästä syystä tulisikin jo pelin suunnitteluvaiheessa tehdä tarvittavat suunnitelmat lokalisoinnin toteuttamiseksi (Chandler 2005). Ilman kunnollista lokalisoinnin huomioon ottamista, joudutaan lähdekoodit käymään manuaalisesti läpi etsien tekstejä, jotka on kovakoodattu lähdekooditiedostoihin.

Tekstien lokalisointi

Oikeaoppisesti toteutettuna tekstien lokalisointi tarkoittaa yhden tiedoston toimittamista käännöstoimistoon, ja heiltä saatavan valmiit käännökset sisältävän tiedoston lisäämistä pelipakettiin. Yksi yleinen käytäntö onkin sijoittaa kaikki tekstiresurssit erilliseen tiedostoon, joka on helppo

toimittaa käännettäväksi. Tätä tapaa käytettäessä tehdään pelin käynnistyksen yhteydessä päätös siitä, mitä tekstejä sisältävää tiedostoa pelissä käytetään. Yleensä tämä toteutetaan pelin käynnistyksen yhteydessä kielivalikolla, josta pelaaja saa valita käytettävän kielen.

Tekstien lataaminen tiedostosta voidaan tehdä ajonaikaisesti tai heti alkulatausten yhteydessä. Yleisempää on tekstien lataaminen muistiin heti kielivalikon jälkeen. Tällöin tekstejä voidaan käyttää, kuin ne olisi määritelty erillisiin muuttujiin. Yleinen käytäntö onkin nimetä jokainen teksti erikseen vakioksi, jonka arvo luetaan tiedostosta pelin käynnistyksen yhteydessä.

Lokalisoinnin sudenkuopat

Ensimmäistä peliä toteuttaessaan pelinkehittäjän voi olla vaikea kuvitella lokalisointia muuna kuin tekstiresurssien kääntämisenä muille kielille. Todellisuudessa lokalisointi on huomattavasti laaja-alaisempi aihe. Lokalisointi käsittää myös pelin sisällön sopivuuden erilaisiin kulttuureihin. Käytännössä tämä tulisi ottaa huomioon muun muassa siten, ettei pelissä loukata tietoisesti tai tiedostomatta mitään kulttuuria tai uskontoa.

Vaikka peli olisi lokalisoitu tekstien suhteen onnistuneesti, saattaa peliin jäädä huomaamattomia väärän kielisiä tekstejä. Tällaisia tekstejä saattaa helposti jäädä esimerkiksi kuvatiedostoissa oleviin teksteihin. Tekstejä lokalisoitaessa tulisikin tarkistaa varsinaisten pelitekstien lisäksi myös pelin grafiikat etsien ”piirrettyjä” tekstejä.

4.10 Fonttimoottori

Usein pelejä toteutettaessa halutaan käyttää erilaista tekstityyppiä (eli kirjasinta) kuin mitä käyttöjärjestelmässä on sisäänrakennettuna. Yleensä pelin graafista ilmettä rakennettaessa graafisen ilmeen suunnittelijat haluavat itse määritellä pelissä käytettävän kirjasimen tyylin, värin ja koon (ja samalla varmistua siitä, että kirjasin löytyy kaikista käytettävistä laitteista). Onkin harvinaisempaa nähdä peli, jossa käytetään käyttöjärjestelmästä valmiiksi löytyviä kirjasimia kuin peli, jossa käytetään itse rakennettua kirjasinta. Jos toteutettavassa pelissä halutaan käyttää omaa kirjasinta, pitää kirjasimelle toteuttaa oma kirjasimen piirtojärjestelmä. Tällaisesta järjestelmästä käytetään nimitystä fonttimoottori.

Käyttämällä peliä (tai yritystä) varten toteutettua fonttimoottoria saavutetaan useita etuja. Suurimpana etuna on se, että graafikot voivat luoda peliin tyylillisesti parhaiten sopivan kirjasimen. Lisäksi käyttämällä itse tehtyä fonttimoottoria, saadaan tarkka tieto siitä, miten tekstit todellisuudessa piirretään ruudulle. Useimmissa ohjelmointikielissä on valmiiksi ohjelmoidut funktiot tekstin ruudulle piirtämistä varten. Ohjelmoijaa päästetään harvoin vaikuttamaan piirtotapaan. Hyvin toimiva ja loppuun asti suunniteltu fonttimoottori ei ole kertakäyttötavaraa, vaan sitä voidaan

käyttää mahdollisesti hieman mukauttamalla myös muissa yrityksen toteuttamissa peleissä.

Tehokkaan ja hyvin toimivan fonttimoottorin toteuttaminen ei kaikissa tapauksissa ole helppo eikä nopea asia. Tehokkaan tekstinpiirtojärjestelmän ohjelmoiminen vaatii tavallista enemmän keskittymistä ja oikeiden ohjelmointitapojen käyttämistä. Varsinkin jos pelissä on paljon ruudulle piirrettävää tekstiä, tulee fonttimoottori toteuttaa erityisellä tarkkuudella. Lisäksi erikoismerkit ja käyttäjän syötteen vastaanottaminen tulee ottaa huomioon omaa fonttimoottoria suunniteltaessa. Oma fonttimoottori määrittelee pelissä käytössä olevat merkit ja tästä syystä ohjelmoijan täytyy pitää huoli siitä, ettei pelaaja pääse syöttämään sellaisia merkkejä joita oma fontti ei tue.

Rakentaminen

Fonttimoottori koostuu yleensä vähintään kolmesta osasta. Ensimmäinen osa on varsinainen luokka, jossa on fonttimoottorin toiminnalle tärkeät asiat kuten tekstin piirtofunktiot (määrittelee kuinka teksti piirretään ruudulle). Piirtofunktioita tulee yleensä olla useita erilaisia riippuen siitä, miten tekstiä halutaan ruudulle piirtää. Fonttimoottori-luokkaan tulisi yleensä rakentaa myös tekstin asemointijärjestelmä, jolla voidaan määrittää tekstin alkamispiste (X ja Y akselilla) tekstin eri kulmiin. Luokkaan tulee myös tehdä taulukko, joka sisältää käytössä olevat merkit samassa järjestyksessä kuin mitä ne on fonttikuvaan piirretty.

Toisena osana fonttimoottorissa tarvitaan kuva, joka sisältää piirrokset jokaisesta käytettävästä merkistä. Yleisimmin tämä toteutetaan siten, että jokainen käytössä oleva merkki piirretään vierekkäin yhteen kuvatiedostoon (Kuvio 8). Jokaisen yksittäisen merkin tulee olla oikean kokoinen suhteessa toisiin merkkeihin. Yleensä kuva jaetaankin erillisiin ruutuihin, joihin jokainen merkki piirretään erikseen. Ruutujen korkeudet ovat yleensä samat kaikissa pienissä kuvissa, mutta yksittäisen ruudun leveys riippuu piirrettävästä merkistä.

Kuvio 8 *Esimerkki kirjasinkuvasta (Typebox)*

Kuviossa 8 nähdään esimerkki kirjasimesta. Tavallisesti kirjasimeen ei kannata laittaa mitään taustaväriä. Tämä johtuu siitä, että kirjasimen taustaväri on helppo päättää kooditasolla, mutta taustavärin pois ottaminen kirjasinkuvasta on miltei mahdotonta. Kuvio 8 poikkeaa lisäksi tavallisesti peleissä käytettävästä kirjasimesta siitä syystä, että kirjasin on tasavälinen. Tämä tarkoittaa sitä, että kaikki merkit ovat toistensa kanssa yhtä leveitä (engl. monospace). Käyttämällä tämän tyyppistä kirjasinta tekstistä tulee konekirjoituksenomaista (Inkinen 2002).

Fonttimoottorissa tarvitaan myös kirjasimen määrittelytiedosto. Käytännössä tämä tiedosto sisältää jokaisen kirjasinkuvassa olevan merkin leveyden. Tavallisesti tällainen tiedosto voidaan muodostaa esimerkiksi siten, että jokaisen merkin leveys kirjoitetaan tiedostoon pilkulla erotettuna. Tällaisessa tapauksessa tulee ottaa huomioon, että tiedostossa olevien leveysarvojen on oltava samassa järjestyksessä kuvassa olevien merkkien kanssa.

4.11 Säikeet, ajastimet ja pelin tahdistus

Säikeellä tarkoitetaan ohjelman/pelin sisälle käynnistettävää omaa ”ohjelmaa”. Peli voikin käynnistää useita säikeitä, joista jokaisella on oma tehtävänsä pelin suorituksen kannalta. Tavallisesti peliohjelmoinnissa säikeitä käytetään asioiden ajastamiseen ja pysyvän toiminnan ylläpitämiseen. Esimerkiksi pelisilmukka on usein ohjelmoitu säikeeksi. Tieto-

kone pystyy suorittamaan vain yhtä säiettä jokaista tietokoneessa olevaa prosessoria kohti, joita on tavallisissa tietokoneissa vain yksi.

Jo jonkin aikaa on ollut myynnissä moniprosessorisia tietokoneita, jotka ainoina laitteina pystyvät suorittamaan useampia säikeitä samanaikaisesti. Lisäksi kolmannen sukupolven pelikonsoleissa on valmiiksi sisäänrakennettuna useampi prosessori, jotta säikeitä saadaan suoritettua päällekkäin. Tavallinen yksiprosessorinen tietokone toimii siten, että käyttöjärjestelmä jakaa suoritusaikaa jokaiselle säikeelle tasaisesti. Käyttöjärjestelmä antaa suoritusaikaa niin vähän yhdelle säikeelle kerrallaan, että käyttäjälle pelin/ohjelman toiminta näyttää siltä kuin tietokone tekisi ”monta asiaa samanaikaisesti”.

Peliohjelmoinnissa säie kannattaa mielestäni ajatella juuri sisäiseksi ohjelmaksi. Kun ohjelmoija määrää uuden säikeen käynnistettäväksi, tarkoittaa tämä sitä, että ohjelmoija haluaa suorittaa pelissä useita asioita samanaikaisesti pelin toteuttamisen helpottamiseksi tai pelin tehostamiseksi. Pelissä oleva vihollinen on hyvä esimerkki säikeestä. Siinä vaiheessa kun vihollinen ”synnytetään” kenttään, aloitetaan vihollissäie. Tämä säie kuvaa vihollista sen koko olemassaolon ajan. Säie kutsuu ennalta määritettyjä funktioita niin kauan, että säie lopetetaan. Säikeen avulla pystyttään pitämään huolta siitä, että vihollinen etsii pelaajaa ja toimii ennalta määritellyn tekoälynsä pohjalta.

Yksinkertaisessa peliohjelmoinnissa selvitään yhdellä säikeellä. Tarvittava säie on pelisilmukan pyörittämisestä vastaava säie. Säie kutsuu itseään tietyin väliajoin ja tekee ennalta määritetyt tehtävät. Tällaisia tehtäviä voivat olla esimerkiksi pelaajan sijainnin muuttaminen, syötteen vastaanottaminen ja ruudun uudelleenpiirtäminen. Useissa laajemmissa peleissä tarvitaan säikeitä enemmän kuin yksi.

Useiden säikeiden yhtäaikaiseen käyttöön sisältyy yksi tärkeä huomioon otettava asia, synkronointi. Säikeiden synkronoinnilla tarkoitetaan sitä, että säikeiden välillä huolehditaan siitä, etteivät säikeet pysty tekemään samanaikaisesti mitään, mikä saattaisi pelissä aiheuttaa ongelmatilanteita. Esimerkkinä tällaisesta tilanteesta voisi olla peli, jossa on käytössä kaksi säiettä. Ensimmäisen säikeen tarkoitus on pyytää pelaaja-oliolta pelaajan sijainti ruudun uudelleenpiirtämistä varten. Toisen säikeen tehtävänä on liikuttaa pelaaja-oliota. Tässä esimerkissä pelaajan liikutus tehdään siten, että aluksi pelaajan nykyinen sijainti siirretään kahteen muuttujaan prevPosX:n ja prevPosY:n. Tämän jälkeen sijaintimuuttujista positionX ja positionY poistetaan sen hetkiset arvot. Tämän jälkeen pelaajan uusi sijainti lasketaan perustuen prevPosX ja prevPosY muuttujien arvoihin. Jos tällaisessa tilanteessa käyttöjärjestelmä antaakin uudelleenpiirtosäikeelle suoritusaikaa ja säie pyytää pelaaja-oliolta sen hetkisen sijainnin, saattaa uudelleenpiirtosäie saada ainoastaan tyhjät arvot takaisin ja peli kaatua alustamattomien muuttujien käytön takia.

Edellä on yksinkertaistettu esimerkki siitä, millaisia ongelmia synkronoimaton usean säikeen käyttö saattaa aiheuttaa, mutta juuri tällaiset ongelmat ovat yleisiä huonosti suunnitellun säietoteutuksen yhteydessä. Esimerkin ongelmaan on kaksi ratkaisua. Ensimmäinen ratkaisu olisi pitää huoli siitä, että pelaaja-olion sijaintia ei missään tilanteessa nollata. Toinen, yleensä parempi, vaihtoehto olisi synkronoida säikeiden suoritus siten, että edellä esitettyä tilannetta ei pääsisi syntymään. Yksinkertaisissa peleissä kannattaa pyrkiä ratkaisemaan ongelmat hyödyntäen ensimmäistä ratkaisumallia, mutta hiemankin hankalammissa peleissä kannattaa tutkia synkronoinnin mukanaan tuomia mahdollisuuksia. Säikeiden synkronointia Java ohjelmointikielellä käsittelevä hyvä artikkeli löytyy osoitteesta http://today.java.net/pub/a/today/2004/08/02/sync1.html (Friesen 2004).

Säikeiden käyttäminen tuo peleihin yhden erittäin tärkeän lisämahdollisuuden eli helposti toteuttavan tahdistamisen/ajastamisen. Säikeiden suoritus pystytään keskeyttämään missä tahansa vaiheessa itse määritellyksi ajaksi. Tämä tarjoaa mahdollisuuden mm. yksinkertaisen animaation tekemiseen. Yksinkertaistetusti animaatio pystyttäisiin toteuttamaan siten, että suoritetaan kahta funktiota peräkkäin. Ensimmäisenä kutsutaan ”piirräkuva”-funktiota, joka valitsee ja piirtää oikean kuvan näytettäväksi ruudulla ja toisena funktiona kutsutaan säikeen ”nukutus”-funktiota siten, että parametriksi annetaan toivottu nukkumisaika. Jos parametriksi annetaan esimerkiksi 20, pyörisi animaatio noin 40–50 kuvan sekuntivauhdilla.

Oman kokemukseni mukaan mobiililaitteet tukevat usean säikeen yhtäaikaista käyttöä hyvin. Ainoa ongelma säikeiden käytössä on joissakin laitteissa epätarkka säikeen nukkumisaika. Käyttämällä useita säikeitä pelin pyörittämiseen ei varsinainen pelinopeus muuttuisi huomattavasti, vaikka epätarkkuutta säikeiden nukkumisajassa esiintyisikin.

4.12 Grafiikat peleissä

Hyvin suunniteltu graafinen ilme on tärkeä osa tietokonepelejä. Ikävä tosiasia on, että yksittäisenä asiana grafiikat ovat suurin vaikuttaja pelaajien tekemiin pelien ostopäätöksiin. Huono peli, josta on saatu otettua hyviä kuvaruutukaappauksia pelipakkaukseen, myy todennäköisesti paremmin kuin erittäin hyvä peli, jossa on huono graafinen ulkoasu. Tästä syystä grafiikoihin tulee kiinnittää erityistä huomiota peliä toteutettaessa.

Kaksiulotteisissa ja kolmiulotteisissa peleissä on erityyppinen lähestymistapa pelien graafisuuteen. Kaksiulotteisissa peleissä grafiikat koostuvat yleensä kuvista, joita liitetään käytettäväksi pelissä pelin tarvitsemalla tavalla. Käytännössä tämä tarkoittaa sitä, että esimerkiksi peleissä olevat animaatiot koostuvat kuvista, jotka eroavat vain hieman toisistaan ja joita piirretään nopeassa tahdissa toistensa tilalle. Näin saadaan peli näyttä-

mään siltä, kuin se sisältäisi liikkuvaa kuvaa. Kaksiulotteisten pelien grafiikat toteutetaan yleensä käyttäen perinteisiä kuvankäsittelyohjelmia.

Kolmiulotteisissa peleissä päätyökalut ovat 3D-mallinnusohjelmat. 3D-mallinnusohjelman avulla luodaan kaikki esineet/asiat, joita pelissä käytetään (näistä esineistä ja asioista käytetään nimeä mallit). Tämän jälkeen tavallisella kuvankäsittelyohjelmalla luodaan ns. pinnat näille malleille. Pinnoilla tarkoitetaan kuvaa, joka piirretään mallin päälle, kuten vaatteet ovat ihmisen päällä. Kuitenkin pinta liitetään malleihin vasta ohjelmallisesti. Tällä tavalla pystytään luomaan laajoja kenttiä, joita on mahdollista katsoa mistä tahansa suunnasta siten, että ne näyttävät mahdollisimman aidoilta.

Animaatio kaksiulotteisissa peleissä

Aikaisemmin mainittiinkin jo, että animaatio toteutetaan piirtämällä hieman erilaisia kuvia peräkkäin mahdollisimman nopeasti, jotta saavutetaan efekti liikkuvasta kuvasta. Hyvänä esimerkkinä tästä on lapsena käytetty animaation piirtotapa. Kun tavallisen vihkon sivujen alareunaan piirtää tikku-ukkoja, jotka ovat hieman eri asennossa joka kuvassa ja tämän jälkeen selaa vihkoa nopeasti näyttää aivan siltä kuin tikku-ukko juoksisi. Kaksiulotteisissa peleissä animaatio perustuu täysin samaan ilmiöön.

Jos esimerkiksi pelaajan hahmo halutaan animoida kävelemään, piirretään pelaajan hahmosta kolmesta kymmeneen erilaista kuvaa, joissa hahmon jalat ovat hieman eri asennoissa. Tämän jälkeen kun hahmo halutaan saada kävelemään ruudulla, aletaan piirtää ruudulle yksi näistä kuvista kerrallaan nopeassa tahdissa. Vielä kun taustaa liikutetaan hahmon takana sopivalla vauhdilla, saadaan vaikutelma kävelemisestä.

Piirtojärjestys

Kolmiulotteisissa peleissä ei ohjelmoijan tarvitse välittää piirtojärjestyksestä. Piirtojärjestyksellä tarkoitetaan sitä, mikä esine ruudulla näkyvistä esineistä on päällimmäisenä. Kuitenkin kaksiulotteisissa peleissä ohjelmoijan pitää tehdä päätös siitä, miten esineet piirretään ruudulle. Piirtojärjestys määräytyy sen mukaan, että ensimmäisenä piirretään taka-alalle jäävät asiat ja sitten siirrytään tavallaan lähemmäksi katsojaa ja viimeisenä piirretäänkin ruudulla pintaan jäävät asiat. Kolmiulotteisissa peleissä tämä yleensä on toteutettu siten, että esineille annetaan arvo, joka ilmoittaa esineen ”syvyyden” pelialueella.

Ongelmia piirtojärjestyksessä saattaa tulla kaksiulotteisissa peleissä siinä tapauksessa, että pelaaja pystyy muuttamaan sijaintiansa myös syvyyssuunnassa. Tällaisessa tilanteessa kannattaa harkita vaihtoehtoa, jossa kaikille pelissä oleville esineille määritellään myös syvyysarvo. Tämän jälkeen verrataan pelaajan syvyysarvoa ruudulla näkyvien esineiden syvyysarvoon ja tähän perustuen tehdään päätös piirtojärjestyksestä.

Tilesetti

Käsitteelle tileset (engl. Tileset) ei ole järkevää suomenkielistä vastinetta. Kenttäeditoreja käsittelevästä kappaleesta saatiin kuva siitä, että tavallisesti kaksiulotteisissa peleissä kentät muodostuvat pienemmistä palasista, joita toisiinsa yhdistelemällä kentät rakennetaan. Tileset tarkoittaa kuvaa, joka koostuu pelissä olevista erilaisista pikkukuvista. Tileset on kuvana samanlainen kuin kuviossa 8 nähty fonttikuva. Erona fonttikuvaan on se, että kirjainten ja numeroiden sijasta pienissä kuvissa on pelissä esiintyviä objekteja.

Kenttä piirretään siten, että tileset-kuva piirretään ruudulle yhtä monta kertaa, kuin mitä kentässä tarvitaan yksittäisiä tilejä (eli yksittäisiä taustaobjekteja). Jos pelissä käytetään vaikkapa 20:tä kuvaa vaakasuunnassa ja 10:tä kuvaa pystysuunnassa tarkoittaa tämä, että ruudulle piirretään tilekuva 20*10 kertaa, eli 200 kertaa. Tileset-kuva piirretään yleensä sellaisenaan ruudulle, mutta kuva rajataan siten, että se käsittää vain sen alueen, johon yksittäinen kuva halutaan piirtää. Tämä saattaa olla koneelle erittäin työläs tehtävä, varsinkin jos kaikki kuvat joudutaan piirtämään uudestaan jokaisella uudelleenpiirtokerralla.

4.13 Törmäysmallinnus

Törmäysmallinnus saattaa olla pelistä riippuen kaikkein vaikein osuus toteuttaa. Törmäysmallinnuksella tarkoitetaan sitä, ettei pelaajan hahmo voi juosta esimerkiksi tiiliseinän läpi (jos se ei pelisuunnittelun mukaan ole tarkoitus). Törmäysmallinnus vaikeutuu huomattavasti, kun siirrytään kaksiulotteisesta maailmasta kolmiulotteiseen maailmaan. Tutkintotyötä varten toteuttamassani CyberPong pelissä jouduin toteuttamaan yksinkertaisen törmäysmallinnuksen mailan ja pallon välille. Tämä oli vielä kohtalaisen suoraviivainen tehtävä, vaikka tämäkin tuotti itselleni erittäin paljon päänvaivaa. Käsittelen törmäysmallinnuksen toteuttamista CyberPong pelissä myöhemmin luvussa 6.3.

Törmäysmallinnuksesta on kirjoitettu useita kirjoja, ja sen toteuttamisen helpottamiseksi internetistä on löydettävissä hyvinkin pitkälle suunniteltuja algoritmeja. Näitä molempia lähteitä kannattaa hyödyntää mahdollisimman paljon. Mielestäni törmäysmallinnuksessa on tärkeätä lähteä siitä, että pyrkii aluksi itse toteuttamaan alkeellisen törmäysmallinnuksen ennen, kuin lähtee kopioimaan jotain valmiiksi loppuun asti hiottua algoritmiä. Tällä tavalla oppii ymmärtämään törmäysmallinnuksessa esiintyviä ongelmia ja pystyy muokkaamaan valmiita törmäysmallinnusalgoritmeja paremmin omaan peliin sopiviksi. Kolmiulotteista törmäysmallinnusta käsittelevä Kurt Millerin artikkeli on ehdottomasti lukemisen arvoinen (Miller 2000). Metanet Softwaren artikkeli ”Collision detection and Response” käsittelee puolestaan törmäysmallinnusta kaksiulotteisessa maailmassa (Metanet Software).

5 Mobiilipeliohjelmointi

Mobiilipelien ohjelmointi on melko samanlaista kuin tietokonepelien ohjelmointi. Suurin osa asioista on vain tavallaan pienemmässä mittakaavassa. Mobiilipeliä suunniteltaessa suunnittelijan/suunnittelijoiden tulee olla koko ajan tietoisia mobiilipelejä koskevista rajoitteista. Muistin määrä, prosessorin teho, näytön koko ja syötelaitteet vaihtelevat eri laitteissa. Tässä luvussa käsitellään pääasiat, joita tulisi ottaa huomioon mobiilipeliä toteutettaessa.

Seuraavassa luvussa käsitellään tarkemmin mobiilipelin toteutuksessa ilmenneitä asioita. Toteutin mobiilipelin käyttäen Java ohjelmointikieltä. Tästä syystä tämänkin luvun alaluvut perustuvat Java ohjelmointikieleen mobiilimaailmassa.

5.1 Ohjelmointikielet

Mobiilipelien ohjelmoinnissa käytetään eri ohjelmointikieliä kuin tietokonepelien ohjelmoinnissa. Vaihtoehtoina mobiilipelien ohjelmointikieliksi ovat Java ohjelmointikielestä kevennetty versio *J2ME* (Java 2 Mobile Edition), *Symbian* sekä *Brew*.

Java (*J2ME*)

J2ME on Euroopassa mobiilipelien kehityksessä yleisessä käytössä ja suurin osa Euroopassa myytävistä kaupallisista peleistä onkin toteutettu käyttämällä *J2ME:tä*. *J2ME:n* ongelmana mobiilipelien suhteen on kielen hitaus. Java on tulkattava kieli, eli kaikki lähdekoodi käytetään tulkin läpi ennen varsinaista lähdekoodien suoritusta. Tämä vaikuttaa suoraan siihen, että Javalla toteutetut ohjelmat/pelit eivät voi koskaan olla yhtä nopeita kuin muilla kielillä toteutetut pelit. Tämän eron edellytys kuitenkin on, että ohjelma/peli on toteutettu samalla tavalla molemmilla vertailtavilla kielillä.

Suurin osa tällä hetkellä markkinoilla olevista matkapuhelinmalleista tukee Java sovelluksia. Puhelimen mainospuheissa yleensä mainitaankin mahdollinen Java-tuki, jos sellainen puhelimessa on. *J2ME:n* kattavuus matkapuhelinmarkkinoilla onkin huomattavasti parempi verrattuna *Symbianiin* ja *Brewiin*.

Joissakin Java puhelimissa on sisäänrakennettuna niin sanottu 3D-rajapinta. Tämä rajapinta mahdollistaa kolmiulotteisten pelien ohjelmoinnin myös matkapuhelimissa. Ikävä kyllä suurin osa Java-puhelimista on vielä tällä hetkellä liian hitaita kolmiulotteisten pelien pyörittämiseen.

Symbian

SymbianOS on alkujaan matkapuhelinvalmistajien kehittämä mobiililaitteita varten julkaistu käyttöjärjestelmä (Symbian). Nykyisin *Symbiania* tukevia matkapuhelimia on markkinoilla jo jonkin verran, vaikka huomattavasti Javaa tukevia puhelimia vähemmän.

Symbian on ohjelmointikielenä läheistä sukua C++:lle. *Symbian* on mielestäni huomattavasti hankalampi ohjelmointikieli kuin J2ME. *Symbian* pakottaa ohjelmoijan itse huolehtimaan muistinhallinnasta. *Symbian* ei ole tulkattava kieli ja se onkin tästä syystä huomattavasti J2ME:tä nopeampi.

Symbian puhelimissa pystytään ohjelmointikielen paremman tehokkuuden vuoksi ohjelmoimaan myös kolmiulotteisia pelejä. Toisaalta tämä johtuu myös siitä, että suurin osa *Symbian* puhelimista on paremman tason matkapuhelimia.

Brew

Brew on Qualcomm:in kokonaisvaltainen käyttöjärjestelmä matkapuhelimiin. *Brew* puhelimilla pystytään toteuttamaan myös kolmiulotteisia pelejä ohjelmointikielen tehokkuuden ansiosta. *Brew* muistuttaa monessa mielessä *Symbiania*. Kumpikin on niin matkapuhelimen käyttöjärjestelmä kuin samalla ohjelmointikielikin. Lisäksi molempien ohjelmointikielten kirjoitusasu muistuttaa C++-kieltä.

Brew-puhelimia ei vielä tällä hetkellä myydä Suomessa ja Euroopassakin *Brew* puhelinten tilanne on vielä heikko. Amerikassa *Brew* on kuitenkin huomattavasti suositumpi alusta, vaikka *Brew*-puhelimia myydään siellä huomattavasti Java-puhelimia vähemmän. Vuonna 2004 Amerikassa Java-puhelimia myytiin 39 miljoonaa kappaletta kun *Brew*-puhelimia myytiin 23 miljoonaa kappaletta (ResearchAndMarkets 2004).

5.2 Määritykset

Java puhelimissa käytetään kahta tärkeätä määritystä. Näiden määritysten tarkoituksena on yhtenäistää mobiililaitteiden toimintaa ja samalla ohjelmoitavuutta.

Mobile Information Device Profile eli *MIDP*

MIDP on yleisimmin käytettyjä käsitteitä mobiilipelien ja sovellusten ohjelmoinnissa. *MIDP* on Sun Microsystems:in tekemä määritys mobiililaitteille. Määritys määrittelee miten sovellusten tulee toimia mobiililaitteissa. Periaatteessa tämä tarkoittaa sitä, että *MIDP* määritys määrittelee funktiot ja luokat, jotka mobiililaitteen tulee toteuttaa. Tällöin ohjelmoija voi tehdä funktiokutsut ja tietää, että mobiililaite toteuttaa ne ainakin jollakin tavalla.

MIDP määrityksestä on ilmestynyt kaksi versiota ja kolmas on parhaillaan tekeillä. *MIDP* 1.0 on ensimmäinen määritysversio, joka valmistui vuonna 2000. Tällä hetkellä *MIDP* 1.0 määrityksen mukaisia matkapuhelimia on kaupoissa todella suuri määrä. *MIDP* 1.0 määrityksen tärkeimpiä etuja mobiilipelien kehitykseen ovat (Mobile Information Device Profile 2000):

- RecordStore – Mahdollistaa tietojen pysyvän tallentamisen
- Graafiset käyttöliittymäkomponentit
- Mobiililaitteen näppäimistön lukumahdollisuus
- Softkeyt – Tarkoittaa ruudun alareunassa olevien nappien hyödyntämistä
- Canvas – Mahdollistaa piirtämisen ilman valmiita komponentteja
- Erilaiset ennalta määritetyt eri kokoiset fontit
- Tekstin ruudulle piirtämisfunktiot

Vuonna 2002 Sun Microsystems julkaisi toisen version *MIDP* määrityksestään, josta käytetään nimeä *MIDP* 2.0. Versio 2.0 on huomattavasti versiota 1.0 spesifisempi. Version 2.0 toteuttavia matkapuhelimia alkaa olla tällä hetkellä markkinoilla jo melko paljon. *MIDP* 2.0 version tuomat tärkeimmät ominaisuudet pelien kannalta ovat (Knudsen 2002):

- GameCanvas – Pelejä varten suunniteltu käyttöliittymätyyppi
- Äänikirjasto, jonka avulla on mahdollista soittaa .wav ääniä puhelimessa
- Uusia käyttöliittymäkomponentteja
- Läpinäkyvyys

Connected Limited Device Configuration eli *CLDC*

CLDC määrittelee perusohjelmointirajapinnat ja Java virtuaalikoneen. Yhdessä *CLDC* ja *MIDP* muodostavat pohjan Java sovelluksille. *CLDC* määrityksestä on myös ilmestynyt kaksi eri versiota, 1.0 ja 1.1. Versio 1.0 on pohjana tällä hetkellä suurimmassa osassa Java puhelimia mutta version 1.1 toteuttavia matkapuhelimia on todella vähän markkinoilla. Uudempi versio 1.1 määrittelee sen, että matkapuhelimessa tulee olla kaksi uutta luokkaa. Uudet luokat ovat Float ja Double. Tämä tarkoittaa sitä, että vasta versiossa 1.1 on mahdollista käyttää suoraan liukulukuja.

5.3 Laitekanta

Mobiilisovellusta toteutettaessa on otettava huomioon mobiililaitteiden keskinäiset erot. Tämä on otettava huomioon heti suunnitteluvaiheesta lähtien. Kunnollisella esisuunnittelulla pystytään peli toteuttamaan suureen osaan erilaisista matkapuhelimista. Onkin hyvä tapa heti peliprojektia käynnistettäessä tehdä suunnitelma kohdelaitteista tai ainakin suunnitelma siitä, että minkä tasoisille puhelimille peliä yritetään toteuttaa.

Referenssitoteutuksella tarkoitetaan sitä, että pelin toteuttamiseen valitaan yhdestä viiteen erilaista matkapuhelinta. Tämän jälkeen näitä valittu-

ja puhelimia käytetään pelin kehitysaikana testaamiseen. Tavallisesti matkapuhelimiksi kannattaa valita hyvin erilaisia puhelimia, jotta saadaan jo toteuttamisvaiheessa selvyys siitä, miten peli kääntyy erilaisille laitteille toimivaksi.

Pelin kehittämisvaiheessa ei vielä kannata lähteä kääntämään peliä kovin monelle laitteelle. Tämä johtuu siitä, että peli tulee todennäköisesti muuttumaan vielä projektin loppuvaiheessa niin paljon, ettei ennenaikaisesta kääntämisestä saada vastaavaa hyötyä. Heti kun projekti on valmis ja testattu voidaan pelin kääntäminen kaikille laitteille aloittaa.

5.4 Maksimi tiedostonkoko

Mobiililaitteissa, erityisesti matkapuhelimissa on usein rajoitetusti tallennuskapasiteettia. Tämä näkyy suurimmassa osassa puhelimista siten, että Java sovelluspakkauksen koko on rajoitettu. Pienimmillään sovelluspakkauksen maksimikoko on rajoitettu noin 64 kilotavuun. Oman kokemukseni mukaan useimmissa puhelinmalleissa kokorajoitus on suhteellinen puhelimen näytönkoon kanssa: yleensä suuri näyttöisissä puhelimissa on myös paljon muistia ja toisaalta taas pieni näyttöisissä puhelimissa on vain vähän muistia.

Kokorajoitusten skaalana voidaan pitää 64 kilotavua alarajana ja ylärajana 24 megatavua (24 000 kilotavua). Matkapuhelinmalleja, joiden yläraja on niinkin korkealla kuin 24 megatavussa, on erittäin vähän. Alarajan omaavia matkapuhelinmalleja on huomattavasti enemmän.

Obfuskointi

Obfuskoinnilla tarkoitetaan lähdekooditiedostojen jälkikäsittelyä. Riippuu käytettävästä obfuskaattorista, mitä kaikkea lähdekooditiedostoille tavallisesti tehdään. Yleisimmät lähdekoodeille tehtävät toimenpiteet ovat luokkien, metodien ja muuttujien uudelleennimeäminen sekä käyttämättömien luokkien, metodien ja muuttujien poistaminen.

Obfuskoinnista saavutettavat hyödyt

Obfuskoimalla valmiit luokkatiedostot saavutetaan monia hyötyjä. Yleensä mobiilisovelluksia obfuskoitaessa on päätavoitteena lopullisen tiedostokoon pieneneminen. Normaaleja mobiilipelin luokkatiedostoja obfuskoitaessa saavutetaan jopa 50 % tilasäästö. Tilasäästö saavutetaan pääasiassa luokkien, metodien ja muuttujien uudelleennimeämisestä. Esimerkiksi ohjelmoijan nimeämä funktio ”firePrimaryWeapon(int roundsToShoot)” muuttuisi obfuskoitaessa muotoon ”ax(int ay)”. Lisäksi tilaa säästyy, koska obfuskointi poistaa kaikki käyttämättömät luokat, metodit ja muuttujat luokkatiedostoista.

Kun kaikki ohjelmoijan käyttämät nimet muutetaan obfuskoinnissa yksinkertaisemmiksi, ja lisäksi luokkatiedostoista poistetaan kaikki käyttämätön tavara, saadaan sovellukseen lisää suorituskykyä. Kuitenkaan suo-

rituskyvyn kasvu ei ole yleensä mitenkään suuri, vaan kyseessä on enemmänkin toimivan sovelluksen pieni tehostaminen.

Obfuskointi lisää myös obfuskoitujen tiedostojen turvallisuutta. Sovelluksen luokkatiedostoja on huomattavasti vaikeampi purkaa, kun mitkään luokkien, metodien tai muuttujien nimet eivät ole selväkielisiä. Kun obfuskointi suoritetaan, tallennetaan obfuskoidut lähdekooditiedostot väliaikaishakemistoon, näitä tiedostoja tutkimalla ei itsekään koodin luoneena voi enää olla varma, mitä koodissa todella tapahtuu.

Näen myös yhtenä obfuskoinnin suurista eduista sen, että ohjelmoijan ei tarvitse missään vaiheessa koodia luodessaan rajoittaa minkään luokan, metodin tai muuttujan nimeä lähdekooditiedoston tiedostokoon kasvamisen pelossa. Myöskään ohjelmoijan ei tarvitse käyttää mitään lyhenteitä, joita seuraava ohjelmoija ei ymmärtäisi.

Obfuskoinnin aiheuttamat haitat

Mitään hyvää ei yleensä saavuteta ilman joitakin haittoja. Obfuskoinnin ollessa kyseessä haitat jäävät kuitenkin todella pieniksi. Ainoina selvinä haittoina obfuskoinnista seuraa joillakin laitteilla yhteensopivuusongelmia. Nämä yhteensopivuusongelmat saadaan kierrettyä sillä, ettei käytetä obfuskoitaessa kaikista raskainta obfuskointia (aggressive obfuscation, Proguard obfuskaattorissa). Raskainta obfuskointia käytettäessä metodeille ja muuttujille annetaan samoja nimiä, jolloin saavutetaan vielä jonkin verran etua tiedostokoon suhteen. Kuitenkaan kaikki päätelaitteet eivät hyväksy tällaista koodia, sillä lähdekoodi ei enää tässä tapauksessa ole Java-standardin mukaista.

Tätä ongelmaa ei käsittääkseni esiinny kaikissa obfuskaattoreissa. Toisena ongelmana voidaan pitää obfuskointiin kuluvaa aikaa tai sen käytön hankaluutta. Yleensä obfuskaattoria käytetään suoraan kääntöprosessiin integroituna, jolloin obfuskoinnin osuus koko sovellukseen kuluvasta kääntämisestä on nykykoneilla noin viiden sekunnin luokkaa.

Obfuskaattorit

Obfuskaattoreita on useita. Osa obfuskaattoreista on niin sanotun GPL-lisenssin alaisena, joka tarkoittaa sitä, että tämän lisenssin alaisuuteen kuuluvien obfukaattoreiden käyttö on maksutonta. Itse olen käyttänyt pääasiassa GPL-lisenssin alaista Proguard obfuskaattoria (http://proguard.sourceforge.net/), omissa töissäni se on toiminut todella hyvin. (ProGuard).

Toinen hyvin toimiva obfuskaattori on RetroGuard (http://www.retrologic.com/), joka kuuluu puolestaan GNU-lisenssin alaisuuteen. Erona näiden lisenssien välillä on se, ettei GNU-lisenssin alaisia sovelluksia saa käyttää kaupallisiin tarkoituksiin ilman erillisen luvan ostamista. (RetroLogic).

Obfuskaattoreita on tarjolla monelta valmistajalta ja kannattaakin katsoa valmistajien kotisivuilta, mitä kaikkia ominaisuuksia heidän tuotteistaan löytyy. Obfuskaattorien mukaan on yleensä pakattu myös muita työkaluja lähdekoodien käsittelyyn.

5.5 Skaalautuvuus

Yksi valmistuvan mobiilipelin tärkeimmistä ominaisuuksista on skaalautuvuus eri laitteiden ja laitealustojen välillä. Siinä vaiheessa kun mobiilipelistä saadaan referenssiversiot täysin valmiiksi ja testattua, alkaa mobiilipelin kääntäminen muille laitteille. Tavallisesti tämä tarkoittaa sitä, että laitteista yhdistetään samantyylisiä laitteita ryhmäksi ja tämän jälkeen jokaiselle laiteryhmälle toteutetaan oma versionsa pelistä.

Pelin kääntäminen toimivaksi kaikille laitteille voi olla pahimmillaan lähes yhtä iso urakka kuin varsinaisen pelin toteuttamisprosessikin. Pelin kääntämistä muille laitteille voidaan helpottaa ja nopeuttaa huomioimalla tuleva kääntöprosessi niin suunnittelu kuin toteutusvaiheissakin. Ilman kääntöprosessin huomioimista voidaan mahdollisesti törmätä lukuisiin erilaisiin ongelmiin.

Toteutuksellisista ongelmista yleisimpiä ovat pelin pyörimisnopeuteen liittyvät ongelmat. Näiden ongelmien ratkaisemiseksi ei ole annettavissa mitään yksittäistä ohjetta, joka voisi auttaa yleisluonteisesti kaikkien laitteiden tai pelien kohdalla. Pyörimisongelmat liittyvät yleensä pelin piirtotehojen tai prosessointitehon loppumiseen. Hyvällä pelisuunnittelulla voidaan toteuttaa peliin toiminnallisuuksia, joita on mahdollista jättää joistakin versioista pois. Näin pystytään helposti vaikuttamaan pelin piirto- ja prosessointitehojen tarpeeseen. Tällaisia asioita voisi olla esimerkiksi raskaampien kenttien tiputtaminen pois pelistä tai niiden korvaaminen kevyemmillä. Matkapuhelinten nopeuksista (ja ongelmista) on kerätty useita tietokantoja internetiin. Yksi parhaista tietokannoista alustavaan nopeudenselvittämiseen on JBenchmark (JBenchmark). Kokemukseni mukaan JBenchmarkin ilmoittamiin asioihin kannattaa kuitenkin suhtautua pienellä varauksella.

Usein peleissä on myös ulkoasun skaalautuvuusongelmia. Matkapuhelinten näyttökoossa on iso ero pienimmän ja suurimman näytön väliltä. Tästä syystä jokaisessa piirtofunktiossa pitäisi pystyä ottamaan huomioon molemmat ääripäät näyttöjen koossa. Tämä ei ole kaikissa tapauksissa helppoa ja usein pelin kääntämisvaiheessa joudutaan tekemään vielä korjauksia piirtofunktioihin. Kannattaa ottaa myös huomioon, että on harvoin järkevää käyttää samaa tilesettiä suuri ja pieni näyttöisissä matkapuhelimissa. Onkin järkevää toteuttaa vähintään kaksi erilaista sarjaa kaikista kuvista eri laitteita varten. Lisäksi tekstin piirtofunktiot kannattaa ohjelmoida erittäin tarkasti, sillä niiden yhteydessä ilmenee usein ongel-

mia. Yleisimpiä tekstin piirtofunktioissa ilmeneviä ongelmia ovat rivitys sekä tekstin rajaukseen liittyvät ongelmat.

Äänet ja musiikki vaativat kääntämisvaiheessa jonkin verran työtä. Tämä johtuu siitä, että matkapuhelimissa on useita erilaisia tapoja toteuttaa äänet sekä musiikki. Musiikkien ja äänien toteuttaminen ei pitäisi tuottaa mitään ongelmia, jos toteutusvaiheessa äänitoteutus on luotu omaan luokkaansa. Tällä tavalla voidaan esiprosessoimalla tai manuaalisesti tiedostoja muuttamalla vaihtaa laitteessa käytettävää äänitoteutustiedostoa. Erilaisia äänitoteutuksia joudutaan tällä hetkellä tekemään noin kymmenen erilaista, jos pyritään kattamaan kaikki yleisimmät markkinoilla olevat puhelinmallit. Suurin osa näistäkin käyttää samoja äänitiedostoja (eli .mid tiedostoja), mutta varsinainen äänten soittaminen joudutaan toteuttamaan eri tavalla.

5.6 Valmistajien luokkakirjastot

Laitevalmistajat toteuttavat puhelimiinsa usein *MIDP* määritysten ulkopuolisia ohjelmointirajapintoja. Näiden rajapintojen avulla on mahdollista käyttää puhelimista ominaisuuksia, jotka muuten jäisivät saavuttamatta. Mobiilisovellusta kehitettäessä on kuitenkin tärkeätä tietää, mitä laitevalmistajien rajapintoja käyttää ja mitä rajapinnan käyttö tarkoittaa laajemmassa mittakaavassa. Laitevalmistajien rajapintojen käyttö onkin vaarallisinta juuri referenssiversioiden kehityksen yhteydessä.

Yleensä laitevalmistajien omia rajapintoja kannattaa käyttää ainakin erilaisten äänitoteutusten tekemisessä (mm. Motorola, O2 ja Panasonic), sekä erilaisten kokoruututoteutusten tekemisessä (mm. Nokia, Motorola). Kokoruututoteutuksella tarkoitetaan sitä, että näiden rajapintojen avulla saadaan vanhemmissakin puhelimissa puhelimen näyttö kokonaan sovelluksen käyttöön. Kokoruututoteutus tuli mukaan *MIDP* 2.0 määrittelyyn, jossa se saadaankin helposti käyttöön yhdellä funktiokutsulla. Rajapintojen käyttö äänitoteutusten tekemisessä tarkoittaa sitä, että jos haluaa soittaa musiikkia tai ääniä esimerkiksi Motorolan A835 puhelimessa, on pakko käyttää Motorolan omaa "BackgroundMusic"- ohjelmointirajapintaa äänien toteuttamiseen.

Jos sovellus suunnitellaan alusta alkaen toimimaan vain tietyn valmistajan puhelimissa, voidaan valmistajan rajapintojen käyttämisen suhteen olla melko huolettomia. Valmistajien omia rajapintoja ei kuitenkaan välttämättä löydy valmistajan kaikista puhelimista. Tämä voi asettaa rajoitteita sovelluksen toimivuudelle. Tämäkin asia kannattaa ottaa huomioon jo suunnitteluvaiheessa siten, että suunnittelee laitekannan, jolle sovellusta pyrkii tekemään, ja tämän jälkeen selvittää näissä puhelimissa toimivat rajapinnat. Jos sovellus pohjautuu liikaa tietyn laitevalmistajan rajapintaan, voi sovelluksen valmistuessa rajapinnan käytöstä pois ottaminen olla todella työlästä, ja pahimmillaan rajapinnan käytöstä poistaminen voi

muuttaa sovelluksen toimintaa kriittisesti. Onkin järkevämpää yrittää pysytellä *MIDP* määritysten mukaisissa rajapintakutsuissa ainakin siihen saakka, että ensimmäinen versio sovelluksesta on valmis.

5.7 Esiprosessointi

Esiprosessoinnilla tarkoitetaan lähdekooditiedostojen käsittelyä ennen niiden käyttämistä varsinaisessa kääntöprosessissa luokkatiedostojen luomiseksi. Esiprosessointi voidaan oman tulkintani mukaan jakaa kahteen päätyyliin. Ensimmäisessä tyylissä lähdekoodeihin ei ennen prosessointia merkitä mitenkään muokattavia kohtia, vaan esiprosessoinnin suorittava sovellus pyrkii löytämään muokattavat kohdat lähdekoodia tutkimalla, etukäteen määriteltyjä sääntöjä noudattaen. Kappaleessa 5.5 käsiteltiin obfuskointia, joka toimii tämän tyylin mukaisesti.

Toinen tyyli eroaa ensimmäisestä tyylistä siten, että siinä lähdekoodeihin merkitään muokattavat kohdat ennen varsinaista lähdekooditiedostojen prosessointia. Kumpikaan näistä tyyleistä ei sido tapaa, jolla lähdekooditiedostojen muokkaus suoritetaan. Esiprosessointiin voidaan käyttää oman valinnan mukaan lähestulkoon mitä tahansa ohjelmointikieltä. Molemmat tyylit sopivat hyvin mobiilipelien esiprosessointiin. Yleensä molempia tyylejä käytetäänkin rinnan toistensa kanssa.

Jälkimmäinen tyyli tuo huomattavan määrän erilaisia etuja ja helpotuksia mobiilipelien toteuttamiseen. Yhtenä pääetuna, jota tämän tyylisellä esiprosessoinnilla saavutetaan, on esiprosessoidun mobiilipelin skaalautuvuus eri puhelinten ja eri laitevalmistajien välillä, sekä useiden, hieman erilaisten, peliversioiden helpompi hallinnointi.

Kuvitellaan tilanne, jossa pitää toteuttaa peli Nokian 6600 ja Motorolan E1000 puhelinmalleille. Molemmat puhelimet ovat *MIDP* 2.0 yhteensopivia. Nokia 6600 puhelin käyttää perinteistä *MIDP* 2.0 standardiin pohjautuvaa äänitoteutusta, kun taas Motorolan E1000 käyttää Motorolan omaa BackgroundMusic toteutusta. Ongelmana tässä on se, ettei Nokian puhelin soita ollenkaan ääniä, jos äänitoteutus on tehty käyttäen Motorolan äänitoteutusta, ja myöskään Motorolan puhelin ei soita ääniä jos äänitoteutus on *MIDP* 2.0 pohjainen. Tällaisen ongelman ratkaiseminen ilman esiprosessointia olisi melko hankalaa. Ainoa mahdollinen tapa olisi luoda kaksi erillistä äänitoteutustiedostoa, ja ennen peliversion kääntämistä pitäisi oikea äänitiedosto siirtää lähdekooditiedostojen joukkoon. Tämä tyyli on toimiva, jos muuttuvia kohtia versioiden välillä on pelissä yksi tai kaksi, mutta laajemmissa peleissä saattaa muuttuvia kohtia olla kymmeniä, jolloin tämä tyyli osoittautuu mahdottomaksi. Pahimmillaan tämä muutos tarkoittaisi kahden eri peliversion jatkuvaa ylläpitämistä. Kun toisesta versiosta löytyy jokin ongelma, pitää se muistaa korjata molempiin versioihin. Projektinhallinnallisesti sekä versiohallinnallisesti tämä tyyli ei olekaan toimiva.

Parhaiten kuvan siitä, että miten esiprosessoinnista hyödytään mobiilisovellusten toteuttamisessa, saadaan tutkimalla edellä mainittua ongelmaa. Yksinkertaisin ratkaisu edellä mainittuun ongelmaan olisi se, että olisi mahdollista tehdä kaksi erillistä äänitoteutusta ja kääntämisen yhteydessä automaattisesti liitettäisiin oikea toteutus lähdekooditiedostojen joukkoon. Juuri näin esiprosessointi tällaisessa tilanteessa toimii.

5.8 Grafiikat

Mobiililaitteiden näytöt ja näytönohjaimet vaihtelevat paljon laitteittain. Suurimmassa osassa mobiililaitteista näytön värierottelukyky on huomattavasti tietokoneen näyttöjä heikompi. Tästä syystä onkin hyvä suunnitella mobiilipelien värimaailma siten, että kuvissa on mahdollisimman suuria värieroja. Tällä tavalla saadaan kadotettua mobiililaitteiden värierottelukyvyn heikkoutta.

Näytönohjaimien tasoerot näkyvät mobiililaitteissa siten, että jotkut puhelimet pystyvät piirtämään erittäin nopeasti raskastakin grafiikkaa, kun taas jotkut puhelimet eivät tunnu selviävän edes yksinkertaisista viivojen piirtämisistä tarpeeksi nopeasti. Kokemukseni mukaan mitä edullisempi puhelin on kysymyksessä, sitä heikompi näytönohjain puhelimessa on. Tietysti poikkeuksiakin tähän sääntöön varmasti löytyy.

Javalla toteutetuissa mobiilipeleissä käytetään yleensä .png kuvia. Joillakin puhelimilla on erikoissääntöjä siitä, että millaisia .png kuvia puhelin hyväksyy. Näitä erikoissääntöjä esiintyy vain muutamissa puhelimissa ja jos tällaiseen erikoissääntö tulee kohdalle ohjelmoitaessa sen huomaa peliä testatessa. Esimerkkinä tällaisesta erikoissäännöstä on joillakin puhelimilla esiintyvä kuvakoon rajoitus, eli puhelin ei yksinkertaisesti suostu lataamaan liian leveätä kuvaa.

Läpinäkyvyys

Joissakin mobiililaitteissa ei ole tukea läpinäkyvyydelle ollenkaan, joka voi aiheuttaa joissakin peleissä erittäin suuria ongelmia. *MIDP* 1.0 määrityksen mukaisesti puhelimen ei tarvitse tukea läpinäkyviä pikseleitä ollenkaan. Onneksemme suurin osa mobiililaitteiden valmistajista on toteuttanut tuen läpinäkyvyydelle vaikka se ei määrityksen piiriin kuulukaan.

Suurin osa mobiililaitteista tukee läpinäkyvyyttä jos sitä hyödynnetään suoraan kuvatiedostoista. Tämä tarkoittaa käytännössä sitä, että ohjelmallisesti ei ole mahdollista luoda läpinäkyviä pikseleitä, mutta jos kuvatiedostossa on määritelty läpinäkyviä pikseleitä osaa puhelin näyttää ne oikein.

6 ”CyberPong”- mobiilipelin toteuttaminen

6.1 Suunnittelu

6.1.1 Peli-idea, tarina, pelin toiminnallisuus ja näiden kehittyminen

Tavallisesti peli-idean ja tarinan kehittyminen (ja kehittäminen) on pitkä prosessi. Nyt tähän vaiheeseen ei ollut mahdollista käyttää riittävästi aikaa, joten peli-ideaksi valittiinkin yksinkertainen pallopeli pienillä lisämausteilla. Ensimmäisenä suunniteltiin peli-idea ja lähdettiin rakentamaan tarinaa pelin ympärille. Liite 1 on viimeisin versio alkuperäisestä peli-idean, tarinan ja toiminnallisuuden suunnitteludokumentista. Dokumenttia ei ole tehty minkään valmiin mallin tai ohjeen mukaisesti. Jos kyseessä olisi laajamittaisempi peliprojekti, kannattaisi dokumentti tehdä huomattavasti tarkemmin. Tässä tapauksessa se kuitenkin palveli tarkoitustaan, sillä peliä oli tekemässä vain kaksi henkilöä.

Peli-idea

Pelin tarkoituksena on ratkaista maailman tulevaisuus. Tämä tapahtuu siten, että maksimissaan neljä pelaajaa pelaa vastakkain *PONG* tyyppistä ylhäältäpäin kuvattua pallopeliä. Jokaiselle pelaajalle arvotaan ruudulta reuna, jota hänen tulee suojella. Suojeleminen tapahtuu siten, että pelaaja liikuttaa omaa mailaansa niin, ettei pallo pääse hänen reunaltaan ulos.

Pelissä on neljä erilaista klaania. Jokaisella klaanilla on oma kenttänsä sekä erikoisase mailassa. Jokaisessa kentässä on joko lisäksi ase tai jonkinlainen apu saman klaanin pelaajaa varten. Esimerkiksi tiedekentässä on liikkuva magneettikenttä, joka vääristää pallon liikerataa. Magneetti ei kuitenkaan koskaan liiku tiedeklaanin pelaajan ruudun alueelle.

Peli etenee siten, että pelaajat valitsevat itselleen klaanin jolla pelaavat peliä. Tämän jälkeen peliä pelataan neljä erää, yksi erä jokaisen klaanin kentässä. Pelin aikana jokaisesta osumasta palloon pelaaja saa yhden pisteen. Neljän erän jälkeen pelaaja, jolla on eniten pisteitä, ilmoitetaan pelin voittajaksi.

6.1.2 Luokkasuunnittelu

Suunnitelma käytettävistä luokista tehtiin heti sen jälkeen kun pelin toiminnallisuus oli selkeytynyt. Luokkasuunnitelma on vain yleisluonteinen suunnitelma siitä, minkälaisia luokkia pelissä käytettäisiin ja miten nämä luokat ovat yhteydessä toisiinsa.

Suuri osa luokkasuunnittelusta tapahtui vasta pelin toteuttamisen yhteydessä. Tässä vaiheessa oli nähtävissä, miten luokkia ja niiden välisiä suhteita kannattaisi toteuttaa. Toteutuksen aikana jouduttiin joitakin kertoja tekemään isojakin muutoksia yleiseen luokkarakenteeseen ennen kuin löydettiin ratkaisu luokkien toiminnalle.

Eniten suunnittelua luokkien toiminnasta vaati aseiden toiminta ja niiden tuominen peliin. Jokainen ase, niin maila-aseet kuin kenttäaseetkin, noudattavat pitkälti samaa kaavaa, mutta ovat kuitenkin merkittävästi erilaisia. Muutaman erilaisen työversion jälkeen päädyin ratkaisuun, jossa toteutin yhteisen rajapinnan kaikille aseille. Tämä rajapinta käsittää funktiot mm. aseen laukaisemiseen ja aseen ajastamiseen.

```
Public Interface Weapon{
	public void fire();
	public void paint(Graphics g);
	public void wait();
	...
}

class ArmyBatWeapon implements Weapon{
	public void fire(){
		...
	}

	public void wait(int milliSeconds){
		...
	}

	public void paint(Graphics g){
		...
	}
}

public class Player{
	Weapon weapon;
	int clan;
	public Player(int clan){
		this.clan = clan;
		if(this.clan == ARMY){
			weapon = new ArmyBatWeapon();
		}
		else if(this.clan == SCIENCE){
			weapon = new ElectricityGun();
		}
	}

	public void fire(){
		weapon.fire();
	}
}
```

Kuvio 9 Koodiesimerkki aseiden toteutuksesta CyberPong-pelissä

Kuviosta 9 nähdään, miten pelissä onnistuttiin välttämään erilaiset aseen funktiokutsut. Nyt Player-luokan fire-funktio pystyttiin toteuttamaan ilman, että jouduttiin tekemään vertailuja siitä, mikä ase on kyseessä. Ilman kyseistä toteutustyyliä olisi todella vaikea toteuttaa aseita pelaajille siten, ettei jouduttaisi luomaan turhia olioita tai muuten tekemään hankalia luokkarakenteita.

6.1.3 Suunnitelma kohdelaitteista ja käytettävistä kielistä

Kohdelaitteet rajattiin käsittämään mahdollisuuksien mukaan kaikki Java yhteensopivat puhelimet. Käytännössä tämä tarkoittaa sitä, että kaikki asiat jouduttiin toteuttamaan heikoimman puhelimen mukaan. Tämä ei kuitenkaan tarkoittanut sitä, että olisin vähentänyt pelistä toiminnallisuuksia tai muuten yrittänyt mahduttaa peliä pienimpään tiedostokokorajoitukseen. Toteutin toiminnallisuudet siten, että ne voidaan asettaa profiilitiedostosta pois päältä. Näin jokaiselle laitteelle pystytään määrittämään erikseen toiminnallisuudet, joita pelissä käytetään.

Referenssilaitteeksi valittiin Series60 alusta *MIDP* 1.0 standardia mukaillen. Tällä tavalla peliä saatiin testattua Series60 alustalle suunniteltuilla emulaattoreilla sekä omalla Nokia N-Gage QD matkapuhelimellani. Mielestäni referenssilaitteeksi kannattaakin valita erittäin yleisluonteinen laite. Jos valitsee hankalan tai harvinaisen alustan, joutuu ohjelmoija tekemään ylimääräistä työtä matkapuhelimen erikoisuuksien ratkaisemiseksi.

Käytettävän kielen suhteen ei ollut tässä tapauksessa mitään epäselvyyttä. Java-ohjelmointikieli on itselleni ennestään tuttu ja pääsin valitsemalla tämän kielen parhaiten opiskelemaan peliohjelmointia, niin ettei minun tarvinnut samalla opiskella uutta ohjelmointikieltä. Lisäksi Java on erittäin yleisesti käytetty mobiilipelien toteuttamiseen ja tämä lisää mielenkiintoa oppia kunnolla ohjelmoimaan yleisesti käytetyllä ohjelmointikielellä.

6.2 Toteutus

6.2.1 Esiprosessointi

Toteutin peliin pohjan esiprosessoinnille siten, että lisäsin niin sanottuun profiilitiedostoon kaikki muuttujien esittelyt, jotka liittyvät matkapuhelinmallien eroihin. Tällaisia muuttujia ovat muun muassa ruudun koon ilmoittavat muuttujat. Tämän järjestelyn avulla profiilitiedostoon on helppo lisätä muutoskohtia kun niille ilmenee tarvetta.

Seuraavana esiprosessointivaiheena olisi toteuttaa järjestelmä, jolla saadaan liitettyä oikeat lähdekoodit oikeisiin projekteihin. Tämän ajattelin toteuttaa siten, että lisään profiilitiedostoon merkintöjä käytettävistä lähdekooditiedostoista. Esimerkiksi äänitoteutustiedoston valinta tehtäisiin profiilissa määritellyn käytettävän tiedoston perusteella.

6.2.2 Sini & Cosini & Tangentti

Useimmissa graafisuuteen perustuvissa peleissä tarvitaan siniä, cosinia ja tangenttia. Kuitenkin vasta *CLDC* 1.1 määritys määrittelee, että nämä tulee löytyä funktioina Math luokasta. Pelissä tarvittiin sini ja cosini arvoja pallon realistisen liikeradan sekä osumakäyttäytymisen laskemiseen. Sini- ja cosini-arvoja käyttämällä on mahdollista toteuttaa esimerkiksi tässä tapauksessa pallolle realistinen liikerata ilman etukäteen määriteltyjä liikeratoja.

En kuitenkaan halunnut rajoittaa pelin toimivuutta vain *CLDC* 1.1 määrityksen toteuttaviin matkapuhelinmalleihin, joten sinistä ja cosinista tehtiin oma toteutus. J2SE:llä toteutettiin apuohjelma, joka tulostaa sini- ja cosini-arvot kahden desimaalin tarkkuudella Java kielen mukaisena taulukkona sinin ja cosinin arvoilla 0:sta 360:een. Arvoja ei voitu kopioida sellaisenaan pelin lähdekoodien joukkoon, koska arvot eivät saisi olla liukulukuina, jotta peli voitaisiin kääntää myös *MIDP* 1.0 määrityksen alaisiin matkapuhelimiin. Muutin apuohjelmaa siten, että se kertoo jokaisen sini- ja cosini-arvon 100:lla, jotta sain muutettua arvot kokonaislukutyyppisiksi. Tämän jälkeen apuohjelman tuloste kopioitiin ja liitettiin ne itse tehtyyn Math luokkaan kahden yksiulotteisen taulukon määrittelyn perään. Tämän jälkeen taulukoille toteutettiin funktiot, jotka palauttavat taulukosta parametria vastaavan arvon. Nämä funktiot nimettiin samoin kuin *CLDC* 1.1 määrityksen mukaiset funktiot on nimetty.

MIDP 1.0 määrityksen mukaisessa Math-luokassa ei ollut mitään tämän pelin kannalta oleellista tai käyttökelpoista funktiota, joten nimeämällä Math-luokka täsmälleen samalla tavoin kuin *MIDP* 1.0 määrityksen mukainen Math-luokka on nimetty, pystyttiin ylimäärittelemään *MIDP* 1.0 määrityksen mukainen luokka. Tällä ylimäärittelyllä saavutettu hyöty on se, että itse tehty Math-luokan toteutus voidaan poistaa sellaisten matkapuhelinmallien sovelluspaketeista, jotka toteuttavat *CLDC* 1.1 määrityksen. Tällaisessa tapauksessa sovellus käyttää suoraan puhelimen omaa Math-luokan toteutusta, joka on huomattavasti nopeampi. Lisäksi tällä saadaan pienennettyä sovelluspaketin kokoa *CLDC* 1.1 puhelimissa.

6.2.3 Pelin tahdistaminen

Pelin tahdistaminen päätettiin toteuttaa usean säikeen avulla. Säikeistyksen toteuttaminen aloitettiin suunnittelemalla säikeiden vastuualueet. Tämän jälkeen tiedettiin jokaisen säikeen tehtävä ja näin pystyttiin nimeämään niiden apumuuttujat. Jokaista säiettä varten toteutettiin oma kokonaislukutyyppinen vakio laiteprofiiliin. Tämän muuttujan arvo vaikuttaa säikeen nukkumisaikaan. Näin jokaisen säikeen suoritusnopeutta pystytään erikseen muuttamaan jokaisessa laitteessa. Tällä saavutetaan se hyöty, että jokaiselle laitteelle pystytään tekemään pelinopeudeltaan so-

piva versio muuttamalla säikeiden suoritusaikoja. Seuraavaksi jokainen säie käsitellään erikseen.

Ensimmäinen säie hallitsee ainoastaan ruudun uudelleenpiirtoa. Tämän säikeen vastuualueena on tehdä piirtokutsu ja tämän jälkeen nukkua määrätty aika. Tämän säikeen avulla pystytään vaikuttamaan niin kutsuttuun *fps*-aikaan. Tämän säikeen nukkumisaika on yksittäisesti suurin pelin prosessointitehon käyttöön vaikuttava muuttuja. Mobiililaitteissa pyritään 15 ruudun sekuntivauhtiin. Tästä syystä nukkumisaikaa on turha säätää pienemmäksi kuin 70 millisekuntia. Tässä tapauksessa, jos yksittäisen piirtofunktion suorittaminen ei kestä pidempään kuin korkeintaan kymmenen millisekuntia, pitäisi ruutu päivittyä noin 12–15 kertaa sekunnissa.

Toisen säikeen tehtävänä on liikuttaa pelaajien mailoja. Tämä säie kutsuu jokaisen neljän mailan liikuttamisfunktiota. Lisäksi jokaisella mailalla on erikseen liikkumisnopeus, jonka arvo merkitsee sitä, montako pikseliä maila liikkuu ruudulla. Tämän säikeen avulla pystytään hallitsemaan kaikkien mailojen päivitysnopeutta, eli liikkumisnopeutta. Lisäksi jokaisen mailan yksittäistä nopeutta voidaan muuttaa muuttamalla muuttujan arvoa, joka vaikuttaa liikutettavien pikseleiden määrään.

Kolmantena säikeenä on pallon liikuttamissäie. Tämän säikeen tehtävänä on liikuttaa palloa sopivin väliajoin. Samoin kuin mailojen liikuttamisessa, pallollakin on erikseen muuttuja liikutettavien pikseleiden määrään. Mailat liikkuvat vain vaaka- tai pystysuunnassa kun taas pallo voi liikkua mihinkä tahansa suuntaan ruudulla. Tästä syystä pallon liikkumismäärän laskeminen ei ole aivan yhtä yksinkertaista. Liikkuminen lasketaan laskemalla hypotenuusan arvo, kun tiedetään pallon liikkuma matka X ja Y akseleilla.

6.2.4 Liukuluvut

Liukuluvut ovat perinteisessä ohjelmoinnissa yleisessä käytössä. Mobiililaitteissa liukuvut tulevat vasta *CLDC* 1.1 määrityksen mukana, eli kattavuus kaikista puhelinmalleista on kirjoittamishetkellä muutaman prosentin luokkaa. Liukulukujen puuttuminen *CLDC* 1.0 määrityksestä johtuu siitä, että varhaisemmissa matkapuhelinmalleissa ei ole laitepuolella tukea liukuluvuille. Tästä syystä *CLDC* määrityksen laativa ryhmä päätti jättää liukuluvut pois ensimmäisestä määrityksestä. Erityisesti mobiilipelien ohjelmoinnissa, mutta myös mobiilisovellustenkin ohjelmoinnissa, liukuluvuttomuus hankaloittaa joitakin asioita. Kuitenkin kaikista liukuluvuttomuuden aiheuttamista ongelmista voi selvitä.

Pelin suunnitteluvaiheessa päätettiin, pyrkiä välttämään liukulukujen käyttöä, jos se vain on suinkin mahdollista. Törmäsin useassa kohdassa ongelmiin tämän päätöksen takia, kuitenkin suurin osa ongelmista oli melko suoraviivaisesti ratkaistavia. Suurin haitta liukuluvuttomuudesta

oli koodin määrän kasvaminen, josta seuraa koodin helppolukuisuuden heikentyminen.

Työläin liukulukujen aiheuttama ongelma tuli esiin pallon liikenopeuden ja liikeratojen toteutuksen yhteydessä. Pallon portaaton liikkuminen tarkoittaa sitä, että pallo pystyy liikkumaan esimerkiksi 0,3 pikseliä vasemmalle kerrallaan (vaikka puhelin pystyy piirtämään vain pikselin tarkkuudella, mutta silti liike näyttää pehmeältä ja tasaiselta). Jouduin muuttamaan pallon toimintaa siten, että pallon X- ja Y-koordinaatit kerrotaan 100:lla, jolloin myös pallon liikenopeus voidaan kertoa 100:lla. Tässä tulee muistaa se, että kuitenkin ruudulle piirtämisen yhteydessä X- ja Y-koordinaatit pitää jakaa sadalla, jottei palloa piirretä ruudun ulkopuolelle.

6.2.5 Törmäysmallinnus

Ensimmäinen vaihe törmäysmallinnuksessa on tutkia pallon liikerata. Aluksi otetaan pallon nykyinen sijainti ja seuraava laskettu sijainti. Tämän jälkeen tutkitaan onko pallon päätepisteessä mitään estettä. Jos törmäystä ei tapahdu, liikutetaan pallo laskettuun kohtaan. Törmäyksen sattuessa lasketaan pallon osumakohta mailaan. Tämän osumakohdan mukaan määritellään suunta, johon pallo lähetetään.

Tässä törmäysmallinnuksessa on muutama vakava ongelma. Suurin ongelma on se, että tällä tavalla laskettuna pallon törmäyksiä yhden silmukan sisällä ei saada tutkittua. Jos pallon nopeus on erittäin suuri, saattaisi pallo huonolla tuurilla mennä mailan läpi, koska pallo saattaisi hypätä tällaisessa tapauksessa esimerkiksi neljä pikseliä kerralla.

Toinen ongelma törmäysmallinnuksessa on se, ettei liikkuvia kohteita huomioida mitenkään. Pallon törmäystä tutkittaessa pitäisi ottaa huomioon myös se, ehtiikö maila liikkumaan pois pallon tieltä ennen törmäystä tai ehtiikö maila tulemaan törmäysalueelle törmäyksen laskemisen jälkeen.

6.3 Lopullinen tuote

Kirjoittaessani tätä tutkintotyöni kirjallista osuutta on vaikea puhua vielä lopullisesta tuotteesta. Tällä hetkellä pelistä onkin valmiina vasta korkeintaan 50 % kokonaisuudesta.

Mitä on valmiina?

Pelimoottori on käytännössä valmis. Kenttien eteneminen, pallon ja mailojen liikkuminen ovat sellaisessa kunnossa, että ne vaativat vain viimeistelyä.

Aseistuksesta pelissä on toteutettu valmiiksi armeijajoukkueen maila-ase. Tämä ase toimii siten, että pelaaja, jonka klaanina on armeijajoukkue, voi kolmoisklikkaamalla nappiansa käyttää klaaninsa erikoisaseen. Erikoisase ampuu kolmeen suuntaan kolme luotia. Nämä luodit lähtevät pelaajan mailasta muiden pelaajien alueita kohti. Jokainen kolmen luodin sarja hajoaa 3 asteen kulmalla muihin saman sarjan luoteihin (Kuvio 10).

Kuvio 10 Emulaattorikuva armeijajoukkueen aseistuksesta Cyber-Pongin työversiosta.

Toteutin myös armeija-kenttään erikoisaseen. Erikoisase toimii siten, että se arpoo oman ilmestymisaikansa 20 ja 40 sekunnin väliltä. Kun ase ilmestyy ruudulle se alkaa ampua luoteja ruudun keskikohdasta myötäpäivään tasaisesti ympäri kenttää siten, että se ei ammu ollenkaan luoteja sille reunalle, jossa armeija-pelaajan maila on. Luoteja on keskimäärin kolme kappaletta koko ajan ruudulla erikoisaseen ollessa toiminnassa.

Molemmilla armeija-aiheisilla erikoisaseilla on sama vaikutus muihin pelaajiin. Luodin osuttua mailaan maila katkaistaan siten, että pelaajalle jää luodin osumiskohdasta riippuen pidempi osa mailaa vielä jäljelle.

Kaikki neljä kenttää toteutettiin niin pitkälle, että grafiikoita päästiin testaamaan kenttiin. Pelin grafiikat eivät ole vielä tällä hetkellä täysin valmiit. Lisäksi aseista ehdittiin toteuttamaan tiede-kenttään magneettikenttä, jonka tarkoitus on vaikuttaa pallon liikesuuntaan osittain sattumanvaraisesti. Magneettikenttä toimii siten, että se liikkuu sattumanvaraisesti kentässä kuitenkaan koskaan siirtymättä tiedeklaanin pelaajan ruudun puolelle. Kun pallo siirtyy riittävän läheltä magneettikenttää, alkaa se vaikuttaa pallon liikerataan. Tämä vaikeuttaa huomattavasti pallon liikeradan ennakoitavuutta.

Mitä jäi kesken?

En ehtinyt toteuttamaan peliin lainkaan ääniä enkä valikkorakennetta. Peli alkaakin tällä hetkellä suoraan siten, että se pyytää jokaista neljää pelaajaa valitsemaan vuorollaan näppäimen, jota pelaaja haluaa käyttää. Tämän jälkeen jokaiselle pelaajalle arvotaan klaani (tästä ei ole vielä mitään visuaalisuutta). Jokaista kenttää pelataan siihen asti, että vain yksi pelaaja on enää jäljellä kentässä.

Kaikki muut kuin armeijaklaanin molemmat aseet ja tiedeklaanin kenttäase on vielä toteuttamatta. Tämä ei kuitenkaan tarkoita sitä, että pelin kehittäminen olisi lopetettu. Pelin kehittämistä jatketaan harrastusmielessä aina kun siihen on aikaa. Kimmo Koivumäki toteuttaa peliin tällä hetkellä vielä grafiikoiden viimeisiä versioita. Uskonkin, että peli tulee valmiiksi kevään 2006 aikana.

7 Yhteenveto ja visio tulevaisuudesta

Tutkintotyön päätarkoituksena oli antaa lukijalle yleiskuva peliohjelmoinnista sekä hieman tarkempi kuva mobiilipelienohjelmoinnista. Tutkintotyön sisällön oli tarkoitus käyttää mahdollisimman paljon pohjana tutkintotyön osana toteutettua mobiilipeliä CyberPong:ia. Työn sisältö oli tarkoitus pitää yleistajuisena niin, että lukija saisi mahdollisimman kattavan yleiskuvan peliohjelmoinnin saralta.

Pelin toteuttamisprosessi osoitti, että mobiilipeliä toteutettaessa kannattaa tehdä kattava suunnitelma pelistä ennen varsinaisen toteuttamisen aloittamista, vaikka kyseessä olisikin pieni ja/tai yksinkertainen peli. Pelin helppo kääntäminen muille laitteille, lokalisointi sekä mobiililaitteiden prosessointi- ja näytönohjain-tehoihin liittyvät asiat tulee ottaa huomioon peliä suunniteltaessa.

Tutkintotyössä esitettyjä toteutusmenetelmiä voidaan suoraan hyödyntää pelejä ohjelmoitaessa. Suurinta osaa työssä esitetyistä asioista käytetään esitetyllä tavalla myös kaupallisissa mobiilipeleissä.

Tutkintotyöprosessi onnistui kokonaisuudessaan alkusuunnitelmieni mukaisesti. Olen erittäin tyytyväinen CyberPong peliin, josta on tulossa huomattavasti alkukuvitelmiani parempi peli. Toivonkin, että jonain päivänä pääsette tutustumaan valmiiseen peliin jossakin muodossa.

Visio tulevaisuudesta

Mobiilipeliala kehittyy mielestäni tällä hetkellä samansuuntaisesti kuin tietokonepeliala kehittyi 90-luvun vaihteessa. Pelaajat näkivät tämän pelien siirtymisenä kaksiulotteisuudesta kolmiulotteisuuteen sekä pelien parempina grafiikkoina ja pelien sisällön lisääntymisenä. Kuitenkin todelliset muutokset tapahtuivat pelialan sisällä. Pelien budjetit ja tuotantoryhmät kasvoivat nopeasti. Pelit muuttuivat yksittäisten henkilöiden harrastuksenaan tekemistä tuotteista yritysten useiden henkilöiden toteuttamiksi kaupallisiksi projekteiksi. Nyt sama kohtalo uhkaa myös mobiilipelialaa.

Mobiililaitteet kehittyvät nopeassa tahdissa tehokkaammiksi ja kaikin puolin paremmiksi pelialustoiksi. Lisäksi mobiilipelien markkinat ovat alkaneet tasaisesti kohota. Markkinoiden nousuhdanteesta esitetäänkin toinen toistaan hurjempia arvioita. Skeptisemmälläkin ajatusmallilla on selvää, että mobiilipelien markkinat ovat kasvamassa kovaa vauhtia. Tämä vaikuttaa mielestäni muun muassa siihen, että mobiilipelejä aletaan pian tuottaa suuremmilla budjeteilla. Tämä on näkyvissä useiden markkinoilla olevien lisenssipelien muodossa. Uskonkin, että suurten kehittäjien tulo markkinoille ja pelien budjettien tasainen kasvu tulee vaikeuttamaan pienten mobiilipelikehittäjien mahdollisuuksia markkinoilla.

Pelaajat tulevat näkemään tämän muutoksen laadukkaampina peleinä, joissa on kaupallisemmat grafiikat ja enemmän sisältöä. Toisaalta pelaajat tulevat näkemään tämän myös innovatiivisten pelien vähentymisenä. Kun pelien budjetit alkavat lähestyä tietokonepelien budjetteja, ei pelien kehittäjillä ole enää varaa ottaa suuria riskejä toteuttamalla innovatiivisia pelejä. Uskonkin, että tällä hetkellä eletään viimeisiä aikoja, jolloin pienten pelikehittäjien on mahdollista tulla osaksi mobiilipelimarkkinoita olematta kuitenkaan osa suurta pelikehitys yritystä.

Keskeisiä lyhenteitä ja käsitteitä

Brew Brew on suosittu mobiilipelien ohjelmointikieli erityisesti Amerikassa. Brew muistuttaa pitkälti C++-kieltä.

C++ C++ on yleisimmin kaupallisten tietokonepelien toteuttamiseen käytetty ohjelmointikieli. Yleensä kaupallisten pelien toteuttamisessa käytetään joko DirectX tai OpenGL rajapintaa C++-kielen kanssa.

CLDC Connected Limited Device Configuration eli CLDC on määritys, joka määrittelee Java-puhelimissa mitä kaikkia perusohjelmointirajapintoja puhelimesta tulee löytyä. Määrityksestä on käytössä tällä hetkellä versiot 1.0 ja 1.1.

FPS Frames Per Second eli FPS tarkoittaa sitä, kuinka monta kertaa ruutu piirretään uudestaan yhden sekunnin aikana. Suurempi luku tarkoittaa sulavampaa liikkuvaa kuvaa ja pienempi luku tarkoittaa pätkivämpää liikkuvaa kuvaa.

J2ME J2ME on suosituin ohjelmointikieli mobiilipelien toteuttamiseen. J2ME pohjautuu Java-ohjelmointikieleen. Java on tulkattava kieli ja tästä syystä jonkin verran muita kilpailevia ohjelmointikieliä tehottomampi. Javaa tukevia mobiililaitteita on markkinoilla erittäin paljon.

MIDP Mobile Information Device Profile eli MIDP on määritys, joka määrittelee ohjelmointirajapintoja joita Java-puhelimista tulee löytyä. Määrityksestä on tällä hetkellä ilmestynyt kaksi versiota 1.0 ja 2.0.

PONG Pong on Atarin vuonna 1972 julkaisema kuuluisa kahdelle pelaajalle suunnattu pallopeli. CyberPong pelin pelimekaniikka perustuu osittain PONG:iin.

Symbian Laitteistoläheinen ohjelmointikieli, joka muistuttaa hyvin pitkälti C++-kieltä. Symbianilla on huomattavasti heikompi laitetuki kuin Java-kielellä. Symbiania käytetään pääasiassa erilaisten mobiilisovellusten toteuttamiseen.

Lähteet

Chandler, Heather 2005. The Game Localization Handbook: Localization Production Pitfalls. [online] [viitattu 28.9.2005]. www.gamasutra.com/features/20050105/chandler_01.shtml

Fitzgerald, Jason 2005. Gran Turismon luoja puhuu!. [online] [viitattu 21.9.2005]. www.granturismoworld.com/gtw.php?jumpTo=fi_FI

Friesen, Jeff 2004. Java Tech: The ABC's of Synchronization, Part 1. [online] [viitattu 4.10.2005]. http://today.java.net/pub/a/today/2004/08/02/sync1.html

Heinonen, Mikko 2005. Gran Turismo 4. [online] [viitattu 19.9.2005]. www.peliplaneetta.net/arvostelut/479/

Inkinen, Maritta 2002. Luettavuus ja typografia. [online] [viitattu 27.9.2005]. www.valt.helsinki.fi/staff/mainkine/3luento.htm

JBenchmark. [online] [viitattu 6.10.2005]. www.jbenchmark.com/index.jsp

Knudsen, Jonathan 2002. What's New In *MIDP* 2.0. [online] [viitattu 5.10.2005]. http://developers.sun.com/techtopics/mobility/*MIDP*/articles/*MIDP*20/

Koivumäki, Kimmo 2005. Mobiilipeligrafiikka Series 60 -alustan matkapuhelimiin. Tutkintotyö. Tampereen Ammattikorkeakoulu, Tietojenkäsittely. Tampere.

LABORSTA Internet. [online] [viitattu 19.9.2005]. http://laborsta.ilo.org/

Metanet Softaware. N Tutorials – Collision Detection and Response. [online] [viitattu 4.10.2005]. www.harveycartel.org/metanet/tutorials/tutorialA.html

Miller, Kurt 2000. Basic Collision Detection. [online] [viitattu 4.10.2005] www.flipcode.com/articles/article_basiccollisions.shtml

Nareyek, Alexander 2004. AI in Computer Games. [online] [viitattu 12.9.2005]. www.ai-center.com/publications/nareyek-acmqueue04.pdf

Olson, Jason 2004. The One With The Game Loop. [online] [viitattu 2.10.2005]. http://geekswithblogs.net/jolson/articles/2173.aspx

Pac-Man. [online] [viitattu 9.9.2005]. http://en.wikipedia.org/wiki/Pac_man#Monsters

Piqna, Chris 2004. The History of Mario. [online] [viitattu 27.9.2005]. www.gamer-talk.net/feature33.html

PONG-Story. [online] [viitattu 19.9.2005]. www.pong-story.com/atpong2.htm

Powell, Robert 2004. Double Buffering Windows Forms. [online] [viitattu 15.9.2005]. www.bobpowell.net/doublebuffer.htm

ProGuard. [online] [viitattu 8.10.2005]. http://proguard.sourceforge.net/

ResearchAndMarkets 2004. Java Versus BREW. [online] [viitattu 5.10.2005]. www.researchandmarkets.com/reports/127849/

RetroGuard. [online] [viitattu 8.10.2005]. www.retrologic.com/

Rouse III, Richard 2001. Game Design: Theory & Practice. Plano: Wordware Publishing, Inc.

Storey, Mike 2004. Optimizing Code to Minimize the Impact of Heap Management. [online] [viitattu 2.10.2005]. http://websphere.sys-con.com/read/45001.htm

Sun Microsystems 2000. Mobile Information Device Profile (JSR-37). [online] [viitattu 20.10.2005]. http://jcp.org/aboutJava/communityprocess/final/jsr037/index.html

Symbian.Symbian OS. [online] [viitattu 4.10.2005]. www.symbian.com/about/symb-os.html

Typebox [online] [viitattu 27.9.2005]. www.typebox.com/typebox/free.html

Liitteet

Liite 1: CyberPong suunnitelmadokumentti

Story:

In a urban tomorrow, the wars between mankind are solved in a more humane way, on the grounds of the CYBERDOME(work title for the game). The leaders of the human clans gather around to decide the earth's final destiny. Military, a clan called the "Green Power", "Drugs'licit", and the clan that calls itself "ScientBrits" take part in this game that sets the fate for earths 16 billion people. Will the earth be governed with guns, regulations and tight laws? Or will the mankind step 3000 years back technologically in an attempt to make the world a greener place to live? Or will all the drugs be legislated and the earth fall to a same anarchy as it was at the end of the 21st century? Or will the earth be ruled by english scientists (there are very few of them).

Clans:

Military
- Pad special weapon uses gun technology and relies on mass destruction
- Level is very organized and gun powered.

Green Power
- Pad taking advantage of plants
- Level could be a forest of somekind

Drugs'licit
- Pad using pills & steroids. Special weapon could affect other players view or the controls for the pad, or it could affect to balls "logic". It could also give extra speed for Drugs'licit's pad.
- Psychedelic & weird level

ScientBrits
- Pad using ultimate technology. Magnetism, electricity and technical innovations.
- Level should be "electrified", and should look very "high tech"

Game controls:

Maximum of four players choose a key from phones keypad. Each player uses his/her key to control the pad as follows:
Single click - Changes pads direction of movement
Double click - Special weapon 1 (depends on the clan)
Triple click - Special weapon 2 (depends on the clan)
All the players can interrupt other players attempt to use special weapon by clicking his/her button fast enough.
(Some special weapons are limited by amount, others can be used within specified timeframe ??)

Questions:

- Should there also be a single player game, or is this game only meant for multiplaying purposes?
- Should clan be randomized, or should each player have a chance to select clan?
- Does each game contain only one "round", or is the game played as a series of multiple rounds/levels?
- Should the played level be randomized or does the levels appear in same order in each game?
- Should there be for example one level designed for each clan?
- Should there be shields for every player, so that game wouldn't end for each player when the first miss with the ball occurs?

Gameplay:

Multiplayer game is played as a drop game, when player misses a ball he/she is eliminated (if shields are run out), when only one player exists he is considered as a winner for the round. Hits with the ball are counted throughout each round and whoever has most hits at the end of the last round wins the game. If one round starts to take too long, game can be hardened by accelerating balls speed, or decelerating players bats movement, or the walls (and the pads) be closened towards the center of the screen.

Level variation points:

Magnetic fields
Ball logic
Electricity
Level size
Pad locations
Pad logic
Pad size
Amount of "power ups"
Amount of "shields"
Obstacles on levels
(Musics)

Flowchart:

Splash screen
|
Title screen, containing loading bar
|
Main menu
| | | | |
New game Credits Options Highscore Exit
|
Amount of players
|
Key selection
|
(Clan selection)
|
(Level selection)
|
Game/level intro
|
Game
|

New round
|
Game over, final scores
|
"Enter name for highscore"
|
Highscore
||
Back to mainmenu