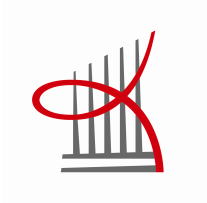


Pelin yhteensopivuus Java-mobiilialustojen välillä

Herkko Noponen

Opinnäytetyö
Toukokuu 2011
Tietotekniikan koulutusohjelma
Ohjelmistotekniikka
Tampereen ammattikorkeakoulu

TAMPEREEN AMMATTIKORKEAKOULU

Tampere University of Applied Sciences

Tampereen ammattikorkeakoulu
Tietotekniikan koulutusohjelma, ohjelmistotekniikka

Tekijä	Herkko Noponen
Työn nimi	Pelin yhteensopivuus Java-mobiilialustojen välillä
Sivumäärä	30
Valmistumisaika	toukokuu 2011
Työn ohjaaja	lehtori Jari Mikkolainen
Työn tilaaja	Cape Peace Software Oy, ohjaajana Toimitusjohtaja Tommi Lukkarinen

Tiivistelmä

Cape Peace Software Oy lähti tekemään 2010 kesällä mobiililaitteisiin peliä. Alkuperäinen peli kehitettiin Java ME alustalle. Jo valmiin Java-ohjelmakoodin vuoksi sovitukset BlackBerry ja Android mobiilialustoille katsottiin tärkeimmiksi.

Tässä työssä esitellään alkuperäisen Java ME pelin idea, arkkitehtuuri ja sovituksia varten luotu rajapinta. Pelisovelluksen Android ja BlackBerry sovitukset luotiin käyttämällä Eclipse-kehitysympäristöä. Molempia mobiilialustoja varten asennettiin valmistajien tarjoamat liitännäiset Eclipseen. Tärkeimmät käytössä olleet ominaisuudet olivat laite-emulaattorit ja vianjäljittimet(debugger).

Ongelmia tuli alkuperäisen Java ME pelin samanaikaisesta kehityksestä sovitusten kanssa. Työssä kerrotaan myös ongelmista Android ja BlackBerry alustojen tuomista ongelmista.

Avainsanat Android, BlackBerry, Java ME, porttaus, sovittaminen

Tampere University of Applied Sciences, Bachelor's Degree
Department of Computer Systems Engineering, Software Engineering

Writer	Herkko Noponen
Thesis	Portability of a game between mobile Java-platforms
Pages	30
Graduation time	may 2011
Thesis supervisor	lecturer Jari Mikkolainen
Co-operating company	Cape Peace Software Oy, Chief Executive Officer Tommi Lukkarinen as supervisor

Abstract

In summer 2010 Cape Peace Software Oy began to create a game for mobile devices. The original game was developed for Java ME plaftorm. Due to the original Java-code already in place, ports for Android and BlackBerry were deemed as the most important targets.

This thesis will represent the architecthure of the original Java ME game and the interface created for the ports. The games BlackBerry and Android ports were created by using Eclipse IDE (Integrated Development Environment). For both platforms, there was an Eclipse plugin which were installed. The most important features of these plugins were their device-emulators and debuggers.

Problems arose as the original Java ME game was under development at the same time as the ports. This work also tells about the problems Android and BlackBerry platforms brought into the work.

Keywords Android, BlackBerry, Java ME, porting, ports

Sisällysluettelo

1 Johdanto

Cape Peace Software Oy lähti kehittämään peliä mobiililaitteisiin Java ME-alustalla. Tarkoituksena oli saada tekijöille kokemusta Java ME alustasta, kehittää yrityksen tietotaitoa ja tehdä mahdollisuuksien mukaan kaupallisesti menestyvä tuote. Tietoa haluttiin myös mobiilialan sovelluskaupoista ja niiden käytöstä.

Tehty peli haluttiin sovittaa mahdollisimman usealle mobiilialustalle. Itse saavuin tässä vaiheessa mukaan tekemään pelistä versiot Android- ja BlackBerry-alustoille. Tullessani mukaan peli ei ollut vielä kokonaan valmis. Sovitusten tekeminen aloitettiin ennen alkuperäisen pelin valmistumista.

Tässä dokumentissa lähdetään liikkeelle luvusta 2 esittelemällä alkuperäinen Java ME sovellus. Aluksi selvitetään mikä Java ME on ja käydään läpi mitä sen sisältämiä tekniikoita on käytetty ja miten ne liittyvät toisiinsa. Pelin idea käydään läpi ja selvennetään miten peli etenee. Luvussa selvennetään myös alkuperäisen pelin komponentteja ja mitä resursseja on käytetty. Myös arkkitehtuurista kerrotaan korkean tason kuvaus. Lopuksi luvussa kerrotaan vielä peliin tehdystä rajapinnasta joka helpottaa sovitusten tekemistä.

Kolmas luku keskittyy käsittelemään BlackBerry sovitusta. Kehitystyökalut käydään läpi ja kerrotaan lyhyesti läpi niiden tärkeimmät toiminnot. Luvussa käydään nopeasti läpi varsinainen sovitustyö. Enemmän kerrottavaa oli vianselvityksestä ja virheiden korjaamisesta. Näitä käydään yksityiskohtaisemmin läpi.

Neljäs luku käy läpi Android sovituksen. Android-alustasta kerrotaan tarvittavia tietoja jotta lukija ymmärtää paremmin sovitustyötä. Kuten BlackBerryn kanssa, Android -kehitystyökalut esitellään. Tärkeänä osana selitetään Android sovellusten elinkaarta ja alustan tapaa käsitellä resursseja. Grafiikalle ja äänelle on omat alalukunsa. Vienselvitykselle ja virheiden korjaamiselle on omistettu myös oma alalukunsa Android-alustalla. Lopuksi käydään vielä paketointi ja allekirjoitus läpi. Sovelluspaketin luontia varten tehdystä komentojonosta kerrotaan viimeiseksi.

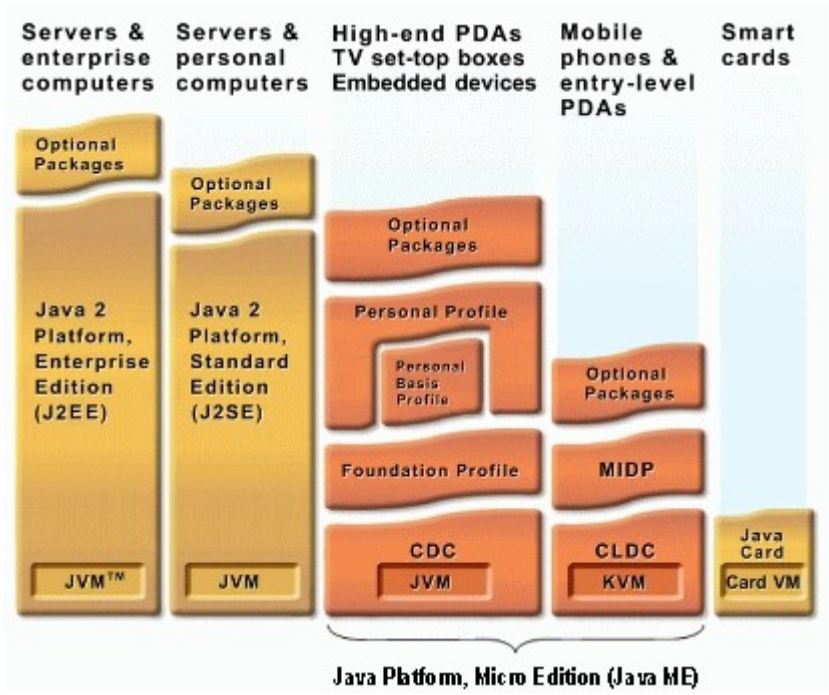
Aivan lopuksi käydään yhteenvetona läpi se, mitä työssä saatiin aikaiseksi. Yhteenvedossa selvennetään myös Android ja BlackBerry alustojen eroja ja työmääriä. Tärkeimmistä työkaluista löytyy myös jotakin kerrottavaa.

2 Java ME-peli

Java ME on alunperin Sun Microsystems -yrityksen luoma Java-alusta sulautetuille järjestelmille ja mobiililaitteille. Oraclen tietojen mukaan (Java ME Technology, Oracle) Java ME alustat perustuvat kolmeen osaan;

- konfiguraatioon, joka tarjoaa peruskirjastot ja virtuaalikoneen ominaisuudet laajalle laiteryhmälle,
- profiiliin, jossa on joukko ohjelmistorajapintoja pienemmälle laiteryhmälle, ja
- vaihtoehtoiseen pakettiin, jossa määritellään teknologia-kohtaiset ohjelmistorajapinnat.

Mobiililaitteille tarkoitettu konfiguraatio on CLDC (Connection Limited Device Configuration). Tämä on suunniteltu laitteille joissa akku, prosessoriteho ja grafiikkaominaisuudet ovat rajoitettuja. Laajalti käytössä on myös MIDP-profiili (Mobile Information Device Profile) joka tarjoaa täyden Java-sovellusympäristön kännyköihin ja vastaavanlaisiin mobiililaitteisiin (kuvio 1).



Kuvio 1: Java ME alusta (Kuva: Oracle)

CLDC ja MIDP pohjaisissa ympäristöissä Java-sovellus toteutetaan laajentamalla ohjelmistorajapinnan tarjoama MIDlet-luokka omaan käyttöön sopivaksi. MIDlet-luokkaa käyttäviä sovelluksia kutsutaan midleteiksi.

Cape Peace Software Oy lähti kehittämään pelisovellusta Java ME alustalle, koska se kattaa suuren osan mobiilimarkkinoilla olevista laitteista. Sovellus on tehty käyttäen CLDC 1.1 ja MIDP 2.0 tarjoamia kirjastoja. Laitetta testattiin enimmäkseen Nokian kosketusnäytöllisillä puhelimilla, joissa oli S60 5th edition. Sovelluksen on raportoitu toimineen myös muiden valmistajien puhelimilla, joissa on CLDC 1.1 ja MIDP 2.0 asetusten määrittelemät microedition Java-kirjastot.

Sovelluksen Java ME versio perustuu MIDlet-luokan laajentamiseen ja *javax.microedition* pakettien grafiikka- ja äänikirjastojen käyttöön. Midletin taustalla pelilogiikka toimii yksinkertaisen Javan-koodin avulla. Koska pelilogiikan ohjelmakoodit eivät käytä erikoisia kirjastoja, voitiin näitä ohjelmakoodeja käyttää myös sovitusten teossa ilman ongelmia.

2.1 Pelin idea

Pelin idea on seuraava. Pelaaja on ulkoavaruudesta maahan hyökkäävä vihreä lonkerohirviö, jonka tarkoitus on vallata ja tuhota maapallon asukkaat. Itse pelissä hirviö pysyy oikeassa ylälaidassa ja taustatähtiä liikuttamalla saadaan optinen illuusio sen liikkumisesta avaruuden halki.

Vastaan tulee maapalloa kiertäviä satelliitteja, jotka saattavat yhtäkkiä ampua ohjuksen hirviötä kohden. Pelaajan täytyy ampua laserilla lähestyvät ohjukset. Tämä tapahtuu koskettamalla kosketusnäyttöä haluttuun kohtaan, jolloin laser laukeaa ja tuhoaa ohjukset, joihin se koskettaa.

Pelin kulkiessa eteenpäin, taistelu kovenee ja ohjusten määrä lisääntyy. Lopussa hirviö saavuttaa maapallon ja sitä kiertävän kuun ja avaruusaseman. Voitto saavutetaan tuhoamalla tarpeeksi monta ohjusta. Häviö vaatii kolme osumaa hirviöön. Kuvasta (kuvio 2) näkyy konsepti aktiivisesta pelitilanteesta.

Kuvio 2: Konseptikuva pelitilanteesta

2.2 Komponentit ja resurssit

Sovellus käyttää aktiivisesti erilaisia kuva- ja äänitiedostoja. Sovelluksen teossa oli mukana ammattigraafikko, joka suunnitteli ja piirsi pelissä käytetyt kuvat. Lähes kaikki sovelluksen piirtämä grafiikka perustuu valmiiksi piirrettyihin kuviin. Sovelluksen piirtämän tekstin fontti on tehty yhdestä kuvasta, johon on piirretty kaikki kirjaimet. Sovellus pilkkoo tämän kuvan pienempiin yhden kirjaimen sisältäviin kuviin ja piirtää halutut tekstit käyttäen näitä pieniä kuvia yksittäisten kirjainten piirtoon ruudulle.

Muut kuvat sovellus lataa muistiin käyttäen koodiin määriteltyä listaa kuvien nimistä. Midletistä kutsutaan drawImage-metodia CapeDevice-rajapinnan kautta, kun halutaan piirtää tietty kuva. Parametrina annetaan halutun kuvan tunnistenumero, jonka mukaan midletissä osataan käyttää kuvataulukosta oikeaa kuvaa.

Kuvat voidaan piirtää myös tiettyyn kulmaan. Kuvista, joita halutaan kääntää lasketaan ohjelman ajon aikana uusi bittikartta halutussa kulmassa, jotta kuvista ei tarvitsisi luoda erikseen valmiita kuvia eri kulmiin. Näin säästetään valmiin sovelluksen viemää tallennustilaa. Kuvia on sekä PNG- että JPG-formaatissa. Sovellus käsittelee kuvat *javax.microedition.lcdui.Image* -pohjaisina olioina (koodiesimerkki 1).

Koodiesimerkki 1: Kuvan lataus tiedostosta muistiin

```
...
import javax.microedition.lcdui.Image;
...
```

```
public void loadImage(int i, String image) {
    try {
        images[i] = Image.createImage(image);
        ...
    } catch (OutOfMemoryError e) {
        ...
    } catch (IOException e) {
        ...
    }
}
...
```

Käytetyt äänet oli tallennettu MP3- ja WAV-formaatissa. Alunperin äänet olivat MP3-formaatissa mutta äänien toistaminen hidasti sovelluksen ajoa, joka johti kokeiluun kompressoimattoman äänitiedoston kanssa. Äänitiedostoja myös leikattiin lyhyemmiksi, jotta sovelluksen vaatima tallennustila ei olisi liian suuri. Sovellus lataa äänet *javax.microedition.media.Manager*-luokan avulla samaisen paketin Player-rajapintoihin (koodiesimerkki 2). Näin sovelluksen ohjelmakoodissa voidaan kutsua Player-rajapinnan yksinkertaisia metodeja, kun ääniä halutaan toistaa sovelluksessa.

Koodiesimerkki 2: Äänitiedostojen lataus muistiin

```
...
import javax.microedition.media.Manager;
import javax.microedition.media.MediaException;
import javax.microedition.media.Player;
...
public void loadSound(int i, String sound, boolean isWav) {
    if(SOUNDS){
        try{
            InputStream is = getClass().getResourceAsStream("/sounds/"+sound);
            if(!isWav){
                System.out.println(i);
                sounds[i] = Manager.createPlayer(is, "audio/mpeg");
                sounds[i].realize();
            }else {
                sounds[i] = Manager.createPlayer(is, "audio/x-wav");
            }
            sounds[i].prefetch();
        } catch (OutOfMemoryError e) {
            ...
        } catch (MediaException e) {
            ...
        } catch (IOException e) {
```

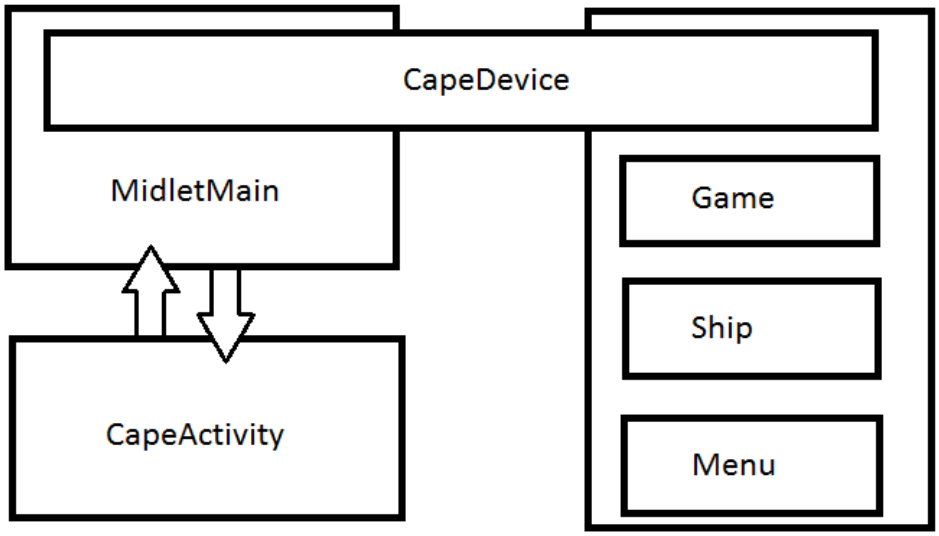
```
                ...
            }
        }
}
...
```

2.3 Arkkitehtuuri

Sovellus on suunniteltu siten, että eri alustoille tarkoitetut sovitukset olisivat mahdollisimman helposti ja nopeasti toteutettavissa. Suunniteltaessa sovellusta pelilogiikka eroteltiin omiin luokkiinsa ja kaikki alustasta riippuvaiset asiat hoidetaan CapeDevice-rajapinnan kautta (kuvio 3). Pelin logiikka ja sen käsittelemät tiedot ovat pilkottu kolmeen luokkaan;

- Game-luokka, joka perii Runnable luokan ja sisältää pelisilmukan ajon,
- Ship-luokka, joka pitää yllä vektorissa itsestään luodut oliot ja sisältää peliobjektien tiedot. Jokainen ship-olio osaa piirtää itsensä,
- Menu-luokka, joka piirtää valintaruudut ja sisältää niiden logiikan.

MidletMain-luokka on Java ME versiossa se, joka piirtää kuvat, soittaa äänet ja käsittelee käyttäjän syötteet. MidletMain on periytetty MIDlet-luokasta ja se toteuttaa CapeDevice-rajapinnan vaatimat metodit. Tämä luokka on ainoa komponentti, joka käyttää alla olevan alustan tarjoamia piirto-, ääni- ja syötteidenlukupalveluja. Pelilogiikan sisältävät luokat käyttävät vain ja ainoastaan *java.util* -paketin tarjoamia luokkia.



Kuvio 3: Sovelluksen arkkitehtuurikuvaus

2.4 CapeDevice -rajapinta

Sovitusten tekoa varten CapeDevice-rajapinta (koodiesimerkki 3) oli kaikista tärkein komponentti sovelluksessa. Pelilogiikan sisältävät luokat käyttävät vain CapeDevice-rajapinnan tarjoamia kutsuja. Eri alustoille sovellusta sovitettaessa tarvitsee pitää huolta siitä, että CapeDevice-rajapinnan toteutus on kunnossa.

CapeDevice-rajapintaa jouduttiin muokkaamaan projektin aikana useita kertoja, jotta kaikki mahdolliset suorat viittaukset Game, Ship ja Menu -luokista MidletMain -luokkaan voitaisiin hoitaa rajapinnan kautta. CapeDevice-rajapinta valmistautui lopulliseen malliinsa vasta myöhäisessä vaiheessa projektia.

Suoria viittauksia MidletMain-luokkaan löytyi useista kohdista ohjelmakoodia. Ensimmäiset versiot CapeDevice-rajapinnasta eivät sisältäneet tarvittuja kutsuja ja MidletMain-luokka sisälsi jopa pelilogiikkaan liittyvää tominnallisuutta. Kaikki tämä korjattiin toimimaan halutun arkkitehtuurin mukaisesti. Korjaukset olivat välttämättömiä, jotta sovitusten teko muille Java-kieltä käyttäville alustoille olisi siistiä ja helppoa.

Koodiesimerkki 3: CapeDevice-rajapinnan kuvaus

```
public interface CapeDevice {
    ...
    public void loadImage(int id, String image);
    public void loadSound(int id, String sound);
```

```
    public void drawImage(int id, int x, int y, double rotation, int
orientation);
    public void rotateImage(int id, double rotation);
    public void drawLine(int x0, int y0, int x1, int y1, boolean dotted);
    public void drawString(int x, int y, String s);
    public String getFile(String file);
    public void setFile(String file, String contents);
    public int getHeight();
    public int getWidth();
    public void flush();
    public void playSound(int id);
    public void close();
    public void stopSound(int id);
    public void initSound(int id);
    public int getImageWidth(int id);
    public int getImageHeight(int id);
}
```

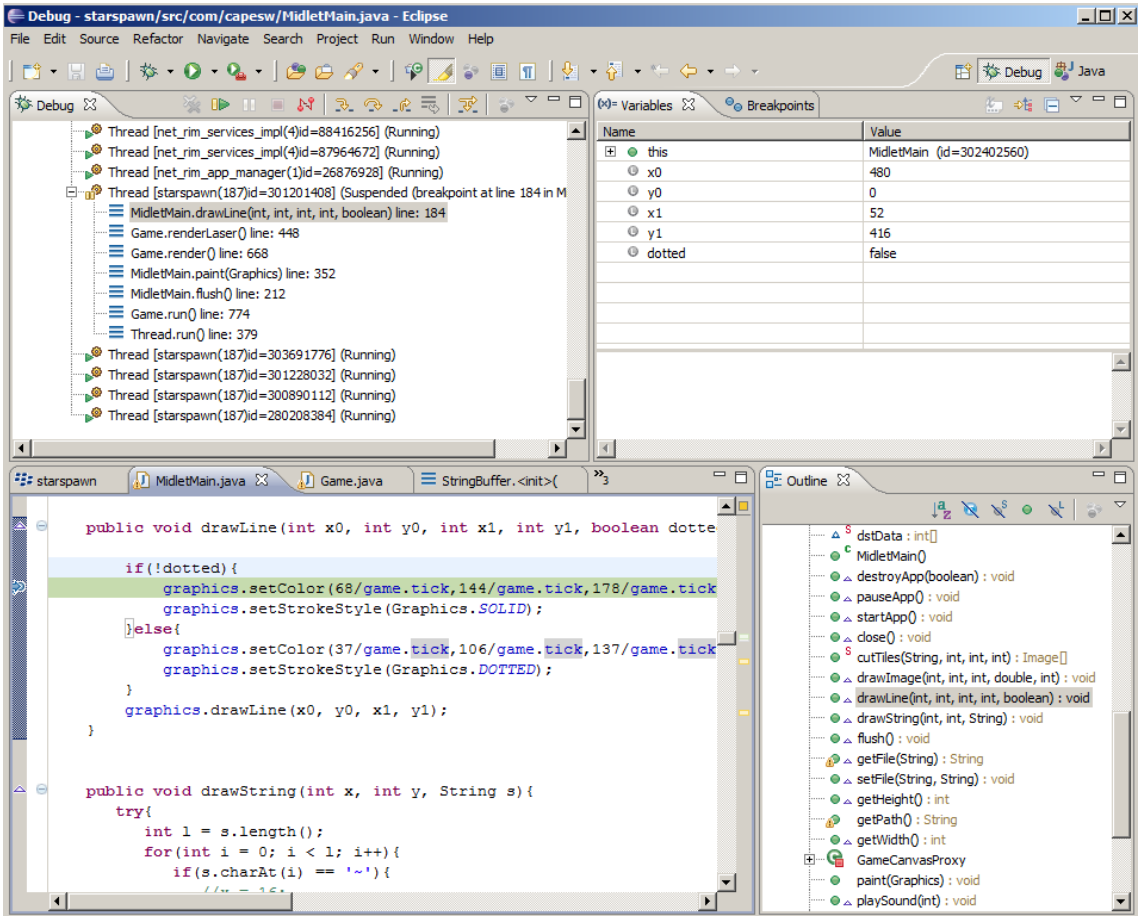
3 BlackBerry sovitus

BlackBerrylle sovittaminen päätettiin pelkästään sillä tiedolla, että BlackBerryn laitteet ovat hyvin Java sidonnaisia. Täyttä selvyyttä ei ollut siitä, toimivatko Java ME standardien mukaiset MIDP 2.0 midlet-sovellukset suoraan laitteissa. Myös se, minkälaiset työkalut BlackBerryn valmistaja RIM (Research in Motion) tarjoaa sovelluskehittäjille, ei ollut täysin selvillä.

Kaikki tarvittavat tiedot saatiin BlackBerryn valmistajan RIMin kehittäjille suunnatuilta web-sivuilta (Developer Zone, 2010). BlackBerry on tehty täysin MIDP 2.0 ja CLDC 1.1 yhteensopivaksi alustaksi. Valmiiden midlettien pitäisi toimia vaivattomasti BlackBerry laitteissa kunhan ne pakataan BlackBerryn omilla työkaluilla. Sovelluksen ohjelmakoodia täytyi kuitenkin muokata yllättävien poikkeustapausten johdosta.

3.1 BlackBerryn kehitystyökalut

BlackBerry alustalle tapahtuvaa sovelluskehitystä voidaan tehdä kahdella eri kehitysympäristöllä. BlackBerryn vanhalla Java Development Environment -kehitystyökalulla, jonka RIM on itse nimennyt olevan jo legacy-ohjelmisto. Uudempi keino on käyttää BlackBerryn Eclipse-liitännäistä. Tämä liitännäinen sisältää kaikki tarvittavat työkalut BlackBerry-sovellusten tekoon (kuvio 4).



Kuvio 4: BlackBerryn Eclipse liitännäisen vianselvitys -olotila

Eclipse liitännäisessä tulee mukana BlackBerryn laitesimulaattorit. Simulaattorissa voidaan ajaa kehiteltävää sovellusta aivan kuin kyseessä olisi aito BlackBerry-laite. Simulaattorissa voidaan simuloida puhelimen eri tilanteita. Mahdollista on mm. asettaa puhelimen suuntautuminen ja verkon tila.

Sovellusta voidaan myös ajaa vianjäljittimen (debugger) kanssa jolloin mahdollisten virheiden etsiminen on huomattavasti nopeampaa. Vaikka BlackBerryn piti toimia MIDP 2.0 profiilin mukaisesti, vianselvitys tuli tarpeelliseksi työkaluksi projektin aikana. BlackBerryn Eclipse-liitännäinen tekee projektin kääntämisen ja paketoinnin automaattisesti. Ainoa asia mikä jätetään kehittäjän varaan on paketin mahdollinen allekirjoittaminen tilanteessa, jossa on käytetty BlackBerryn suojattuja rajapintoja.

3.2 Sovittaminen BlackBerry-alustalle

BlackBerry-alustan sovitusta lähdettiin tekemään BlackBerry Eclipse-liitännäisellä. Eclipsellä aloitettiin uusi BlackBerry Midlet -projekti, johon kopioitiin valmiit

lähdekoodit Java ME projektista. Kuvat ja äänet otettiin suoraan alkuperäisestä Java ME projektista uuteen BlackBerry-projektiin. Koska BlackBerryssä alkuperäinen Java ME ohjelmakoodi toimi lähes sellaisenaan, ei varsinaista sovittamistyötä jouduttu tekemään. Isoin osa BlackBerry-alustan sovituksesta oli vianselvitystä ja vikojen ratkaisujen kehittelyä.

3.3 Vianselvitys ja virheiden korjaus

Vianselvitys BlackBerry-alustalle lähti liikkeelle kokeilemalla suoraan Java ME projektin lähdekoodeja uudessa BlackBerry Eclipse-projektissa. Projekti kääntyi, mutta simulaattorissa ohjelma ei käynnistynyt lainkaan. Tähän syy löytyi BlackBerryn projektitiedostoon määrittelemättä jääneestä midlet kohdasta. Tämä piti määrittää kertomalla luokan koko pakettipolku eli *com.capesw.MidletMain.* Pelkkä luokan nimi ei riittänyt.

Vianselvituksen kanssa kokeillessa sovellus lähti käyntiin mutta mitään ei tapahtunut. Tähän syyksi paljastui midlet-luokan startApp, destroyApp ja pauseApp -metodit. BlackBerryn ja Java ME dokumentaation mukaan nämä metodit olivat suojattuja, mutta käytännössä BlackBerrylle ne täytyi määritellä julkisiksi metodeiksi. Korjauksen jälkeen sovellus käynnistyi ja pelisilmukka rupesi pyörimään.

Pelivalikon jälkeinen pelitila ei toiminut. Syyksi tähän paljastui Ship-luokassa käytetty tyyli muuttumattomien muuttujien alustuksessa. Osaan muuttujista haluttiin tallentaa laitteen ruudun leveys ja korkeus kutsumalla CapeDevice-rajapinnan kautta getHeight- ja getWidth-metodeita. Testatessa paljastui, että pelin käynnistyessä nämä metodit eivät aivan heti palauttaneet oikeita arvoja. Tämä kaatoi sovelluksen siinä tilanteessa, jossa satunnaislukuja tuottava metodiin joutui parametrinä nolla tai negatiivisia lukuja. Ongelma ratkaistiin hakemalla ruudun leveys- ja korkeustiedot vasta, kun niitä oikeasti tarvittiin.

Suurin ongelma tuli vastaan käyttäessä Graphics-luokan drawRGB-metodia. Alkuperäinen Java ME sovellus käytti tätä käännettyjä kuvia piirtäessä. Metodi drawRGB haluaa parametrinä RGB-taulukon sekä joukon koko- ja paikkatietoja.

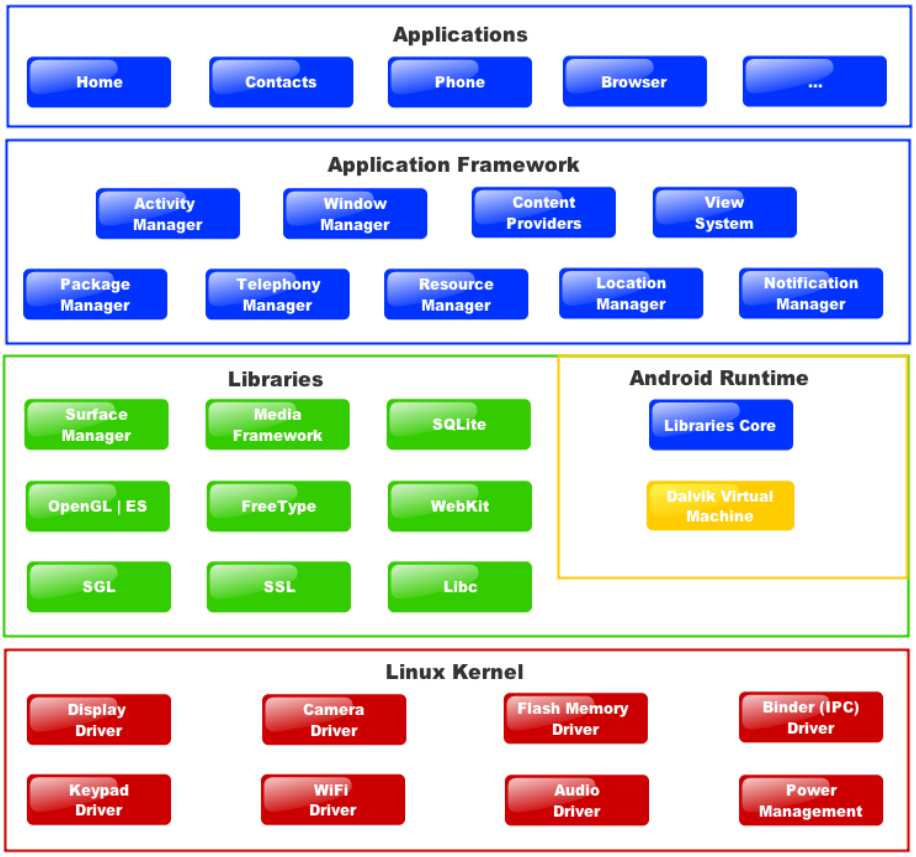
BlackBerry-alustalla kyseinen metodi tuotti tietyissä tilanteissa poikkeustapahtuman toisin kuin normaaleilla Java ME alustoilla, joissa metodi toimi kuten luvattu. Tämän vuoksi objektien piirtoon täytyi luoda BlackBerryllä toimiva ratkaisu. Ongelma korjattiin tekemällä raakamuotoisesta RGB-taulukosta Image-olio ja piirtämällä se BlackBerryllä toimivalla drawImage-metodilla.

4 Android sovitus

Android sovituksen teko lähti nopeasti liikkelle hyvien ohjeiden ja helpon Android SDK (Software Development kit) asennuksen jälkeen. Google pitää yllä täysin päivitettyä Android kehityssivustoa (The Developers Guide, Google). Sivustolta löytyy kaikki ohjeet ja työkalut mitä Android-alustalle kehittäminen vaatii. Sovelluksen Android sovitukseen päädyttiin Android-alustan markkinaosuuden mahdollisen kasvun ja alustan Java-kielen vuoksi.

4.1 Android-alusta

Android on Linux-ytimen päälle rakennettu kokonainen käyttöjärjestelmä. Linux ytimen ja mukaan otettujen kirjastojen (kuvio 5) päälle on rakennettu täysin uusi sovelluskehys, jossa on kaikki tarvittavat hallintaohjelmat mobiililaitteiden resurssien hyödyntämiseen. Kaiken tämän päällä on lopulliset käyttäjän käyttämät sovellukset kuten vaikkapa internet-selain ja kalenteri.



Kuvio 5: Androidin arkkitehtuuri (Kuva: Alvaro Fuentes)

Android-alustan tärkein osa on Dalvik-virtuaalikone, jonka toiminta on täysin erilainen, kuin oikeiden Java-virtuaalikoneiden. InfoQ verkkosivuston Scott Delapin kirjoittaman artikkelin mukaan (Delap, 2010) merkittävimpien erojen joukossa on Dalvikin pieni koko ja sen käyttämä tavukoodi, joka ei ole Javan tavukoodia, vaan Dalvikin omaa tavukoodia. Dalvikia ei ole sovitettu Java ME tai Java SE -alustojen luokkakirjastojen profiileihin. Dalvik käyttää osajoukkoa Apache Harmony -projektin Java-toteutuksesta. Usein käytetyt Java-paketit kuten java.nio, java.lang ja java.util ovat siis kehittäjien käytettävissä.

Jokainen Android-sovellus ajetaan omassa prosessissa, yksityisessä Dalvik-virtuaalikoneen ilmentymässä. Dalvik on varta vasten suunniteltu toimimaan useana samanaikaisesti ajettavana virtuaalikoneena. Alla olevalle Linux-ytimelle jätetään huolehdittavaksi säikeistys ja matalan tason muistinhallinta (Delap, 2010).

4.2 Androidin kehitystyökalut

Google tarjoaa Androidin kehitystyökalut ilmaiseksi. Android SDK paketissa tulee mukana mm. emulaattori, android-komentorivityökalu, useita erilaisia työkaluja liittyen projektien kääntämiseen ja paketointiin, sekä emulaattorin hallintasovellus.

Useimmat työkaluista ovat komentoriviltä ajettavia. Tämä mahdollistaa batch-scriptien käytön, jos kääntäminen tai paketointi halutaan automatisoida. Jos kehityksen haluaa tehdä pelkästään näillä työkaluilla, Apache Ant-ohjelma täytyy asentaa, jotta projektin kääntäminen onnistuu.

Google tarjoaa myös Android Development Tools (ADT) -liitännäisen Eclipseen. Tämä mahdollistaa koko kehitystyön tapahtumisen Eclipsellä. Liitännäinen tarvitsee toimiakseen edellä mainitut Androidin kehitystyökalut. Asennuksen jälkeen koko kehitysprosessin voi tehdä Eclipsen kautta. Tässä tapauksessa Apache Ant-ohjelmaa ei tarvitse asentaa koska Eclipse sisältää sen.

Kehittäjien on hyvä tutustua alla oleviin Android SDK -työkaluihin, jolloin Android-projektin tiedostot ja tiedostorakenne tulisi ymmärrettyä paremmin. Emulaattoreita voidaan ajaa ja luoda Eclipsen kautta ja Android kehitystyökalujen vianselvittäjä toimii

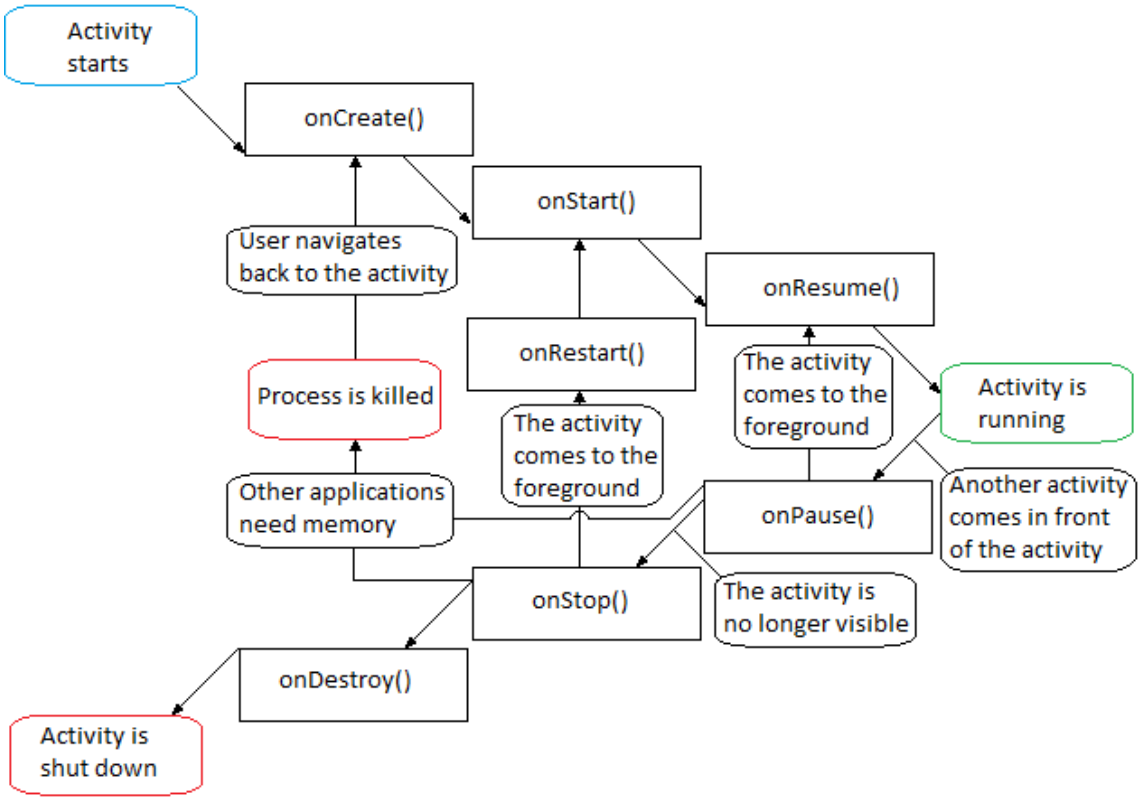
suoraan Eclipsen normaalin vianselvittäjän sijasta Android projektien ja emulaattorin kanssa.

4.3 Sovelluksen elinkaari ja resurssit

Android sovelluksilla ei ole perinteistä ohjelman aloitusosoitetta, vaan ne sisältävät tarvittavia komponentteja, joita järjestelmä kykenee ajamaan tarpeen mukaan (The Developer's Guide, Google). Androidissa on olemassa neljä erilaista komponenttityyppiä, joista tässä työssä keskitytään aktiviteetti-komponenttiin.

Aktiviteetti näyttää visuaalisen käyttöliittymän jollekin yksittäiselle käyttäjän käyttämälle sovellukselle, tämä voi olla vain yksinkertainen osa isompaa ohjelmaa tai vaikkapa täysikokoinen peliruutu.

Aktiviteetin elinkaari on erilainen kuin normaalisti sovelluksissa. Tämä johtuu Androidin komponenttiajattelusta ja tavasta millä ohjelmia ajetaan laitteessa. Tarkoituksena on pitää automaattisena muistinhallinta, sovellusten käynnistys ja sammutus piilottaen kaiken käyttäjältä. Aktiviteetille on määritelty tarkasti sen elinkaari ja missä vaiheessa mitäkin tapahtuu (kuvio 6) (The Developer's Guide, Google).



Kuvio 6: Aktiviteetin elinkaari

Android sovitusta tehtäessä erilainen elinkaari tuli ongelmalliseksi, koska Java ME lähdekoodeja ei oltu luotu näiden tietojen perusteella. Pelilogiikassa ei oltu varauduttu jokaiseen tilanteeseen mihin aktiviteetti saattaa joutua. Myös se miten Android käsittelee sovelluksia ja niiden sammuttamista toi esiin virheitä testauksen aikana. Vaikka ohjelma sammutettiin kutsumalla finish-metodia, uudelleenkäynnistyksen jälkeen sovelluksen tila oli sekaisin eikä se rakentunut kokonaan uusiksi.

Käytössä olevista kuva- ja äänitiedostoista luodaan Androidin käännösvaiheessa R-luokka joka sisältää listan käytössä olevien resurssien tunnistenumeroista. Tästä luokasta luodaan ajon aikana resurssi-olio, jolta pyydetään halutut resurssit. Tavallisesti kehittäjä kykenee käyttämään koodissa R-luokkaan kehitettyjä enumeraatioita, joilla viitataan tietyn resurssin tunnistenumeroon.

Koska pelilogiikan sisältämät lähdekoodit sisälsivät listan käytettävistä kuva- ja äänitiedostoista, resursseja ei voitu hakea muistiin resurssi-oliosta. Aluksi täytyi selvittää tiedoston nimen perusteella resurssin tunnistenumero, jonka perusteella resurssin saa käyttöönsä (koodiesimerkki 4).

Kuvatiedostojen täytyy sijaita Android-projektissa */res/drawable* ja äänitiedostojen */res/raw* -kansiossa. Tiedostojen nimissä ei saa esiintyä kuin numeroita ja pieniä kirjaimia väliltä *a-z*, alaviiva-merkki on myös sallittu. Muuten R-luokkaa ei kyetä luomaan ja projektin kääntäminen epäonnistuu.

Koodiesimerkki 4: Resurssien käyttöönotto Androidissa

```
// Googlen oma esimerkki resurssien hakemisesta
// käyttää suoraan R-luokan tarjoamia enumeraatioita
ImageView imageView = (ImageView) findViewById(R.id.myimageview);
imageView.setImageResource(R.drawable.myimage);

// pelissä käytetty tapa saada resurssi
// filename ei saa sisältää tiedostopäätettä, android tunnistaa tämän itse
images[i] = resources.getDrawable(resources.getIdentifier(filename,
                                  "drawable", "com.capesw"));
```

4.4 Grafiikan piirto

Toteutettu aktiviteetti-luokka pitää sisällään View-luokan, jonka onDraw-metodi ylikirjoitettiin omaan käyttöön sopivaksi (koodiesimerkki 5). Sovelluksen aktiviteetti asettaa rakentajassa mainitusta view-luokasta luodun olion setContentView-metodilla päänäkymäksi ja asettaa näkymän koon koko ruudun kokoiseksi. Näkymälle asetetaan myös kosketustapahtumia lukeva kuuntelija, jolla saadaan napattua kaikki näytöllä tapahtuvat kosketustapahtumat, jotta käyttäjän syötteeseen voidaan reagoida.

Koodiesimerkki 5: Ylikirjoitettu View-luokka muokattuna omiin tarpeisiin

```
private class CapeView extends View {
    public CapeView(Context context) {
        super(context);
    }

    public void onDraw(Canvas canvas) {
        currentCanvas = canvas;
        if(game!=null) game.render();
        System.gc();
    }
    protected void onSizeChanged(int w, int h, int oldw, int oldh){
        super.onSizeChanged(w, h, oldw, oldh);
        if((w>0 && h>0 ) && game==null){
            game = new Game(activity);
            game.start();
        }
    }
}
```

Näkymän onDraw-metodissa käydään jokainen kerta, kun pelilogiikka haluaa ruudun päivittyvän. Tämä ei tapahdu suoraan kutsumalla onDraw-metodia, vaan muualla koodissa kutsutaan näkymän postInvalidate-metodia. Kyseistä metodia joudutaan käyttämään normaalin Invalidate-metodin sijaan säikeistyksen vuoksi.

Tämän jälkeen näkymä suorittaa onDraw-kutsun, jossa ruudun piirto tapahtuisi normaalisti. Koska sovellusta ei oltu luotu alunperin Androidille, talteen otetaan onDraw-metodille parametrinä annetun canvas olio ja kutsutaan peli-olion render-metodia. Tässä Game-luokan render-metodissa taas kutsutaan aktiviteetin piirtokutsuja CapeDevice-rajapinnan kautta. Kaikki piirto tapahtuu aina siihen canvas-olioon, joka on viimeiseksi tallennettu onDraw-metodissa currentCanvas-muuttujaan.

CapeDevice-rajapinnassa on määritelty kolme erilaista piirtofunktiota, drawLine drawString ja drawImage -metodit. Sovelluksessa käytetyt kuvat pidetään muistissa Drawable-tyyppisinä olioina. Poikkeuksena fontteja ylläpitävä taulukko, joka on tyyppiä BitmapDrawable.

Kuvien piirto (koodiesimerkki 6) tapahtuu asettamalla Drawable-tyyppiselle kuvalle sen reunalueet, tämä eroaa Java ME tyylisestä tavasta, jossa annetaan piirtokäskylle pelkät koordinaatit ja ankkuri. Jos kuva halutaan piirtää vinoon tietyn kulman mukaisesti, Androidissa on mahdollista pyörittää Canvas-luokkaa tietystä koordinaatista haluttuun kulmaan. Piirron jälkeen canvas palautetaan alkuperäiseen asentoon.

Koodiesimerkki 6: drawImage-metodissa käytetty piirtotyyli

```
// Esimerkki miten drawImage-metodi piirtää käännetyn kuvan canvasta
// kääntämällä. Kuva piirretään myös annettujen koordinaattien keskelle.
    currentCanvas.save();
    currentCanvas.rotate((float) rotation, x, y);
    images[id].setBounds(x - (imgW / 2), y - (imgH / 2), x
                + (imgW / 2), y + (imgH / 2));
    images[id].draw(currentCanvas);
    currentCanvas.restore();
```

Viivojen piirto tapahtuu CapeDevice-rajapinnan tarjoamalla drawLine-metodilla. Aktiviteetissä viivan piirrossa käytetään suoraan Androidin Canvas-luokan drawLine-metodia.

Tekstin näytölle piirtämisessä käytetään sovelluksen ajon alussa pilkottua fonttikuva-taulukkoa. Kirjoitusfunktiolle annetaan kirjoitettava merkkijono ja koordinaatit mistä piirto aloitetaan (koodiesimerkki 7). Koodi käy yksitellen läpi annetun tekstipätkän kirjain kirjaimelta piirtäen yksitellen kirjaimet fonttikuva-taulukosta.

Koodiesimerkki 7: drawString-metodi kokonaisuudessaan

```
public void drawString(int x, int y, String s) {
    try{
        int l = s.length();
        for (int i = 0; i < l; i++) {
            if (s.charAt(i) == '~') {
                y += 24;
            } else {
                font[ABC.indexOf(s.charAt(i))].setBounds(x,y,
                    x+ font[ABC.indexOf(s.charAt(i))].getIntrinsicWidth(),
                    y+font[ABC.indexOf(s.charAt(i))].getIntrinsicHeight());
```

```
                font[ABC.indexOf(s.charAt(i))].draw(currentCanvas);
                x += font[ABC.indexOf(s.charAt(i))].getIntrinsicWidth();
            }
        }
    } catch(ArrayIndexOutOfBoundsException e){
        e.printStackTrace();
    }
}
```

4.5 Äänien toisto

Äänten soittoon käytetään Androidin tarjoamaa MediaPlayer-luokkaa. Koska sovelluksessa käytetään vain muutamia eri äänitiedostoja, pystytään pitämään yllä pientä MediaPlayer-olioita sisältävää taulukkoa (koodiesimerkki 8). CapeDevice-rajapinta tarjoaa pelilogiikan käytettäväksi initSound, playSound ja stopSound -funktiot.

Äänet alustava funktio initSound ei Androidissa ole tarpeellinen, koska MediaPlayeri on prepared tilassa heti äänitiedoston lataamisen jälkeen. Äänet voidaan soittaa ja pysäyttää playSound ja stopSound -funktioilla. Teknisesti ääniä ei ikinä pysäytetä kokonaan, jotta ne eivät menisi stopped tilaan, tällöin MediaPlayer pitäisi asettaa uusiksi prepared tilaan. Ääni tauotetaan ja soittokohta asetetaan tiedoston alkuun ilman, että käytettäisiin stop-funktiota.

Koodiesimerkki 8: Äänen lataus MediaPlayer-tyyppiseen taulukkoon

```
public void loadSound(int i, String sound, boolean isWav) {
    if (SOUNDS) {
        try {
            // saamme stringin jossa on tiedostopääte, se täytyy poistaa
            String tmp = sound.substring(0, sound.length() - 4);
            int resid = resources.getIdentifier(tmp, "raw", "com.capesw");
            if(resid!=0){
                sounds[i] = MediaPlayer.create(this, resid);
            }
            if (sounds[i] == null) {
                System.out.print("failed to load: " + sound);
            }
        } catch (IndexOutOfBoundsException e) {
            e.printStackTrace();
        }
    }
}
```

4.6 Vianselvitys ja vikojen korjaus

Vianselvitys aloitettiin varhain kehitystyön lomassa. Useimmat CapeDevice-rajapinnan vaatimista metodeista jäi alkuun toteuttamatta, jotta suunniteltujen konseptien toiminta voitiin tarkistaa varhaisessa vaiheessa. Vianselvitys toteutettiin suurimmaksi osaksi emulaattorissa, koska oikeaa Android-laitetta ei vielä ollut työn alkuvaiheessa saatavilla.

Suurimpia ongelmia Android sovituksessa oli se, että näkymä ei ollut sovelluksen aloitusvaiheessa saanut oikeita korkeus- ja leveys-arvoja. Tämä aiheutti ongelman kun Game-luokassa luodaan useita Ship-olioita. Ship-luokassa käytettiin CapeDevice-rajapinnan kautta getHeight ja getWidth -funktioita asettamaan peliobjektien koordinaatteja.

Näiden funktioiden piti palauttaa koko ruudun leveys- ja korkeusarvot, mutta sovelluksen käynnistyksen jälkeen näkymä palautti hetken ajan näiden kautta vääriä arvoja. Osasyynä ongelmaan oli se, että alkuperäistä sovellusta ei oltu suunniteltu täysin Androidin aktiviteetin elinkaaren mukaisesti.

Asia korjattiin (koodiesimerkki 5) ylikirjoittamalla näkymän onSizeChanged-metodi. Tätä metodia kutsutaan joka kerta, kun näkymän koko muuttuu. Game-olion luonti siirrettiin kyseiseen metodiin. Metodi tarkastaa onko näkymän leveys ja korkeus nollaa suurempi ja luo Game-olion, jos oliota ei ole vielä luotu. Näin varmistetaan se, että peliobjektit ovat oikeissa alkukoordinaateissa.

Tässä työssä on aikaisemmin mainittu sovelluksen elinkaaren erilaisuuden tuomista ongelmista. Ongelmana oli tilanne, jossa käyttäjä lähtee sovelluksesta pois joko back tai home -näppäintä painamalla, jolloin sovellus menee oletetusti taustalle ja taukotilaan. Koska pelilogiikassa ei oltu varauduttu tarpeeksi mahdollisten taukotilojen varalta. Pelitilanteen saaminen oikeaan tilaan aktiviteetistä oli hankalaa. Myös äänet jäivät soimaan tässä tilanteessa.

Helpoin tapa ratkaista asia oli saada sovellus käynnistymään uudestaan tilanteessa, jossa käyttäjä on painanut back tai home -näppäintä ja tullut takaisin sovellukseen. Tämäkään ei ollut helppoa, koska normaali finish-metodin kutsu ei onnistunut tappamaan muita kuin Aktiviteettia ajavan säikeen. Käyttäjän tullessa takaisin peliin pelitila oli mennyt

sekaisin eikä uutta peliä oltu onnistuttu luomaan vanhojen säikeiden ollessa vielä taustalla.

Lopullinen ratkaisu ongelmaan oli etsiä aktiviteetin onDestroy-metodissa sovelluksen prosessin tunnistenumero (PID) ja antaa Androidin killProcess -järjestelmäkäsky sovelluksen tunnistenumerolle. Tämä varmisti sovellukselle oikeanlaisen käyttäytymisen tilanteissa, jossa back tai home -näppäintä painettiin. Aktiviteetin onDestroy-metodiin luotiin myös kutsut ääntä säilyttävien MediaPlayer-olioiden release-funktioihin, jolla äänten soiminen saatiin pysäytettyä.

4.7 Sovelluksen allekirjoittaminen

Android sovelluksen täytyy olla allekirjoitettu. Android-järjestelmä ei suostu asentamaan sovelluksia, joita ei ole allekirjoitettu sertifikaatilla. Sertifikaatin tavoitteena on tunnistaa sovelluksen kehittäjä ja käyttää sertifikaatteja sovelluksesta toiseensa menevien yhteksien luottamuksen varmentimena. Sovelluksen allekirjoitus ei vaadi keskitettyä sertifikaattiauktoriteettia joten itse tehdyt sertifikaatit ovat toimivia (The Developer's Guide, Google).

Avaimet kyetään luomaan Javan JDK-paketin mukana tulevalla keytool-ohjelmalla. Keytool-ohjelmistolla luodaan oma avainvarasto-tiedosto (keystore). Tätä avainvarastoa hyväksi käyttäen allekirjoitetaan jarsigner-ohjelmalla kääntämisen ja paketoinnin seurauksena saatu allekirjoittamaton apk-tiedosto.

Kun paketti on allekirjoitettu. Androidin ohjeet suosittelevat paketin kohdistusta zipalign-ohjelmalla. Ohjelmalla varmistetaan paketoimattoman tiedon olevan kohdistettuja neljän tavun rajoihin. Tällöin Android-järjestelmä pystyy tekemään optimointeja paketin sisältämien tiedostojen lukemisessa.

Allekirjoitettu ja oikein paketoitu apk-paketti on valmis asennettavaksi Android laitteisiin.

4.8 Android sovituksen automaattinen kääntö ja paketointi

Työn tilaaja halusi saada vaivattomasti Java ME sovelluksen lähdekoodeista ja resurssitiedostoista valmiin Android sovelluksen niin sanotusti nappia painamalla. Tätä varten luotiin MS-DOS komentojono joka

- luo tarvittavan kansiorakenteen Android-projektia varten,
- luo android-komentorivityökalua käyttäen Android projektin Android-sovituksen koodien päälle,
- kopioi Java ME projektista tarvittavat lähdekooditiedostot ja resurssit Android projektin oikeisiin kansioihin,
- nimeää uudelleen resurssien nimet oikeanlaisiksi Android projektia varten,
- kääntää Apache Ant -ohjelmalla projektista release -version,
- allekirjoittaa käännöksen luoman paketti-tiedoston sisällön jarsigner -ohjelmalla ja kohdistaa tavut zipalign -ohjelmalla.

Lopputuloksena sovelluksesta on luotu uuden Android projektin /bin kansioon apk-muotoinen paketti, joka on valmis asennettavaksi laitteisiin. Skriptin ansiosta Android sovituksesta ei tarvitse ylläpitää kuin Android projektin käyttämä lähdekoodi- ja asetus-tiedosto sekä XML-tiedosto, jossa on määritelty joidenkin muuttujien arvoja.

5 Yhteenveto

Tässä työssä käsiteltiin Java-pohjaisen mobiilisovelluksen sovittamista Android ja BlackBerry mobiilialustoille. Työssä esiteltiin Java ME alustalle alunperin suunniteltu mobiilisovellus. Java ME Sovelluksesta esiteltiin idea, resurssit, arkkitehtuuri ja sovituksia varten luotu rajapinta. Vain oleellisimmat asiat sovitusten tekoon liittyen selitettiin.

BlackBerry sovitusta käsiteltiin huomattavasti vähemmän kuin Android sovitusta. Tämä johtui lähinnä siitä, että BlackBerry alustalla alkuperäinen Java ME sovellus toimi lähes sellaisenaan. Sovitustyössä tutkittiin vain mahdolliset viat ja kehitettiin näihin ratkaisut niin, että lähdekoodin muutokset kyettiin viemään alkuperäiseen Java ME projektiin.

Eniten työtä vaati Android sovitus. Koska Android-järjestelmä on tehty täysin uudenlaisella tavalla ja se kattaa mitä eriskummallisempien sovellusten kehityksen. Täytyi yksinkertaisten asioiden tekoa varten tutkia laajoja oppaita, siitä miten Android-järjestelmässä on satuttu tekemään asiat. Täytyykin kysyä, miten tarpeellinen Java ME standardien kirjastoista irrottautuminen omaan Dalvik-virtuaalikoneeseen ja Android-kirjastoihin on? Tämä ratkaistaneen tulevaisuudessa älypuhelinten markkinaosuuksilla.

Android sovituksessa luotiin uusi alustakohtainen ohjelmakoodi, joka toteutti CapeDevice-rajapinnan. Projektin aikana Android sovitus jouduttiin kirjoittamaan lähestulkoon uudestaan, koska alkuperäisen Java ME sovelluksen alustariippuvainen osa ja CapeDevice-rajapinta muuttui toiminnaltaan lähes kokonaan ensimmäisen Android sovituksen valmistumisen jälkeen.

BlackBerry ja Android sovitusten teossa käytettiin Eclipse-kehitysympäristöä. Käytössä oli myös tortoiseSVN -ohjelmisto versionhallinta-järjestelmän käsittelyyn. Lähdekoodien käsittelyyn ja vertailuun käytettiin välillä myös notepad++ -tekstieditoria Eclipsen oman editorin rinnalla.

Android sovitus kehitettiin ja testattiin siihen kuntoon, että sovellus toimii oikein oikeassa Android laitteessa. Android sovitus ei kuitenkaan ole kirjoitushetkellä vielä täysin julkaisukelpoisessa kunnossa. Konfiguraatiotiedosto täytyy käydä kunnollisesti

läpi ja oikeankokoiset taustakuvat Android-laitteille ovat vielä tekeillä. Myös uusimpien Android-laitteiden tehokkuuden vuoksi pelilogiikka saattaa joutua korjailtavaksi.

BlackBerry sovituksessa korjatut ongelmat täytyy saattaa alkuperäiseen Java ME projektiin. Ohjelma toimi lähes moitteetta simulaattorissa. Koska varsinaisia BlackBerry laitteita on hankala saada Suomesta, ei sovellusta ole voitu testata aidolla laitteella. Täten toimintavarmuutta ei ole taattavissa. Suunnitelmissa on kuitenkin BlackBerry version julkaisu jossain vaiheessa tulevaisuutta.

Projektin aikana opin työskentelemään toisen henkilön luoman koodin parissa. Aluksi tuntui hankalalta ymmärtää outoa ja uutta ohjelmakoodia. Jotta valmista ohjelmakoodia kykeni ymmärtämään nopeammin ja paremmin, järjestelmällisyys oli tärkeää. Asioiden oikeanlainen jäsentely ja tietynlainen avoin asennoituminen helpotti työn sisäistämistä.

Lähteet

Java ME Technology, Oracle [online] [viitattu 3.9.2010] Saatavissa: http://www.oracle.com/technetwork/java/javame/tech/index.html

Delap, Scott 12.11.2007 Google's Android SDK Bypasses Java ME in Favor of Java Lite and Apache Harmony. InfoQ [online] [viitattu 3.9.2010] Saatavissa: http://www.infoq.com/news/2007/11/android-java

The Developer's Guide, Google [online] [viitattu 3.9.2010] Saatavissa: http://developer.android.com/guide/index.html

Developer Zone, BlackBerry [online] [viitattu 3.9.2010] Saatavissa: http://na.blackberry.com/eng/developers/