Lars-Patrick Corander

Alustariippumattoman mobiilipelin kehitys

Vancouver 2010 - Official Mobile Game

Metropolia Ammattikorkeakoulu
Insinööri
Tietotekniikka
Insinöörityö
2.5.2011

Metropolia

Tiivistelmä

Tekijä(t) Otsikko Sivumäärä Aika	Lars-Patrick Corander Alustariippumattoman mobiilipelin kehitys 38 sivua + 1 liite 2.5.2011
Tutkinto	insinööri (AMK)
Koulutusohjelma	tietotekniikka
Suuntautumisvaihtoehto	ohjelmistotekniikka
Ohjaaja(t)	Head Of Studio Markus Pasula yliopettaja Jarkko Vuori

Insinöörityössä tarkoituksena oli toimia tärkeänä osana alustariippumattoman Vancouver 2010 -mobiilipelin kehityksessä peliohjelmoijana. Tämän lisäksi tutkittiin alustasta riippumattoman pelin tai minkä tahansa graafisen sovelluksen ohjelmalliseen toteutukseen liittyviä haasteita. Tarkasteltiin kuinka alustojen eri resoluutiot vaikuttavat graafisen käyttöympäristön ja pelimaailman piirrossa tai kuinka laitteiden vaihteleva suorituskyky tulee huomioida pelin mekaniikoissa ja fysikaalisissa simuloinneissa, jotta peli toimisi mahdollisimman tasavertaisesti laitteella kuin laitteella.

Peliä kehitettiin Mr. Goodliving -nimisessä yrityksessä ja kehityksen ydintiimiin kuului yhdeksän henkilöä. Työssä hyödynnettiin joitakin yrityksen valmiita teknologioita, joista tärkeimpänä osana alustariippumaton pelimoottori EMERGE. Pelimoottori mahdollisti pelikoodin kääntämisen useille eri alustoille ja abstrahoi peliltä kaikki matalantason alustariippuvaisuudet.

Laaja alustariippumattomuus toi tiettyjä rajoitteita pelin ohjelmointiin, kuten esimerkiksi olio-ohjelmoinnista, osoittimista ja staattisista taulukoista oli luovuttava. Peli valmistui täysin ajallaan, se sisälsi viisi eri talviurheilulajia sekä tuki loppujen lopuksi hieman yli tuhatta eri alustaa. Pelistä saatiin erittäin skaalautuva, sillä pienimmillään pelipaketin koko oli 64 kilotavua ja suurimmillaan 20 megatavua.

Vaikka skaalautuvuus toi mukanaan useita kompromisseja eikä pelissä voitu hyödyntää viimeisintä teknologiaa, niin juuri erinomaisen skaalautuvuutensa vuoksi peli saatiin toimimaan yli tuhannella eri alustalla. Skaalautuvuudesta oli myös hyötyä itse pelin kehityksessä, sillä peliin voitiin tehdä tarpeen vaatiessa muutoksia hyvin pienellä vaivalla.

Avainsanat	alustariippumaton, ohjelmointi, mobiili, pelin kehitys

Metropolia

Abstract

Author(s) Title Number of Pages Date	Lars-Patrick Corander Developing Multiplatform Mobile Game 38 pages + 1 appendices 2 May 2011
Degree	Bachelor Of Engineering
Degree Programme	Information Technology
Specialisation option	Software Technology
Instructor(s)	Markus Pasula, Head Of Studio Jarkko Vuori, Principal Lecturer

Software developers working with mobile devices face numerous challenges when trying to cope with the highly fragmented mobile device market of today. Porting native software from one platform to another is a demanding task to do and making a fully featured multiplatform software development engine is by no standards a trivial and cheap way to opt for**.** Therefore, many companies tend to support only a few of the high-end devices available in the market. However, these high-end devices cover only a small fraction of the entire market and hence a company with ability to cost-effectively support a wide range of devices on the market is potentially in a highly advantageous position.

The main goal of this work was to take part in the development of Vancouver 2010 Official Mobile Game as a game programmer and research the development of a cross-platform mobile game. This game project was used to reflect some of the challenges introduced by having the game support multiple platforms. For example, how differences in resolution affect the graphical user interface design and implementation or game world visualization. Or how differences in performance should be compensated to maintain similarity in playability with different kinds of devices.

The game was developed by Mr. Goodliving and the core development team consisted of nine persons. Some of the company's proprietary technologies were utilized during the game development. The most important one of these was the game engine EMERGE, which offered us an extensive API and toolset for cross-platform game development.

Supporting a wide variety of devices had its limitations of course. For example, pointers from C-based languages were forbidden and object-oriented programming features could not be used because of constraints in some supported devices. Despite these constraints the game was released on time and it had five sport events included. In the end it supported more than one thousand devices and the size of the game package varied from 64 kilobytes to 20 megabytes.

Keywords	cross-platform, game development, programming, mobile

Sisällys

Lyhenteitä ja käsitteitä

App Store	Applen ylläpitämä verkossa toimiva sovelluskauppa.
C	Alun perin järjestelmäohjelmointiin tarkoitettu matalan tason ohjelmointikieli.
C++	C-kieleen perustuva olio-pohjainen ohjelmointikieli.
EMERGE	Mr. Goodlivingin kehittämä alustariippumaton pelimoottori.
GCC-esikääntäjä	C/C++ -kielisten kooditiedostojen esikääntäjä. Käsittelee kooditiedostot ennen kuin ne lähetetään varsinaiselle kääntäjälle.
Java	Laitteistoriippumaron olio-pohjainen ohjelmointikieli.
Objective-C	C-kieleen perustuva olio-pohjainen ohjelmointikieli.

1 Johdanto

Mobiililaitteiden merkitys ihmisten jokapäiväisessä elämässä on kasvanut viime vuosina merkittävästi. Kun puhutaan mobiililaitteesta, ei tarkoiteta enää välttämättä vain laskinta, hakulaitetta tai muuta hyvin rajalliseen käyttöön tarkoitettua laitetta. Useat mobiililaitteet lähentelevät ominaisuuksiltaan jo normaalia tietokonetta, ja jotkin mobiililaitteet ovat jopa tehokkaampia kuin hieman vanhemmat perustietokoneet. Mobiililaitteeksi voidaan lukea kaikki tietotekniset ns. kämmenkokoiset laitteet. Nykyajan yleisimpiä mobiililaitteita ovat matkapuhelimet, gps-paikannuslaitteet sekä käsipelikonsolit. Näistä suosituin mobiililaitteen muoto on mobiilipuhelin. Monilla nykyaikaisilla perusmatkapuhelimilla voi soittamisen ja tekstiviestien lähettämisen lisäksi ottaa kuvia, soittaa musiikkia sekä pelata pelejä. Älypuhelimiksi voidaan luokitella puhelimet, jotka lähentelevät ominaisuuksiltaan tietokonetta. Älypuhelimilla voi yleensä esimerkiksi navigoida, selata verkkosivuja. Älypuhelimet ovat myös monesti huomattavasti tehokkaampia ja niissä on enemmän tallennustilaa kuin perus mobiilipuhelimsessa. [7.]

Mobiilipeli on peli, jota voidaan pelata mobiililaitteella. Useimmiten mobiilipelillä tarkoitetaan peliä joka toimii mobiilipuhelimessa. Käsikonsolilla pelattavia pelejä taas ei ole yleensä luokiteltu mobiilipeleiksi, sillä ne ovat usein huomattavasti kehittyneempiä kuin mobiilipelit. Mobiilipelien ja käsikonsolipelien ero on kuitenkin hyvin pieni, sillä monet älypuhelimissa toimivat pelit vastaavat laadultaan jo käsikonsolipelejä. Lisäksi monissa nykyaikaisissa käsikonsoleissa on mahdollisuus hakea pienempiä pelejä myös verkosta. Tämä kaventaa mobiilipelien sekä käsikonsolipelien välistä eroa entisestään. Tässä työssä termi mobiilipeli ei rajaudu vain mobiilipuhelimiin vaan se käsittää sisälleen myös käsikonsolipelit. [8.]

Hyvin suuri osa mobiilipelejä kehittävistä yrityksistä on fokusoitunut vahvasti, ellei täysin älypuhelimiin viime vuosina. Ensisijaisesti sovelluksia kehitetään Applen laitteille ja toiseksi suosituimpia ovat Android-mobiilikäyttöjärjestelmää hyödyntävät älypuhelimet. Miksi fokusoida ainoastaan näille laitteille? Vastaus lienee se, että nyky-ajan älypuhelimet vastaavat teholtaan jopa hieman vanhempaa tehokasta pöytätietokonetta. Tuolloinen tehokas tietokone kulkee nykyään mukana taskussa. On

siten hyvin selvää, että kiinnostus mobiililaitteissa toimiviin sovelluksiin on kasvanut räjähdysmäisesti. Mobiililaitemarkkinat ovat kuitenkin erittäin fragmentoituneet, eli laitevalmistajia on useita eikä kellään ole merkittävän suurta osuutta markkinoista hallinnassaan. Pyrin työlläni osoittamaan alustariippumattomuuden merkityksen näillä markkinoilla sekä sen miksi tukea myös ehkä hieman edullisempia mobiililaitteita. Hyvin toteutetussa kehitysympäristössä sovelluskehittäjä voi keskittyä suurilta osin juuri huippulaitteen implementaatioon ja hyödyntämällä tätä kehitysympäristöä skaalata sovelluksensa kustannustehokkaasti monelle eri laitteelle sekä näin tavoittaa erittäin laajalti yleisöä sovelluksellaan.

Olin mukana kehittämässä vuonna 2010 Vancouverissa järjestettyjen talviolympialaisten virallista mobiilipeliä. Peli kehitettiin Suomessa yrityksessä nimeltä Mr. Goodliving, joka on amerikkalaisen Real Networksin tytäryhtiö. Toimin projektissa peliohjelmoijana ja vastasin kahden kilpailulajin mekaniikoiden ohjelmoinnista sekä lukuisista muista pelin kehitykseen liittyvistä ohjelmatoteutuksista. Pelin tuli tukea yli tuhatta eri mobiililaitealustaa. Pelissä hyödynnettiin yrityksen kehittämää alustariippumatonta pelimoottoria EMERGEä, joka abstraktoi lukuisia alustariippuvaisia osia peliohjelmoijalta.

Työn pääasiallisena tarkoituksena on antaa lukijalle kokonaisvaltainen kuva alustariippumattomasta pelinkehityksestä mobiililaitteilla. Tutkin ensin markkinoita ja alustariippumattoman sovelluksen kannattavuutta. Tämän jälkeen tuon esille joitakin alustariippumattoman pelin teknisiä perusvaatimuksia ja niille mahdollisia ratkaisuja. Lopuksi kuvailen hieman EMERGE-pelimoottorin rakennetta ja käyn läpi Vancouver 2010 -mobiilipeliprojektin.

Mr.Goodliving Oy

Mr.Goodliving oli vuonna 1999 perustettu suomalainen ns. kasuaalipelehin erikoistunut mobiilipeliyritys, joka nousi ensimmäisen kerran otsikoihin suuremman amerikkalaisen yrityksen Real Networksin ostaessa sen vuonna 2005 noin 15 miljoonalla eurolla. Pääasiallisena syynä ostoon oli yrityksen kyky tukea kustannustehokkaasti tuhansia eri mobiililaitteita hyödyntämällä talon sisällä kehitettyä alustariippumatonta pelimoottoria EMERGEä. Mr. Goodlivingin pelistudio kuitenkin suljettiin 2011 alkuvuodesta Real

Networksin toimesta sen siirtäessä fokustaan muunlaiseen liiketoimintaan. Mr. Goodliving tuotti keskimäärin 15 peliä vuodessa, joista osa kehitettiin talon sisällä ja osa ulkoistettiin. Suurin osa peleistä oli lisenssipelejä ja osa omia ideoita, näistä tunneituimpana Playman Sports -urheilupelisarja ja Tiki Towers. [9.]

2 Mobiililaitemarkkinat

Tässä luvussa käyn läpi päällisin puolin mobiililaitemarkkinoiden tilanteen. Kuinka markkinat esimerkiksi eroavat muista tietotekniikan alojen markkinoista, kuten tietokonemarkkinoista. Käsittelen myös markkinoiden laajaa fragmentoitumista. Luvusta selviää tällä hetkellä suosituimmat mobiililaitevalmistajat ja mobiilikäyttöjärjestelmät sekä niiden markkinaosuudet.

Mobiililaitemarkkinat kehittyvät ja muuttuvat jatkuvasti erittäin nopealla vauhdilla. Markkinoista erityisen haasteellisen tekee markkinoiden fragmentoituminen. Valmistajia ja käyttöjärjestelmiä on lukuisia sekä useimmat valmistajat tuovat jatkuvasti uusia laitteita markkinoille uusien ominaisuuksien kera. Täysin uusia käyttöjärjestelmiäkin julkaistaan kohtalaisen usein. Tästä syystä laitteet, joissa on runsaasti tehoa, kattava ja monipuolinen kehitysympäristö sekä suhteellisen suuri käyttäjäkunta, keräävät itselleen suuren määrän omistautuneita kehittäjiä ja yrityksiä. Hyvinä esimerkkeinä helposti lähestyttävästä mobiililaitteista on Applen valmistamat tuotteet: iPhone, iPad, ja iPod. Kehittäminen yhdelle alustaperheelle helpottaa sovelluskehittäjän arkea huomattavasti, mutta kääntöpuolena tässä ratkaisussa on se, että kehittäjä ei pääse tavoittamaan kuin vain pienen osan koko mobiililaitemarkkinoista. Siksi ennen sovelluksen kehittämisen aloittamista olisi tärkeätä huomioida voisiko sovellus mahdollisesti toimia muillakin markkinoiden laitteilla ja kuinka paljon suurempi yleisö silloin on potentiaalisesti tavoitettavissa. On selvää, että esimerkiksi raskasta 3D-renderöintiä sisältävä sovellus rajautuu mahdollisesti vain muutamille tehokkaille älypuhelimille sen teknisten vaatimusten takia. Mutta mikäli kyseessä oleva sovellus ei ole varsinaisesti riippuvainen markkinoiden uusimmista ominaisuuksista tai erittäin tehokkaasta laitteistosta, useamman alustan tukeminen voi potentiaalisesti maksaa itsensä moninkertaisesti takaisin.

Tietokoneiden teknologia kehittyy myös jatkuvasti kovalla vauhdilla, mutta käytännössä alusta pysyy pääosin samana ja kehitysympäristöön tulee vähän muutoksia. Sovelluskehittäjän näkökulmasta muutoksia tulee hyvin vähän. Tietokoneiden välinen alustariippumattomuus tai käyttöjärjestelmäriippumattomuus harvemmin muodostuu ongelmaksi, sillä suosittuja käyttöjärjestelmiä on vain muutama. Mobiililaitemarkkinoilla uusia alustoja ja käyttöjärjestelmiä taas julkaistaan useammin. Mobiililaitteiden väliset

erot ovat paljon suuremmassa merkityksessä, sillä laitteissa on lukuisia erilaisia tapoja ohjata niitä. Esimerkiksi toisissa on näppäimet, kun toisissa taas pelkkä kosketusnäyttö. Hieman vanhemmat mallit eivät tue liukulukuja laisinkaan. Laitteista itsestään löytyvien erojen lisäksi suuria eroja on myös laitteiden käyttöjärjestelmissä.

Älypuhelimet

Älypuhelimet ovat selkeästi vallanneet reilusti markkinaosuutta niiden tavoittaessa nyt myös ns. keskivertokuluttajat, eikä niitä markkinoida enää niinkään vain bisnespuhelimina. Johtavat yritykset älypuhelinmarkkinoilla ovat edelleen Nokia ja RIM (Research In Motion, Blackberry -puhelimia valmistava yritys), ja muillakin valmistajilla käyttäjiä riittää. Apple on vallannut itselleen hyvän markkinaosuuden suhteellisen lyhyessä ajassa. Taulukosta 1 on nähtävissä, että jo muutaman eri alustan tukemisella voi tavoittaa huomattavasti enemmän asiakkaita.

Taulukko 1. cnet Newsin julkaisema tilasto, maailman viiden suosituimman älypuhelinvalmistajan toimitettujen yksiköiden määrä (yksiköt miljoonissa) ja markkinaosuus. [1.]

Valmistaja	**2010 Yksikköä toimitettu**	**2010 Markkinaosuus**	**2009 Yksikköä toimitettu**	**2009 Markkinaosuus**	**Vuotuinen kasvu**
Nokia	100,3	33,1 %	67,7	39,0 %	48,2 %
RIM	48,8	16,1 %	34,5	19,9 %	41,4 %
Apple	47,5	15,7 %	25,1	14,5 %	89,2 %
Samsung	23,0	7,6 %	5,5	3,2 %	318,2 %
HTC	21,5	7,1 %	8,1	4,7 %	165,4 %
Muut	61,5	20,3 %	32,6	18,8 %	88,7 %
Yhteensä	**302,6**	**100,0 %**	**173,5**	**100,0 %**	**74,4 %**

Kaikki mobiilipuhelimet

Taulukosta 2 on nähtävissä kokonaisvaltaisesti kaikkien mobiilipuhelinvalmistajien markkinaosuudet ja toimitettujen laitteiden määrät. Mukana ovat siis myös älypuhelimet.

Taulukko 2. Gartnerin julkaisemat tilasto mobiilipuhelinten myynneistä ja markkinaosuuksista vuosilta 2009 ja 2010 (miljoonaa yksikköä) [2]

Valmistaja	2010 Yksikköä toimitettu	2010 Markkinaosuus	2009 Yksikköä toimitettu	2009 Markkinaosuus
Nokia	461,3	28,9 %	440,9	36,4 %
Samsung	281,1	17,6 %	235,8	19,5 %
LG	114,2	7,1 %	122,0	10,1 %
RIM	47,5	3,0 %	34,3	2,8 %
Apple	46,6	2,9 %	24,8	2,1 %
Sony Ericsson	41,8	2,6 %	54,9	4,5 %
Muut	704,1	37,9 %	298,2	24,6 %
Yhteensä	**1596,8**	**100,0 %**	**1211,2**	**100,0 %**

Ainoastaan Applen laitteiden tukeminen saattaa vaikuttaa hyvältä vaihtoehdolta, kun tarkastellaan taulukkoa 1. Mutta kun lukuja verrataan kaikkien markkinoilta löytyvien mobiilipuhelimien kanssa (taulukko 2), Applen osuus onkin marginaalisen pieni. Monet nykyajan mobiilipuhelimista ovat suorituskyvyltään jo joidenkin älypuhelimien vertaisia, ja ne soveltuvat muutenkin oivallisesti pelaamiseen. On totta, että Applen laitteilla sovelluksia myydään suhteessa paljon enemmän kuin muilla laitteilla. Mutta taulukko 2 osoittaa hyvin, kuinka suuret erot ovat kyseessä laitteden lukumäärien välillä. Näin suuret laitemäärät kompensoivat jonkin verran vähemmän vilkasta sovelluskauppaa. On myös hyvin todennäköistä, että Applen suosion myötä sovellusten ja sovelluspalvelujen kysyntä kasvaa myös muilla alustoilla.

Käyttöjärjestelmät

Käyttöjärjestelmät kuvastavat parhaiten markkinoiden fragmentoitumista. Monesti mobiililaitteen valmistaja kehittää oman käyttöjärjestelmänsä, joka on optimoitu juuri sen laitteille. Markkinoilla on toki myös monialustaisia käyttöjärjestelmiä, kuten Android, Brew OS tai Windows Mobile. Hyvin suosittua Android-käyttöjärjestelmää ovat hyödyntäneet älypuhelimissaan muun muassa HTC, LG, Samsung ja Motorola. Vaikka Samsung on hyödyntänyt Android käyttöjärjestelmää, se toi markkinoille myös oman käyttöjärjestelmänsä Badan, joka on tarkoitettu tavallisille mobiilipuhelimille ja alemman tason älypuhelimille. Tämä kuvastaa hyvin laitevalmistajien kovaa tahtoa

tuoda juuri niiden oma käyttöjärjestelmänsä markkinoille. Taulukosta 3 nähdään että tilanne eri käyttöjärjestelmien välillä on tasaväkinen.

Taulukko 3. Gartnerin julkaisema tilasto eri käyttöjärjestelmistä älypuhelimissa (miljoonaa yksikköä) [2]

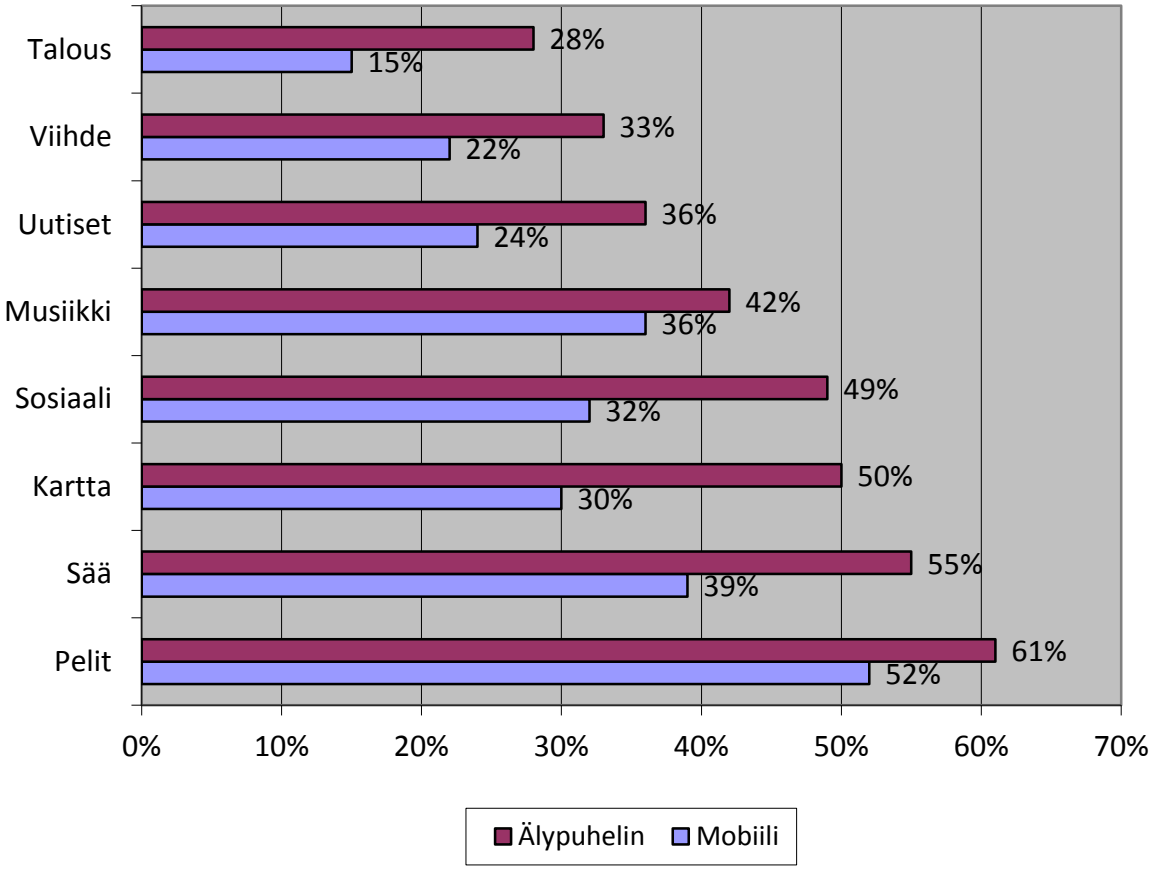
Käyttö-järjestelmä	**2010 Yksiköitä**	**2010 Markkinaosuus**	**2009 Yksiköitä**	**2009 Markkinaosuus**
Symbian	111,6	37,6 %	80,8	46,9 %
Android	67,2	22,7 %	6,7	3,9 %
RIM	47,4	16,0 %	37,3	19,9 %
iOS	46,5	15,7 %	24,8	14,4 %
Windows Mobile	12,3	4,2 %	15,0	8,7 %
Muut	11,4	3,8 %	10,4	6,1 %
Yhteensä	**296,6**	**100,0 %**	**172,3**	**100,0 %**

Mobiilisovellus- ja mobiilipelimarkkinat

Tällä hetkellä Applen App Store on kirkkaasti kärjessä mobiilisovellusten myynnissä. Tästä johtuen kehittäminen Applen laitteille vaikuttaa hyvin houkuttelevalta vaihtoehdolta. Siksi taas yhä useampi yritys ei kehitä pelejä enää muille alustoille, vaan on jättänyt leikin kesken ja suunnittelee sovelluksia vain Applen laitteille. Mutta kuten taulukosta 2 esimerkiksi on nähtävissä, Applen laitteiden osuus maailmalla on kuitenkin suhteellisen pieni, eikä sitä kautta kehittäjä pysty tavoittamaan laajaa yleisöä markkinoilla.

Markkinoilla on lukuisia suorituskyvyltään iPhonen vertaisia laitteita. Ongelma ei kuitenkaan ole laitteissa vaan kehitysympäristöissä ja sovellusten myynti- ja välityspaikoissa. App Store on helppoudellaan ja yksinkertaisuudellaan saanut aikaan räjähdysmäisen kasvun mobiilisovellusmarkkinoilla. Tämä on herättänyt paljon kehittäjiä ja kaikki ovat lähteneet ”takomaan kultaa” samoille apajille. Mutta on selvää että muut yritykset eivät aio katsoa tätä vierestä, vaan kehittävät todennäköisesti omia ratkaisujaan mobiilisovellusmarkkinoille. Mobiilisovellusten välitys käyttäjille tulee helpottumaan monilla käyttöjärjestelmillä App Storen myötä ja tämä tulee ylläpitämään markkinoiden fragmentoitumista, mikä puolestaan vahvistaa sovelluskehittäjän tarvetta

tukea useampaa kuin yhtä alustaa, mikäli myynti halutaan maksimoida. Mobiilipelien suhteen asiat ovat toisaalta hyvällä mallilla, sillä ne ovat suosituin sovellusmuoto mobiilisovellusmarkkinoilla. Kuviosta 1 on nähtävissä eri mobiilisovelluskategorioiden suosio. Kuviossa mitataan sitä, kuinka suuri osa käyttäjistä on ostanut viimeisen 30 päivän aikana sovelluksen kyseisestä kategoriasta. [5.]



Kuvio 1. Suosituimmat mobiilisovelluskategoriat

3 Alustariippumattomuus

Alustariippumattomuudella tarkoitetaan yleensä jotakin teknologian osaa tai työkalua, joka ei ole riippuvainen jostain tietystä ympäristössä. Ympäristö voi olla esimerkiksi laitteistoalusta tai käyttöjärjestelmä. Hyviä esimerkkejä alustariippumattomuudesta ovat nykyaikaiset ohjelmointikielet, kuten C-kieli tai Java. C-kieli piilottaa käyttäjältä useimmat laitteistoriippuvaisuudet eikä käyttäjän tarvitse ohjelmoidessaan huolehtia esimerkiksi prosessorin käskykannasta. Alustariippumaton sovellus tai ohjelmointikieli ei kuitenkaan tarkoita, että se toimisi automaattisesti kaikilla laitteilla tai käyttöjärjestelmillä, vaan sitä, että siinä ei ole suoria sidoksia laitteistoalustaan tai käyttöjärjestelmään. Tämä ei tarkoita sitä että sidoksia ei voisi käyttää vaan sitä että sidokset voidaan joko käännön tai ajon aikana muuntaa alustan yhteensopiviksi. Useimmiten nämä sidosten muunnokset tehdään ohjelman käännön aikana. Oliopohjaisissa kielissä tämä toteutetaan yleisesti hyödyntämällä luokkien perintää sekä polymorfismia. [3.]

Alustariippumattomia sovelluksia on kahdenlaisia. On sellaisia, joissa yksi käännetty versio toimii suoraan kaikilla alustoilla, ja sellaisia joissa kaikille tuetuille alustoille on käännettävä oma versionsa. Kun on kyse peleistä, jotka vaativat useimmiten maksimaalisen suorituskyvyn, sovelletaan jälkimmäistä tyyppiä, sillä kaikista tehokkainta on aina käyttää suoraan alustan natiivia rajapintaa. Tämä ei olisi mahdollista ensimmäisessä tyypissä, sillä sovelluksen täytyy ajon aikana muuntaa tapahtumat yhteensopiviksi alustan kanssa.

3.1 Alustariippumattoman pelin perusvaatimukset

Alustariippumatonta sovellusta kehitettäessä on tärkeätä huomioida sen kaikki alustasta riippuvaiset ominaisuudet. Ominaisuudet vaihtelevat paljolti riippuen sovelluksen tarkoituksesta. Alustariippumattoman pelin kehittäminen on haastellista, sillä pelin mekaniikoiden tulee toimia mahdollisimman samalla tavalla jokaisella alustalla ja useimmiten pelin täytyy toimia reaaliajassa riittävällä nopeudella. Täysin samanlaista pelikokemusta on kuitenkin mahdotonta saavuttaa kahdella täysin erilaisella laitteella. Voi olla, että toisessa laitteessa olevat näppäimet yksinkertaisesti soveltuvat pelaamiseen niin paljon paremmin kuin toisen laitteen kosketusnäyttö. Tai

jos pelin kuvaruutu päivitetään toisessa puhelimessa 60 Hz:n taajuudella ja toisessa vanhemmassa mallissa esimerkiksi 10 Hz:n taajuudella, on selvää, että pelikokemus ei ole täysin samanlainen. Laitteiden eroja voidaan kuitenkin kompensoida, jos peli implementoidaan mahdollisimman skaalautuvaksi ja pelin eri pelimekaaniset ominaisuudet ovat säädettävissä.

Alustariippumattoman pelin kehitykseen ei ole yhtä oikeaa ratkaisua, joka saa pelin toimimaan automaattisesti kaikilla alustoilla. Tärkeä aspekti heti toimivan pelimoottorin lisäksi on pelin itsensä skaalautuvuus. Osa pelin skaalautuvuudesta voikin tulla osaksi jo pelimoottorin ansiosta, mutta useimmiten skaalautuvuus on otettava huomioon jo peliä kehittäessä. Pelin suorituskyky, muistinkulutus ja grafiikat on hyvä olla säädettävissä. Pelimekaniikkaan on myös hyvä tehdä säädettäviä paremetrejä, joilla pelimekaaniikkaa voidaan hienosäätää hieman helpommaksi laitteilla, joissa on huonompi pelattavuus, jotta pelissä saavutetut tulokset olisivat vertailukelpoisia.

3.2 Pelimoottori

Ennen alustariippumattoman pelin kehittämistä on järkevää kehittää jonkinlainen pelimoottori. Pelimoottorin tarkoituksena on abstrahoida peliltä kaikki alustariippuvaiset osat ja ennen kaikkea tarjota pelille kaikki sen tarvitsemat perusominaisuudet, kuten kuvanpiirron, käyttäjän syötteen tai äänien soittamisen. Pelimoottoria kehitettäessä on hyvä kartoittaa, kuinka laajalti laitteita on tuettava ja pitääkö pelimoottorin olla mahdollisesti laajennettavissa uusia laitteita tai käyttöjärjestelmiä varten.

Pelimoottorin yleisimpiä peruskomponentteja ovat

- järjestelmä
- grafiikka
- käyttäjän syöte
- ääni
- datan tallennus ja luku
- muistin hallinta
- resurssien hallinta

Järjestelmärajapinnan tarkoituksena on abstrahoida itse sovelluksen alustus, käynnistys ja sulkeminen peliltä. Yleensä pelin ei tarvitse suoraan kutsua järjestelmärajapintaa. Rajapintaa kutsutaan usein suoraan main-funktiosta. Yleensä rajapinta vastaanottaa myös käyttöjärjestelmältä tulevat tapahtumat ja kutsut sekä välittää ne sitten mahdollisesti muille rajapinnoille.

Grafiikkakomponentin tarkoitus on piilottaa peliltä matalantason yksityiskohdat grafiikkaan liittyen ja tarjota yhtenäinen rajapinta grafiikan piirrolle alustasta riippumatta. Pelin ei siis tarvitse tietää, mitä grafiikkakirjastoa piirtämiseen käytetään.

Käyttäjän syöte -rajapinnan tarkoitus on välittää pelille käyttäjältä tulevia komentoja. Rajapinta on usein vahvasti sidonnainen järjestelmän rajapinnan kanssa.

Rajapinta äänien toistoa varten. Mobiilipeleissä äänet ovat monesti olleet hyvin vaatimattomia laitteiston rajoittuneisuuden vuoksi.

Pelitilanteen ja suoritusten tallentamiseksi sekä lukemiseksi on hyvin tärkeä toteuttaa rajapinta datan tallentamiselle ja lukemiselle.

Mikäli pelin on tarkoitus tukea alustoja eri kielillä, täytyy myös muistin hallinta abstrahoida. Esimerkiksi Javassa muistia ei tarvitse erikseen vapauttaa kuten C++:ssa.

Resurssien hallinta vastaa sovelluksen kaikesta staattisesta sisällöstä ja datasta kuten kuvista, äänistä, kenttä-datasta ja niiden pitämisestä muistissa. Resurssien hallinnan avulla voidaan hallinnoida milloin mitäkin on sovelluksen muistissa, esimerkiksi kun peli aloitetaan on luultavasti viisainta vapauttaa kaikki päävalikossa käytetyt resurssit muistista.

3.3 Resoluutioriippumattomuus

Laitteissa on hyvin paljon erilaisia resoluutioita, ja pelin tulisi näyttää mahdollisimman hyvältä sekä kaiken tulisi näkyä ruudulla resoluutiosta huolimatta. Resoluutioriippumattomuus jakautuu kahteen pääryhmään: grafiikat eli kuvatiedostot sekä koordinaatit.

Kuvat

Pelissä käytettäviä kuvia ei kannata skaalata ajon aikana sopivan kokoisiksi, sillä kaikki alustat eivät tue grafiikoiden skaalausta tai se saattaa olla erittäin hidasta. Ajonaikainen grafiikoiden skaalaus ei myöskään näytä hyvältä. On toki mahdollista skaalata grafiikat pelin käynnistyessä, mutta latausaikojen pidentäminenkään ei ole hyvä vaihtoehto. Laitteet olisi hyvä jakaa eri resoluutioluokkiin, joihin ne jaetaan resoluutionsa perusteella. Luokkia ei yleensä tarvitse olla kuin 3–6 kpl, riippuen tuettujen resoluutioiden määrästä ja variaatiosta. Ensisijaisesti tuotetaan grafiikat korkeimpaan tuettuun resoluutioon sen kokoisena, kuin halutaan niiden siinä resoluutiossa näkyvän. Tämän jälkeen kuvaa skaalataan pienemmäksi kuvankäsittelyohjelmassa jollain vakiokertoimella. Kerroin, jolla kuvia skaalataan eri resoluutioluokkien välillä riippuu tuetuista resoluutioista ja resoluutioluokka määrittelystä. Kuviossa 3 on esimerkki samasta grafiikasta neljässä eri resoluutioluokassa.

Kuvio 2. Kuvan skaalaus eri resoluutioluokkiin. Kuvat vastaavat EMERGEssä luokkia R2–R5.

Koordinaatit

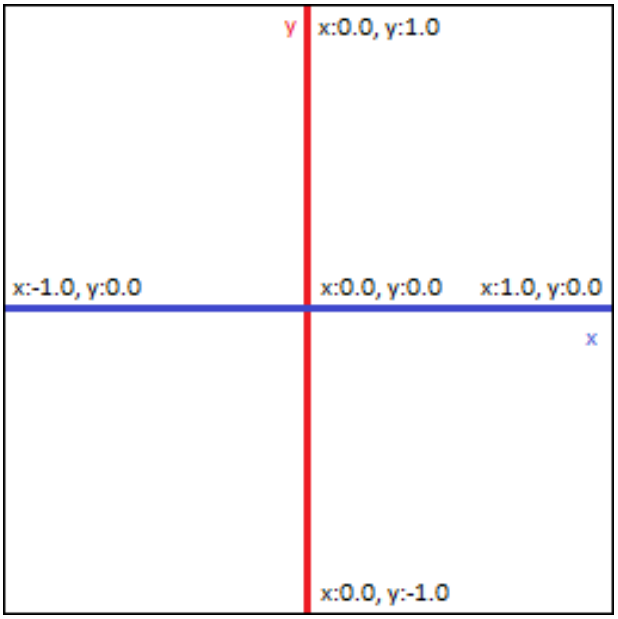
Resoluution vaihdellessa pelissä ei kannata usein tehdä suoria viittauksia ruudun koordinaatteihin, vaan sen sijaan tulisi käyttää jonkinlaista suhteellista koordinaattijärjestelmää. Esimerkikoodi 1 havainnollistaa mahdolliset muunnosfunktiot suhteellisista koordinaateista ruudun koordinaatteihin (pikselikoordinaatit):

```
#define OFFX (RES_X / 2)
#define OFFY (RES_Y / 2)

float get_screen_x( float x ) { return OFFX + x*0.5f*RES_X; }
float get_screen_y( float y ) { return OFFY - y*0.5f*RES_Y; }
```

Esimerkkikoodi 1. Resoluutioriippumaton koordinaatijärjestelmä.

Ruutukoordinaatit alkavat laitteilla yleensä vasemmasta yläkulmasta, jossa x-akseli on poikittainen akseli ja y-akseli pystysuuntainen akseli. Origon ollessa ylälaidassa osoittaa positiivinen y-akseli alaspäin. Esimerkkikoodi 1 muuntaa koordinaattijärjestelmän kuvion 3 kaltaiseksi.



Kuvio 3. Visualisaatio esimerkkikoodi 1:n mukaisesta koordinaattijärjestelmästä.

Koordinaattijärjestelmän tyyppi määräytyy hyvin pitkälti sovelluksen käyttötarkoituksen perusteella. Yleensä peleillä on useita eri koordinaattijärjestelmiä eri käyttötarkoituksia varten. Yleisin esimerkki on valikot ja pelimaailma. Pelimaailma saattaa hyödyntää esimerkiksi 3D-projisiota, kun taas esimerkkikoodissa 1 esitetty malli voisi olla käytössä pelin valikoissa. Tämä koordinaattijärjestelmä varmistaa sen, että esimerkiksi koordinaatti *P{0.8, 0.8}* sijaitsee aina lähellä ruudun oikeaa ylälaitaa resoluutiosta huolimatta.

3.4 Suorituskykyriippumattomuus

Peli on suorituskykyriippuvainen jos, laitteen suorituskyky vaikuttaa siinä tapahtuvien animaatioden tai simulaatioiden nopeuteen. Suorituskykyriippumattomuuden saavuttamiseksi yhtälössä on huomioitava aika. Esimerkkikoodissa 2 on esimerkki suorituskykyriippuvaisesta ohjelmasta.

```
// pelin pääsilmukka: pelin simumlointi ja piirto
float dt = 1/60;     // yksi fysiikka-askel vastaa 1/60 sekuntia
while( running ) {
          physics(dt); // päivitä kappaleiden sijainnit
          draw();    // piirra kappaleet
}
```

Esimerkkikoodi 2. Suorituskykyriippuvainen ohjelma.

Kuten esimerkkikoodissa 2 on nähtävissä, mitä useammin *physics*-funktiota kutsutaan, sitä nopeammin kappaleet liikkuvat. Funktiota ei ole siis synkronoitu ajan kanssa millään tavalla. Esimerkkikoodissa 3 on osoitus huonosta tavasta tehdä suorituskykyriippumaton ohjelma.

```
float lastTime = getCurrentTime();
while( running ) {
          float newTime = getCurrentTime();
          float dt = newTime – lastTime;
          lastTime = newTime;

          physics(dt);
          draw();
}
```

Esimerkkikoodi 3. Epästabiili suorituskykyriippumaton ohjelma.

Esimerkkikoodissa 3 on nähtävissa suorituskykyriippumaton ohjelma, mutta se on epästabiili. Kappaleet liikkuvat pidemmän matkaa jos *dt* on suurempi ja vähemmän matkaa jos pienempi. Mutta jos laite on erittäin hidas tai käyttöjärjestelmä tekee taustalla joitain operaatioita, saattaa *dt* ollakin odottamattoman suuri, esimerkiksi kaksi sekuntia. Kaikki kappaleet liikkuvat kerralla matkan jonka ne liikkuisivat kahden sekunnin aikana, eikä aikaa sen välillä simuloida laisinkaan. Tämä johtaa hyvin epästabiileihin simulaatioihin. Kappaleet voivat pudota toistensa läpi tai jouset voivat ponkaista itsensä äärettömyyksiin. Jotta pelin simulaatio olisi stabiili sitä täytyy ajaa kiinteissä ajanjaksoissa. Ajanjakson pituus on suoraan verrannollinen simulaation tarkkuuteen. Stabiilin simulaation saavuttamiseksi simulaation pääsilmukan tulisi toimia yleensä noin 30–60 Hz:n taajuudella. Esimerkkikoodissa 4 on lopullinen ohjelma, joka toimii stabiilisti halutulla tarkkuudella suorituskykyriippumattomasti.

```
#define FIXED_DT (1/30) // 30 Hz

float lastTime = getCurrentTime();
float timeAcc = 0, totalTime = 0;
while( running ) {
          float newTime = getCurrentTime();
          float dt = newTime – lastTime;
          lastTime = newTime;
          timeAcc += dt;

          while(timeAcc >= FIXED_DT) {
                    physics(FIXED_DT);
                    timeAcc -= FIXED_DT;
                    totalTime += FIXED_DT;
          }

          float drawProg = timeAcc / FIXED_DT;
          draw( drawProg );
}
```

Esimerkkikoodi 4. Stabiili suorituskykyriippumaton ohjelma.

Esimerkkikoodissa 4 fysiikkasimulaatio toimii nyt stabiilisti ja eri taajuudella kuin piirto. Mikäli edellisen ruudun päivityksen jälkeen kulunut aika onkin pitkä, ei mitään jää simuloimatta, vaan ohjelma simuloi fysiikkaa aina vakioajanjaksoissa. Piirtokutsulle on myös lisätty parametri *drawProg,* jonka avulla voidaan visualisoida simulaation välivaiheita. Tämä mahdollistaa sulavat animaatiot pienilläkin simulointitaajuuksilla. [4.]

3.5 Käyttäjän syötteen lukeminen

Käyttäjältä tuleva syöte on myös hyvä muuntaa peliä varten abstraktimpaan muotoon. Sen sijaan että peli lukee jonkin tietyn näppäimen painallusta, pelille riittää yksinkertaistetumpi käsky, kuten esimerkiksi *vasemmalle* tai *oikealle.* Nämä käskyt voivat olla sidottuina eri näppäimiin eri laitteilla. Tämä vaatii sen, että jokaiselle samantyyppiselle laitteelle on luotuna omat näppäin- ja toimintosidokset. Sidosten ei myöskään tarvitse olla välttämättä aina näppäimiä. Esimerkiksi sen sijaan, tiedustellaan

jonkin tietyn näppäimen painallusta kuten esimerkkikoodi 5:n ensimmäisessä ehtolauseessa, kannattaa näppämet ennemmin abstrahoida yleisiksi komennoiksi.

```
// sen sijaan että syötettä tiedustellaan näppäinkohtaisesti..
if ( input_isKeyPressed(KEY_X) ) do_something();

// ..kannattaa syötteet abstrahoida esimerkiksi:
if ( input_getInput(LEFT) ) do_something();
```

Esimerkkikoodi 5. Syötteen abstrahointi.

Muuttujalla *LEFT* voi tarkoittaa jollain laitteella esimerkiksi näppäintä 4 tai jollain laitteella se voi tarkoittaa sitä että käyttäjä painoi kosketusnäytön vasenta laitaa. Kohdistimen sijaintia varten voidaan tehdä esimerkiksi esimerkkikoodi 6:n mukainen rajapinta.

```
boolean input_isDown(unsigned int i);
int input_getPointerX(unsigned int i);
int input_getPointerY(unsigned int i);
```

Esimerkkikoodi 6. Esimerkkiprototyypit kohdistimelle ja kosketuksille.

Kohdistimelle ja kosketusnäytön kosketukselle on mahdollista tehdä yhteinen rajapinta. Funktioille on lisätty parametri *i,* koska kosketusnäytöt tukevat nykyään useita kosketuksia. Muuttuja *i* on välillä $0 < i \leq n$, jossa *n* on hetkellinen aktiivisten kosketusten määrä. Parametri *i* on luonnollisesti hiirellä ohjatessa aina nolla.

4 EMERGE-pelimoottori

EMERGE on Mr. Goodlivingin kehittämä alustariippumaton pelimoottori, jota on käytetty yli 70 pelin kehityksessä. Pelimoottoria ei ole kaupallistettu, eli se ei ole julkisesti saatavilla vaan se on ainoastaan Real Networksin sekä sen tytäryhtion Mr. Goodlivingin omistuksessa. Kerron tässä luvussa pintapuolisesti EMERGEn toiminnasta ja ominaisuuksista.

EMERGE on pelimoottori, joka mahdollistaa pelin kääntämisen tuhansille eri alustoille. Sen toiminta perustuu vahvasti GCC:n esikääntäjään, eli se tukee esikääntökomentoja, kuten *#ifdef* tai *#include*. GCC:n avulla koodi prosessoidaan optimaaliseen muotoon ennen sen varsinaista kääntämistä. Pelimoottorin tarkoituksena ei ole siis tuottaa yhtä ainutta tiedostoa, joka voidaan sittemmin ajaa eri laitteissa. Pääasiallisena syynä tähän on, että juuri tietylle alustalle tai käyttöjärjestelmälle käännetty peli voidaan optimoida huomattavasti pidemmälle, niin muistinkulutuksen kuin suorituskyvynkin kannalta.

Pelimoottorin suurin hyöty on se, että peli voidaan kääntää eri alustoille yhdestä ja samasta koodipohjasta. Pelikoodia ei siis tarvitse muuttaa millään tavalla, kun sitä käännetään eri alustoille. Pelimoottori asettaa pelin ohjelmoinnille tarkat vaatimukset, joita noudattamalla maksimaalinen alustariippumattomuus voidaan säilyttää – mitä enemmän alustoja, luultavasti sitä enemmän on myös rajoituksia ohjelmoinnin suhteen. Mikäli tuettavana on C-kielisten alustojen lisäksi myös Java-kielisiä alustoja, on esimerkiksi C-kielelle ominaisista osoittimista luovuttava. Tämä toimii luonnollisesti myös toisinpäin, eli Javan erikoisominaisuudet eivät ole myöskään käytettävissä.

Muistin hallinta

Java sekä Applen käyttämä Objective-C tukevat molemmat automaattista muistin hallintaa eli ns. roskankeruuta, mutta C/C++ -pohjaiset laitteet taas eivät. Joten pelikoodiin on lisättävä funktiokutsut muistin vapauttamiselle (vaikka sitä ei kaikilla kielillä tarvittaisikaan). Muistin varaukset ja vapautukset tapahtuvat abstraktin rajapinnan kautta. Esimerkiksi funktio, jonka tarkoituksena on vapauttaa varattua muistia käännetään Javalle vain tyhjänä funktiona ja C-kielisille ohjelmille *free()*- tai *delete* -kutsuna. [6.]

Laiteprofiilit

Laiteprofiilit toimivat hyvin olennaisessa roolissa EMERGEn toiminnan kannalta. Jokaiselle alustalle on lisättävä laiteprofiili, ennen kuin sille voidaan kääntää pelejä EMERGEllä. Profiilista selviää muun muassa laitteen resoluutio, resoluutioluokka, suorituskyky ja lukuisia tunnisteista tuetuista ominaisuuksista: esimerkiksi tukeeko laite alpha-kavavallisia kuvia, tai onko laitteessa kosketusnäyttö ja onko se kenties monikosketusnäyttö. Suorituskyky testataan Mr. Goodlivingin kehittämällä työkalulla, joka mittaa laitteen muistia, prosessorin tehoa ja piirron tehokkuutta.

Ohjelmointi

Esimerkkikoodissa 7 on nähtävissä yksinkertainen esimerkki EMERGE-ohjelmasta. Koodi muistuttaa hieman yksinkertaistettua C-kieltä. Koska EMERGE tukee GCC:n esikääntökomentoja voidaan ohjelmassa käyttää esimerkiksi käskyjä *#include* ja *#define*. Esimerkistä on myös hyvin nähtävissä, miten kokonaislukutaulukko on abstrahoitu muotoon *int_array* kielten välisten erojen vuoksi.

```
#include “common.cjavai”
#define FP 10

global int shipX, shipY;
global int_array myArray;
// Java: int_array -> int[] myArray;
// C/C++: int_array -> int* myArray;

global void loadData() {
           int_array_alloc( myArray, 10 );
           for (int i=0; i<array_length(myArray); i++)
               myArray[i] = getSomeValue(i);
}

global void unlodData() {
           array_free( myArray );
}

global void game_paint() {
           renderBackground();
           gfx_drawImage_SPACESHIP(shipX>>FP, shipY>>FP,
                                   ALIGN_LT, 0);
}
```

Esimerkkikoodi 7. Yksinkertainen ohjelmaesimerkki EMERGEllä.

Optimointi

GCC-esikääntäjä hyödyttää myös hyvin paljon pelien optimoinnissa ja skaalautuvuudessa. Optimoitaessa esikääntäjää käytetään, kun pelistä halutaan karsia ominaisuuksia pois. Peli kehitetään aina ensisijaisesti parhaalle tuetulle alustalle. Kun peli on valmis ja toimii parhaalla laitteella, alkaa varsinainen optimointi muille laitteille. Optimointi tapahtuu niin, että pelille tehdään lukuisia ”optimointilippuja” sen keventämiseksi, kuten esimerkiksi DROP_BACKGROUND_DETAIL_X. Edellä mainittu implementoidaan niin, että taustayksityiskohdista vastaava koodi ja kaikki siihen liittyvät kuvat karsitaan lopullisesta käännöstä pois (esimerkkikoodi 8). Tämänkaltaisia ehtoja tehdään niin monta kunnes lopullisen käännetyn pelin koko on saavuttanut 64 kilotavua tai alle. Nämä ”optimointiliput” järjestetään pelille sopivalla heuristiikalla

tärkeysjärjestykseen ja tarvittavat liput tietylle laitteelle saadaan sitten laskemalla larvo näiden kahden ääripään väliltä. Arvo voidaan laskea käyttämällä esimerkiksi lineaarista interpolointia. Yleensä liput jaotellaan muistinkulutuksen, suorituskyvyn ja piirtotehokkuuden mukaisiin ryhmiin. [10.]

```
global void drawBackground() {
          gfx_setColor(BACKGROUND_COLOR);
          gfx_fillRect(0, 0, X_RES, Y_RES);
#ifndef DROP_BACKGROUND_VEGETATION
          for ( int i=0; i<10; i++ ) {
                    int x = getVegetationX(i);
                    int y = getVegetationY(i);
                    gfx_drawImage_VEGETATION(x, y, ALIGN_LT, 0);
          }
#endif
}
```

Esimerkkikoodi 8. Pelikoodin optimointia.

5 Vancouver 2010 -mobiilipeli

Tässä luvussa käyn läpi Vancouver 2010 mobiilipelin kehityksen. Millainen pelinkehitysprosessi oli ja miten alustariippumattomuus vaikutti pelinkehitykseen? Toimin projektissa peliohjelmoijana ja päävastuualueenani oli lumilautailun ja pikaluistelun ohjelmointi. Näiden lisäksi tein peliin myös paljon muuta geneeristä ohjelmointia, kuten pelitilanteen tallennus ja lataus, suoritusten nauhoitus ja näyttäminen uusintana sekä piirtojärjestyksestä huolehtiva piirtolista.

5.1 Peliprojektin prosessi

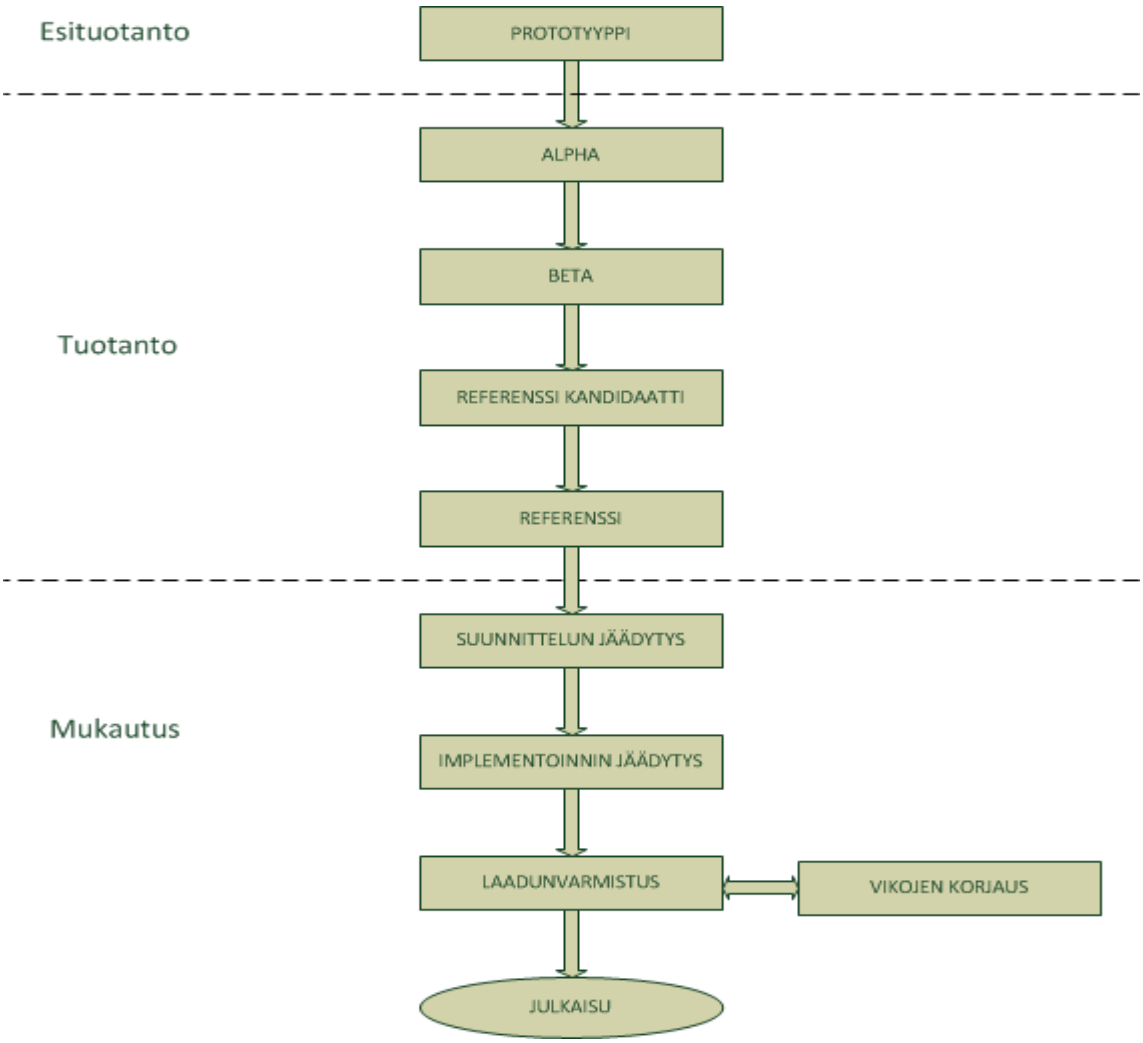
Vancouver 2010 mobiilipeliprojekti seurasi Mr.Goodliving normaalia peliprojekteille suunniteltua prosessimallia, joka jakautuu kolmeen päävaiheeseen: esituotantoon, tuotantoon ja mukautukseen. Esituotantovaihe käsittää lähinnä pelin prototyypin tekemisen ja pelin suunnitelman tekemisen. Tuotannossa tuote kehitetään ja viimeistellään parhaalle mahdolliselle laittelle, eli ns. referenssilaitteelle. Viimeisenä on mukautusvaihe, jossa peli skaalataan ja optimoidaan muille laitteille.

Prototyypissä keskitytään pelin mekaniikkaan ja pyritään saamaan pelimekaniikasta mahdollisimman hauska. Myös pelin alkuperäinen suunnitelma voi muuttua vielä paljon tässä vaiheessa. Prototyypin tekeminen kestää yleensä kahdesta viikosta kuukauteen. Kun prototyyppi on todettu toimivaksi ja hauskaksi, voidaan siirtyä esituotannosta varsinaiseen tuotantoon. Tuotannon ensimmäisenä virstanpylväänä on alpha-versio, johon on tarkoitus saada paikoilleen pelin tärkeimmät perusosat. Tässä vaiheessa pelissä on vielä hyvin paljon tilapäistä grafiikkaa ja toiminnallisuutta, joka korvataan myöhemmissä vaiheissa lopullisilla versioilla.

Tämän jälkeen beta-vaiheeseen valmistellaan pelin loputkin ominaisuudet, kuten valikot ja muut tärkeimmät oheisominaisuudet. Hieman ennen varsinaista referenssivaihetta on referenssiin valmistava vaihe eli referenssikandidaattivaihe. Tässä vaiheessa pelissä ei tulisi olla enää yhtään tilapäistä grafiikkaa tai varsinaisesti keskeneräisiä ohjelmatoteutuksia ja pelin tulisi toimia moitteettomasti referenssilaitteella. Referenssilaitteeksi valitaan yleensä laite, joka kuuluu sillä hetkellä markkinoiden tehokkaimpiin laitteisiin. Peliä testautetaan laajemmalla yleisöllä.

Testauksissa löytyneitä vikoja korjaillaan ja mekaniikoita kehitetään pelin saaman palautteen perusteella. Vasta tämän jälkeen peli voi edetä varsinaiseen referenssiin, jossa pelin tulisi olla teoriassa valmis referenssilaitteelle. Tähän asti pelinkehityksessä ei ole missään vaiheessa varsinaisesti kiinnitetty huomiota pelin skaalautuvuuteen, muutoin kuin noudattamalla EMERGEn perussääntöjä ohjelmoitaessa. Näin ollen ohjelmassa ei ole mitään viittauksia vain C-kielellä tai vain Javalla toimiviin toteutuksiin.

Viimeisimpänä on mukautusvaihe, jossa keskitytäänkin lähinnä pelin optimointiin ja skaalautuvuuten. Mukautuksen ensimmäinen vaihe on ”suunnittelun jäädytys”, jonka jälkeen peliin ei enää suunnitella uusia ominaisuuksia. Joitakin ominaisuuksia on voitu päättää tehtäväksi juuri ennen tätä vaihetta ja niiden implementointia voidaan vielä jatkaa. Tämän jälkeen on ”toteutuksen jäädytys” -vaihe, jossa pelin kaikki ominaisuudet tulee viimeistään olla toteuttettuna. Loppuaika ennen julkaisua käytetään pelin optimointiin, skaalautuvuuteen, laadunvarmistukseen ja vikojen korjailuun. Kuviossa 4 on nähtävissä kokonaisvaltainen kuva koko kehitysprosessista.



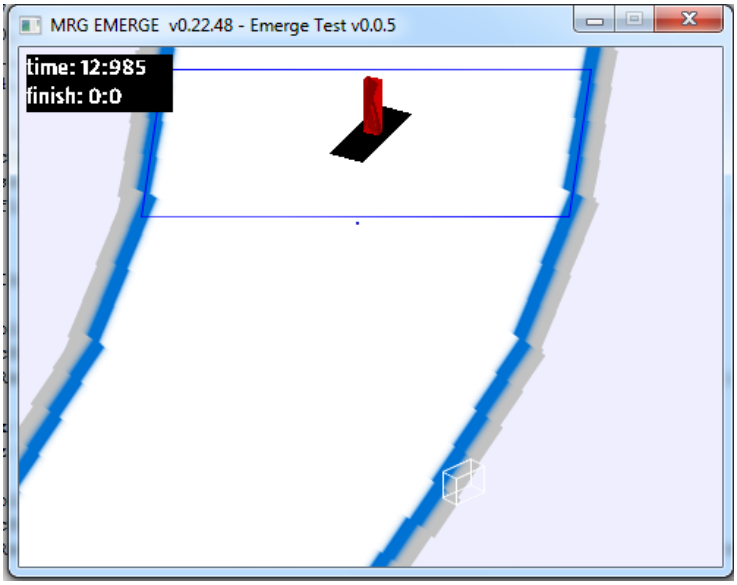
Kuvio 4. Pelinkehitysprosessi.

5.2 Esituotanto

Peliprojekti aloitettiin vuonna 2009 toukokuussa. Pelin kehittäminen alkoi luonnollisesti prototyyppien kehittämisellä. Peliin oli tarkoitus tehdä viisi lajia – pikaluistelu, hiihto, lumilautailu, ampumahiihto ja kumparelasku. Näistä kahteen oli tarkoituksena kehittää täysin uudenlaiset pelimekaniikat. Muissa lajeissa hyödynnettiin Mr.Goodlivingin aikaisemman Playman Sports -pelisarjan hyväksi todettuja pelimekaniikoita. Playman Sportsin juoksemiseen suunniteltua pelimekaniikkaa pystyttiin hyödyntämään pienillä muokkauksilla hiihdossa, ampumahiihdossa sekä kumparelaskussa. Lumilautailuun ja pikaluisteluun sen sijaan oli tarkoituksena kehittää täysin uudet pelimekaniikat. Peli seurasi myös tyylillisesti hyvin vahvasti Playman Sports -sarjaa, eli se oli tyyliltään

hyvin helposti lähestyttävä grafiikaltaan ja pelattavuudeltaan. Näin pelillä oli mahdollisuus laajaan ja moni-ikäiseen käyttäjäkuntaan.

Pelimekaniikkojen prototyypit kehitetään useimmiten vain PC:llä, ellei prototyypin kannalta ole erityisen oleellista, että sitä testataan jollain tietyllä laitteella ensin. Useimmiten pelimekaniikkaprototyypit näyttävät hyvin karuilta, esimerkkinä tästä kuviossa 5 on nähtävissä lumilautailun varhainen prototyyppi. Prototyypissa testataan isometristä projektiota ja lumilautailijan ohjausta. Kuvion 5 mukainen kuvakulma edestä osoittaui toimivaksi ja sitä päätettiin käyttää sittemmin myös pelin tuotantoversiossa.



Kuvio 5. Lumilautailun prototyyppi.

5.3 Kilpalajit

Hiihto

Tavalliseksi hiihdoksi valittiin lyhyen matkan hiihto suoralla radalla. Matkaksi asetettiin 100 metria, jotta yhden pelikerran kesto pysyisi sopivan lyhyenä. Hiihdossa hyödynnettiin Playman Sportsin juoksussa käytettyä pelimekaniikkaa. Mekaniikka toimii niin, että nappuloita ilmestyy ruudun vasempaan ja oikeaan laitaan yksi kerrallaan satunnaisesti. Näitä nappuloita on tarkoitus painaa mahdollisimman nopeasti. Mitä nopeammin nappulaa painetaan sen ilmestymisen jälkeen, sitä enemmän hiihtäjä saa vauhtia. Mikäli pelaajan painallus kohdistuu taas väärälle puolelle, hiihtäjä menettää vauhtiaan. Kuviossa 6 on nähtävissä kuvakaappaus hiihdosta.

Kuvio 6. 100 metrin hiihto.

Pikaluistelu

Pikaluistelussa luisteltava matka on 500 metriä. Pelissä luistellaan kaksi täyttä kierrosta, joten radan pituudeksi on asetettu 250 metriä. Pikaluistelussa pelaajan on tarkoitus täyttää kahta ympyrän muotoista voimamittaria, jotka edustavat luistelijan jalkojen ponnistusvoimaa. Ponnistusvoimaa indikoidaan ympyrän ulkokehällä liikkuvalla osoittimella. Mittareita on kaksi, ja ne on aseteltu ruudun vasempaan sekä oikeaan laitaan. Mittareita täytetään vuorottain. Tämä tapahtuu painamalla vuorotellen vasemmalle tai oikealle riippuen siitä, kumpi on sillä hetkellä aktiivinen. Kosketusnäytöllisissä laitteissa painetaan ruudun vasenta tai oikeaa laitaa. Mittareiden täyttymisnopeus vastaa likimain luistelijan potkujen nopeutta. Aluksi siis mittarit täyttyvät hyvin nopeasti ja vauhdin kasvaessa täyttymisnopeus hidastuu. Mittarin ulkokehällä liikkuva osoitin on tarkoitus saada kiertämään mahdollisimman lähelle täyttä kierrosta eli 360:tä astetta. Mikäli mittari täyttyy yli täyden kierroksen, on painallus virheellinen eikä luistelija saa potkuunsa lainkaan voimaa. Kuviossa 7 on nähtävissä kuvakaappaus luistelusta.

Kuvio 7. 500 metrin pikaluistelu.

Ampumahiihto

Ampumahiihdon hiihto-osuudessa on sovellettu samaa pelimekaniikkaa kuin tavallisessa hiihdossa. Tavallinen hiihto-osuus on nähtävissä kuviossa 8. Kuviossa 9 on nähtävissä kuvakaappaus ampumaosuudesta. Pelaajan on painettava taululle ilmestyvän nuolen osoittamaan suuntaan mahdollisimman nopeasti. Ohi menneistä laukauksista sakotetaan 5 sekuntia.

Kuvio 8. Ampumahiihdon hiihto-osuus

Kuvio 9. Ampumahiihdon ampumaosuus.

Lumilautailu

Lumilautailussa tarkoituksena on yksinkertaisesti selvittää mutkitteleva rata ja päästä ensimmäisenä maaliin. Liian jyrkistä käännöksistä sakotetaan vauhdin menetyksellä. Myös mäen suunnasta poikkeaviin suuntiin laskeminen on huomattavasti hitaampaa. Mäen suunta on pelin ruudulla suoraan alas. Lumilautailu tukee myös kiihtyvyysanturiohjausta, eli peliä voidaan pelata myös kallistelemalla kiihtyvyysanturilla varustettua laitetta. Kuviossa 10 on nähtävissä kuvakaappaus lumilautailusta.

Kuvio 10. Lumilautailu.

Kumparelasku

Myös kumparelaskussa on hyödynnetty samoja mekaniikoita kuin hiihdossa ja ampumahiihdossa. Se kuinka nopeasti kumpareista selviää, määräytyy sattunaisesti ilmestyvien nappuloiden painallusten nopeudesta. Hyppyrissä taas on käytetty samaa mittaria kuin luistelussa. Mitä paremmin hyppy onnistuu, sitä pidemmälle pelaaja hyppää ja saa täten pidemmän etumatkan vastustajaansa. Kuviossa 11 on nähtävissä kuvakaappaus kumparelaskun hypystä.

Kuvio 11. Kumparelasku.

5.4 Pelikoodin rakenteesta

Tuettujen alustojen määrän maksimoimiseksi pelissä ei käytetä ohjelmointikielien sisään rakennettuja olio-ominaisuuksia laisinkaan. Sen sijaan toteutimme esimerkkikoodin 9 tapaisen yksinkertaisen ja abstraktin objektityypin.

```
BEGIN_OBJECT(OlioX) {
           int       kokonaisluku;
           float     liukuluku;
           int_array kokonaislukuTaulukko;
} END_OBJECT(OlioX);
```

Esimerkkikoodi 9. Yksinkertaistettu objekti.

Objektityypin todellinen toteutus määräytyy sen alustan mukaan, jolle se käännetään. Joidenkin alustojen rajoitteiden vuoksi tämä objektityyppi ei saa sisältää funktioita eikä toisten objektien instansseja. Se saa sisältää ainoastaan perustyyppejä sekä perustyyppien taulukoita. Poikkeuksena nämä objektit saavat tosin sisältää *String*-merkkijonoluokan instansseja, sillä tämä on sisäänrakennettu ja järjestelmien välillä siirtyvä luokka toteutettuna EMERGEssä.

Yksinkertaistetun objektirakenteen vuoksi pelissä ei voida suoranaisesti hyödyntää olio-ohjelmoinnin etuja. Joitakin olio-ohjelmoinnin idoita pyritään kuitenkin hyödyntämään pelissä jollain tasolla selkeyden vuoksi. Nyrkkisääntönä on, että yksi kooditiedosto vastaa yhtä luokkaa ja kaikille kooditiedoston funktioille lisätään etuliitteeksi tiedoston nimi. Esimerkkikoodissa 10 on lyhyt ote pelikoodista, joka havainnollistaa pelissä käytettyä ohjelmointimallia.

```
BEGIN_OBJECT(WG2Character){
           String    name;
           byte      head, flag;
           int_array skills;
}END_OBJECT(WG2Character);

global object_array(WG2Character) wg2character_list;

global void wg2character_load(){ /*…*/ }
global void wg2character_free(){ /*…*/ }
```

Esimerkkikoodi 10. Ote wg2character.cjavai tiedostosta.

Useimmiten näille pelin luokkarakenteille toteutetaan funktiot *load(),* ja *free(),* jotka vastaavat luokan muodostinta ja hajotinta. Näitä on kuitenkin aina kutsuttava eksplisiittisesti pelikoodista. Yleensä nämä pelin luokat sisältävät itsessään listan oman luokkansa instansseja. Jos muut luokat haluavat osoittaa näihin objekteihin, se tapahtuu yleensä funktioiden kautta tai tekemällä suoria osoituksia itse taulukkoon.

Mikäli halutaan selkeyttää sitä, että luokalla on osoitin johonkin toisen luokan instanssiin, se voidaan toteuttaa esimerkkikoodin 10 tavalla:

```
// MyObjectX.cjavai
#define MyObjectY_Ptr int
BEGIN_OBJECT(MyObjectX){
           MyObjectY_Ptr ptr_objy;
}END_OBJECT(MyObjectX);

// MyObjectY.cjavai
object_array(MyObjectY) MyObjectY_list;
global int MyObjectY_getObjDataX(int i) {
           ASSERT(i >= 0 && i < array_length(MyObjectY_list));
           return MyObjectY_list[i].dataX;
}
```

Esimerkkikoodi 10. Objektityypin käyttäminen olio-ohjelmalliseen tapaan.

Kuten esimerkkikoodissa 10 näemme, toisiin luokkiin osoittaminen on tehty vain ns. kosmeettiseksi. Tämä kuitenkin selkeyttää koodia huomattavasti ja toiset ohjelmoijat näkevät suoraan mihin objektiin tällä kokonaisluvulla halutaan viitata. Esimerkkikoodista 10 myös nähdään, ettei tietoa haeta suoraan listasta, vaan se tehdään funktiokutsun kautta, jossa tehdään asianmukainen parametrin tarkistus. Tällöin mahdolliset virheet saadaan aikaisessa vaiheessa korjattua.

5.5 Skaalautuvuus

Peliä voidaan skaalata korkealla tasolla kolmella eri parametrilla, joissa maksimiarvo sulkeissa tarkoittaa, että peli optimoidaan maksimaalisesti kyseisen parametrin mukaan

- VC_DROPFEATURES_MEMORY (0 - 10)
- VC_DROPFEATURES_SIZE (0 - 10)
- VC_DROPFEATURES_SPEED (0 - 10)

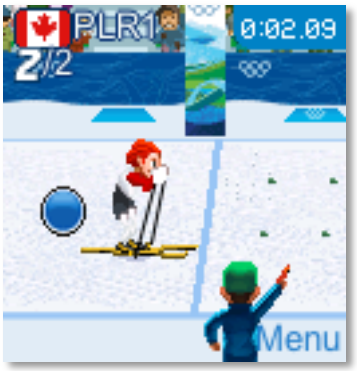
Näiden lisäksi pelille löytyy lukuisia muita parametreja, joilla peliä voi mukauttaa tarkemmin jollekin tietylle alustalle. Optimointiliput on selitetty yleisellä tasolla tarkemmin luvussa 4.5.

Peli tuki loppujen lopuksi hieman yli tuhatta mobiililaitealustaa ja se jakautui viiteen eri resoluutioluokkaan R2–R6. Kuviot 12–15 ovat esimerkkeinä siitä, kuinka sama pelitilanne skaalautuu eri laitteille.

Kuvio 12. Resoluutioluokka 5 – iPhone.

Kuvio 13. Resoluutioluokka 4 – Sony Ericcsson C510.

Kuvio 14. Resoluutioluokka 3 – Motorola Razor.



Kuvio 15. Resoluutioluokka 2 – Nokia 3200.

Kuvissa edellä on muutettu ainoastaan pelin resoluutioluokkaa, eli pelin muita ominaisuuksia ei siis ole jätetty pois. Kuviossa 16 on nähtävissä pelin täysin karsittu versio.

Kuvio 16. Täysin karsittu versio pelistä (kuva on selkeyden vuoksi otettu resoluutioluokasta 5). Kuva otettu täysin samasta kohtaa kuin kuviot 6–10.

Kuvassa 16 lähes kaikki mahdollinen on karsittu pelistä pois. Jäljellä on enää vain pelimekaniikat ja pakolliset grafiikat. Mikäli peli käännetään tähän karsittuun muotoon sekä resoluutioluokkaan 2, pelipaketin lopulliseksi kooksi saadaan noin 64 kilotavua. Jos peli taas käännetään ilman mitään karsintoja resoluutioluokkaan 6, sen koko on noin 20 megatavua. Peli vie levytilaa pienimmillään siis vain 0,32 prosenttia sen täydellisestä koosta.

Sen lisäksi että peli skaalautuu erittäin tehokkaasti kokonsa puolesta, se on myös hyvin muuntautumiskelpoinen laitteiden erilaisten ohjaustapojen välillä. Pelissä saavutettin pelattavuus niin kosketusnäytöillä kuin näppäimilläkin. Ainoat, jotka eivät soveltuneet hyvin pelaamiseen, olivat kapasitiiviset kosketusnäytöt, joita ohjattiin stylus-kynällä. Näitten laitteiden kohdalla hyödynnettiin joitakin peliä helpottavia parametrejä, jotta tulokset olisivat vertailukelpoisia muilla laitteilla pelattujen tulosten kanssa.

6 Yhteenveto

Vancouver 2010 -mobiilipeli on osoittanut, että markkinoiden laajasta fragmentoitumisesta huolimatta useita alustoja voidaan tukea kustannustehokkaasti. Ja ennen kaikkea hyvän pelin tekeminen tässä ympäristössä on mahdollista.

Alustariippumattoman pelin kehityksessä on huomioitava lukuisia asoita. Asiat on hyvä huomioida mahdollisimman aikaisessa vaiheessa, sillä pienetkin muutokset saattavat osoittautua myöhemmissä vaiheissa erittäin työläiksi. Alustariippumattoman pelin tärkeimpiä ominaisuuksia ovat resoluutioriippumattomuus ja suorituskyky-riippumattomuus. Nämä ominaisuudet pätevät vaikka käyttöjärjestelmä pysyisikin samana, sillä erot tulevat monesti täysin itse laitteesta. Vaikka kehitys tapahtuisikin esimerkiksi vain Applen laitteille on huomioitava, että näidenkin laitteiden väliltä löytyy myös merkittäviä eroja sekä resoluution että suorituskyvyn suhteen. Pelin on myös hyvä skaalautua kokonsa ja muistin kulutuksensa puolesta, sillä tallennuskapasiteetti ja käytettävän muistin määrä vaihtelee myös paljolti eri laitteiden välillä.

Jos olio-ohjelmoinnista on esimerkiksi luovuttava, ohjelmoitaessa on sovittava tarkat pelisäännöt. Tarkoin sovittuja sääntöjä ja tiettyjä olio-ohjelmoinnista sovellettuja ideoita noudattamalla on mahdollista ylläpitää selkeää ja toimivaa koodia. Tämän tyyppinen koodi on kuitenkin hyvin altis ei-toivotuille ratkaisumalleille ja ohjelmaan livahtaakin toisinaan rakenteellisesti hyvin epäoptimaalista koodia. Tällaisessa ympäristössä ohjelmointi soveltuu hyvin pienempiin projekteihin, joissa työskentelee kahdesta neljään ohjelmoijaa. Koodin hallinta ja rakenteellisuuden ylläpitäminen tällaisessa ympäristössä voi toisinaan osoittautua haasteelliseksi.

Lähteet

1 Top five smartphone vendors. 2011. Verkkodokumentti. cnet News. <http://news.cnet.com/8301-13579_3-20030831-37.html>. 7.2.2011. Luettu 12.3.2011.

2 Gartner Says Worldwide Mobile Phone Sales Grew 17 per cent in First Quarter 2010. 2010. Verkkodokumentti. Gartner. <http://www.gartner.com/it/page.jsp?id=1372013>. 19.5.2010. Luettu 17.3.2011.

3 Alustariippumattomuus. 2011. Verkkodokumentti. Wikipedia.<http://fi.wikipedia.org/wiki/Alustariippumattomuus>. 30.1.2011. Luettu 18.3.2011.

4 Fix Your Timestep. 2006. Verkkodokumentti. Glenn Fiedler's Game Development Articles and Tutorials. <http://gafferongames.com/game-physics/fix-your-timestep>. 2.9.2006. Luettu 20.3.2011.

5 Games Leading Mobile Apps. 2010. Verkkodokumentti. Marketing Charts (The Nielsen Company). <http://www.marketingcharts.com/direct/games-leading-mobile-apps-14127/nielsen-mobile-apps-sept-2010jpg/>. 1.8.2010. Luettu 13.2.2011.

6 Prata, Stephen. 2005. C++ Primer Plus. 5th ed. Indianapolis: Sams.

7 Matkapuhelin. 2011. Verkkodokumentti. Wikipedia. <http://fi.wikipedia.org/wiki/Matkapuhelin>. 20.4.2011. Luettu 2.5.2011.

8 Mobiilipeli. 2011. Verkkodokumentti. Wikipedia. <http://fi.wikipedia.org/wiki/Mobiilipeli>. 6.3.2011. Luettu 2.5.2011.

9 Mr. Goodliving. 2011. Verkkodokumentti. Wikipedia. <http://en.wikipedia.org/wiki/Mr.Goodliving>. 10.2.2011. Luettu 2.5.2011.

10 Lineaarinen interpolointi, 2011. Verkkodokumentti. Isoverkosto.fi. <http://pooli.isoverkosto.fi/fi/pooli/04_matematiikka/maa/kurssit/maa12/kurssipohja/lineaarineninterpolointi.htm>. 2004. Luettu 2.5.2011.

Lumilautailun pelimekaniikka

Lumilautailijaa kuvataan pisteenä 3D-avaruudessa jolla on suunta $\widehat{D}_b$ ja skalaarinopeus v_b ja skalaarikiihtyvyys a_b. Täten nopeusvektori on suuntavektorin ja skalaarinopeuden tulo. Lautailija pyrkii kääntämään suuntavektorinsa $\widehat{D}_b$:n yhdensuuntaiseksi käyttäjän kääntelemän suuntavektorin $\widehat{D}_i$:n kanssa. Lautailijan suuntaa päivitetään jokaisella fysiikka-askelmalla seuraavanlaisesti (huom. yhtälöissä ei ole huomioitu aikaa sillä fysiikkaa simuloidaan vakioajanjaksoissa)

$$D_n = \widehat{D}_i * k_s + \widehat{D}_b * (1 - k_s) \quad (1)$$

D_n on lumilautailijan uusi laskettu suuntavektori (normalisoitava)
$\widehat{D}_i$ on käyttäjän ohjailema suuntavektori
$\widehat{D}_b$ on lumilautailijan hetkellinen suuntavektori
k_s on lautailijan kääntymisen hetkellinen nopeuskerroin

Kääntymisnopeuskerroin lasketaan taas seuraavalla tavalla

$$k_s = \frac{fric_{lat}}{(fric_{lat} + v_b + 1)} \quad (2)$$

$fric_{lat}$ on laudan sivuttaissuuntainen kitkakerroin
v_b on lumilautailijan hetkellinen skalaarinopeus

Yhtälössä 1 lautailijan suuntavektori $\widehat{D}_b$ lähestyy asymptoottisesti käyttäjän ohjaamaa suuntavektoria $\widehat{D}_i$. Kääntymisnopeus määräytyy nopeuskertoimen k_s mukaan, joka muuttuu lautailijan nopeuden v_b funktiona. Kääntymisnopeutta voidaan säätää parametrin $fric_{lat}$ avulla. Lautailijan kiihtyvyys saadaan seuraavan yhtälön avulla

$$a_b = (\widehat{D}_i \cdot \widehat{D}_b) * slope \quad (3)$$

a_b on lumilautailijan hetkellinen skalaarikiihtyvyys
$slope$ on mäen jyrkkyyskerroin