ANDROID- JA BLUETOOTH-KEHITYS KÄYTÄNNÖN ESIMERKILLÄ

Teemu Gustafsson

Opinnäytetyö
Maaliskuu 2012
Tietotekniikan koulutusohjelma
Ohjelmistotekniikka
Tampereen Ammattikorkeakoulu

TIIVISTELMÄ

Tampereen Ammattikorkeakoulu
Tietotekniikan koulutusohjelma
Ohjelmistotekniikan suuntautumisvaihtoehto

GUSTAFSSON TEEMU:
Android- ja Bluetooth-kehitys käytännön esimerkillä

Opinnäytetyo 140s, josta liitteitä 70s.
Maaliskuu 2012

Tämä opinnäytetyö käsittelee Android-pohjaisille mobiililaitteille suunnattua ohjelmistokehitystä pienen esimerkkiohjelman avulla tarkasteltuna. Samalla tutustutaan myös Bluetooth-kehityksen perusteisiin sen ollessa merkittävä osa käytettävää esimerkkiohjelmaa.

Esimerkkiohjelma on kahden pelaajan kahdella Android-laitteella pelattava arcade-peli, jossa ensimmäinen pelaaja ohjastaa ufoa, jota toinen pelaaja yrittää ampua alas. Laitteiden välinen yhteys muodostetaan Bluetoothia käyttäen.

Esimerkkiohjelma on suunniteltu siten, että se käyttäisi mahdollisimman monipuolisesti hyväkseen Android-sovellusten yleisimpiä komponentteja ja ominaisuuksia. Näitä ovat mm. aktiviteetit, lähetyksien vastaanottajat ja säikeet sekä niiden oikeaoppinen käyttö, kosketusnäytön käyttö, äänentoisto, valikkojen luonti ja käyttö, 2D-grafiikan piirto ja päivitys sekä Bluetooth-yhteyden muodostus ja ylläpito.

Opinnäytetyö on jaettu kolmeen osaan. Ensimmäisessä osassa tutustutaan Anroid-järjestelmään käsittelemällä ensin lyhyesti Androidin historiaa ja esittelemällä tämän jälkeen järjestelmän yleinen arkkitehtuuri ja tyypillisen Android-sovelluksen olennaisimmat komponentit. Toisessa osassa tutustutaan Bluetoothin yleisiin, Androidista riippumattomiin perusteisiin.

Kolmannessa ja keskeisimmässä osassa käydään läpi esimerkkiohjelman komponentit ja niiden käytännön toteutus koodin tasolta katsottuna. Ohjelman koodi kokonaisuudessaan löytyy liitteistä.

Opinnäytetyön tavoitteena on antaa sen lukijalle kattavat perustiedot Android- ja Bluetooth-kehityksestä sekä teorian että käytännön tasoilla.

Avainsanat; Android, Bluetooth, mobiililaitteet, peli, arcade

ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
Degree Programme in Information Technology
Option of Software Engineering

GUSTAFSSON TEEMU: Android- and Bluetooth-development with practical example
Bachelor's thesis 70 pages, appendices 70 pages.
March 2012

This bachelor's thesis is aimed to serve as an introductionary look to Android- and Bluetooth-based software development using a simple 2D-arcade game as an example program to illustrate the different aspects of development for the platform and standard in question.

The game is a two player arcade-game played between two Android-powered mobile devices with the communication between them established through Bluetooth. In the game the first player plays as an UFO that's hovering over a sky which the other player is trying to shoot down by hitting it with his finger on the other device.

The different features of Android under examination are: building applications with multiple activities and threads, 2D-graphics rendering, media playback, sending and receiving in-application messages using Handlers and BroadCastReceivers, touchscreen usage, implementing proper Android-style menus and establishing Bluetooth-communication between two devices.

It is hoped that after reading this document the reader would have the sufficient basic skills to start developing his or hers own Android-applications.

Keywords: Android, Bluetooth, mobiledevices, game, arcade

SISÄLTÖ

TERMIT JA LYHENTEET

API	Application programming interface eli ohjelmointirajapinta on määritelmä jonka avulla eri ohjelmat voivat vaihtaa tai muuten käyttää toisten komponenttien tietoja ja ominaisuuksia hyväkseen.
GUI	Graphical User Interface tarkoittaa käyttöliittymää jonka tarkoituksena on antaa käyttäjälle mahdollisuus kommunikoida GUI:ta käyttävien järjestelmien ja ohjelmistojen kanssa graafisesti tekstinsyötön sijaan.
RAM	Random Access Memory eli keskusmuisti tai käyttömuisti on työmuisti johon latautuvat käyttöjärjestelmän ohjelmat, sovellukset ja näiden tarvitsemat tiedot. RAM-muistiin ladatut ohjelmat ja sovellukset toimivat nopeammin kuin massamuisteilla olevat.
UI	User Interface eli käyttöliittymä on se osa laitetta, ohjelmaa tai järjestelmää jonka kautta tuotetta käytetään. UI voi olla esimerkiksi grafiikkaan pohjautuva eli GUI tai tekstipohjainen CLI(Command-Line Interface).

UIID	UUID on 128-bittinen satunnais merkkijono, jonka tarkoitus on mahdollistaa nimensä mukaisesti uniikkien tunnusten luonti mahdollisimman monille eri kohteille jotta ne voitaisiin helpommin erottaa toisistaan. Täysin ainutkertaisten tunnustusten luonti ei tietenkään ole mahdollista mutta UUID 128-bittisen koon vuoksi vaara kahdesta tai useammasta samasta tunnuksesta on hyvin pieni. UUID:n voi luoda joko itse tai käyttää esimerkiksi netistä löytyviä monia UUID-tunnuksen luomista varten tarkoitettuja satunnaisgeneraattoreita.
URI	Uniform Resource Identifier on merkkijono joka kertoo mistä jokin tietty tieto löytyy esimerkiksi tietokoneen kovalevyltä.
XML	Extensive Markup Language on merkintäkieli joka mahdollistaa laajojen tietomassojen selkeämmän jäsentämisen. XML:ää käyttämällä tallennettavan tiedon merkitys voidaan kuvata itse tiedon sekaan. XML:ää käytetään formaattina tiedonvälityksessä eri järjestelmien välillä sekä dokumenttien tallentamisessa.

1 JOHDANTO

Tämän opinnäytetyön tarkoituksena on tutustuttaa lukija Android-ja Bluetooth-ohjelmistokehityksen perusperiaatteisiin sekä teorian että käytännön tasoilla. Käytännön tasoa valaistaan pienen esimerkki ohjelman avulla jonka tarkoituksena on käyttää mahdollisimman kattavasti hyväkseen teoriaosuudessa läpikäytyjä Androidin ja Bluetoothin perusominaisuuksia.

Esimerkki ohjelmana toimii pieni kahdella Android-laitteella pelattava kahden pelaajan arcade-peli jossa toinen pelaaja ohjastaa taivaalla leijuvaa ufoa jota toinen pelaaja yrittää ampua alas ennen aikarajan umpeutumista. Peliä pelataan kosketusnäytön avulla niin että ufoa liikutetaan laskemalla sormi ufo-kuvakkeen päälle ja liuttamalla sen jälkeen sormea näytön pinnalla ufon seuratessa samaa rataa perästä. Tykillä taasen ammutaan koskettamalla haluttua kohtaa ruudulla jossa tapahtuu tämän jälkeen räjähdystapahtuma.

Tarkasteltavia ominaisuuksia ovat aktiviteettien ja säikeiden käyttö sekä niiden välillä kommunikointi, valikkojen toteutus ja käyttö, kosketusnäytön käyttö, median toisto ja 2D-grafiikan piirto sekä Bluetooth-yhteyden muodostus ja ylläpito.

Opinnäytetyö on jaettu seuraaviin 7 kappaleeseen.

Kappaleessa 2 käsitellään lyhyesti Androidin- ja Android-laitteiden historiaa ja arkkitehtuuria. Kappaleessa 3 esitellään tyypillisen Android-applikaation keskeisimmät komponentit. Kappaleessa 4 käydään Bluetooth-yhteyden muodostamisen ja ylläpidon yleiset periaatteet läpi niiltä osin mitkä eivät koske Android-ympäristön omia, Bluetoothin käyttöön liittyviä vaatimuksia.

Kappaleessa 5 käydään ensin nopeasti läpi esimerkki ohjelman yleinen toiminnallisuus ja esitellään sen jälkeen applikaation varsinaiset komponentit ja niiden käytännön toteutus koodin tasolta katsottuna. Samalla tutustutaan muutamiin muihinkin Androidin

piirteisiin joita ei kappaleessa 3 käyty läpi kuten 2D-grafiikan piirtoon, kosketusnäytön ja valikkojen käyttöön ja toteutukseen sekä median toistoon.

Kappaleessa 6 käydään läpi mahdolliset parannukset mitä ohjelman tekoaikana on tullut mieleen mutta joita ei ole ehditty tai pystytty lisäämään applikaatioon sekä selitetään miksi vaatimusmäärittelyssä luvatut mutta ohjelmasta puuttuvat ominaisuudet jätettiin tekemättä.

Kappale 7 sisältää loppuyhteenvedon jossa kerrataan opinnäytetyön aikana tehdyt huomiot ja opetukset Android- ja Bluetooth-kehitykseen liittyen.

2 YLEISTÄ ANDROIDISTA

2.1 Androidin taustaa

Android on mobiililaitteille suunnattu, avoimeen lähdekoodiin perustuva ja Linux-ytimen päälle rakennettu ohjelmistoympäristö, joka koostuu käyttöjärjestelmästä, väliohjelmistosta sekä joukosta keskeisimpiä avain-applikaatioita. Alunperin Android kuului Android inc. nimiselle yhtiölle mutta Google osti yrityksen vuonna 2005 ja siitä saakka kehitys on tapahtunut hakukonejätin johdolla.

Google ei kuitenkaan itse valmista Android-pohjaisia puhelimia vaan yritys lisensoi Androidin sitä haluaville valmistajille jotka saavat yksilöidä puhelimensa kuten haluavat kunhan he pitävät huolen että lopputuote on kuitenkin yhteensopiva muiden valmistajien Android-puhelimien kanssa ja että puhelimet voivat suorittaa myös kolmansien osapuolien kehittämiä applikaatioita.

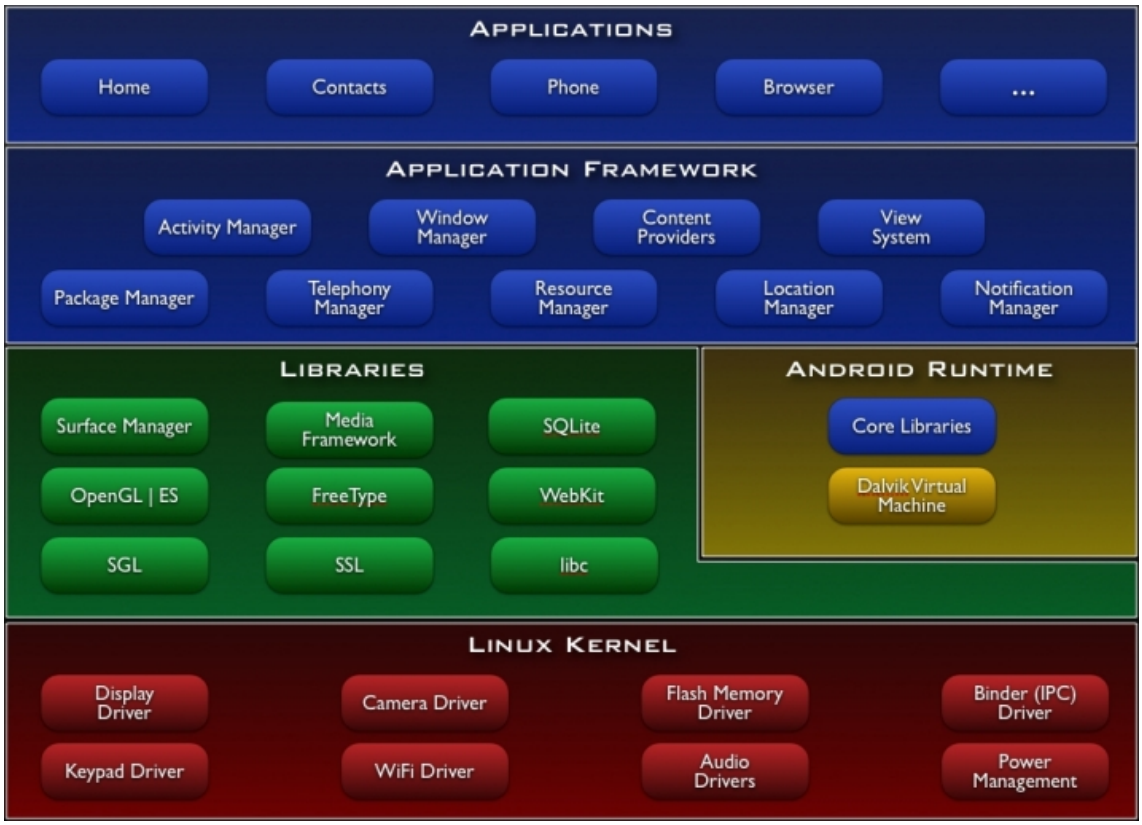
Vapaa muokattavuus onkin yksi Androidin suurimmista houkutuksista eri puhelinvalmistajille ja ohjelmistokehittäjille. Houkuttelevuutta lisää myös se että Android-applikaatioiden ohjelmointikieli perustuu suosittuun Javaan siitä huolimatta että sen peruskomponentit ovat kirjoitetut C- ja C++-kielillä.

Suurimpia Android-puhelin valmistajia ovat tällä hetkellä Samsung, Motorola ja HTC.

2.2 Arkkitehtuuri

Tämä ala-kappale perustuu Android kehittäjien kotisivuilta what is android?-osiosta löytyviin tietoihin(http://developer.android.com/guide/basics/what-is-android.html).

Androidin arkkitehtuuri koostuu kuvan 1 esittämistä komponenteista.



Kuva 1: Android-järjestelmän arkkitehtuuri

Applikaatiokerros(Applications) sisältää kaikki Android-järjestelmän mukana tulevat tai muuten tuetut ohjelmat ja ominaisuudet. Kerroksesta löytyviä applikaatioita ovat mm. sähköposti, SMS-ohjelmisto, kalenteri, karttapalvelut ja nettiselain.

Koska Android on avoin alusta kaikkia perus-applikaatioita voidaan muokata, lisätä tai korvata kokonaan omilla ohjelmilla, käyttäen samoja työkaluja kuin Android-kehittäjätkin käyttävät.

Applikaation ohjelmistokehys(Application Framework)-kerros sisältää kaikki Android-kehitykseen tarvittavat Java-pohjaiset nimiavaruudet ja luokat sekä erilaiset managerit ja palvelun tarjoajat.

Kirjasto(Libraries)-kerros tarjoaa peruspalveluita applikaatioille ja ajonaikaisille komponenteille. Palveluihin kuuluvat mm. erilaiset grafiikkatuet(2D,3D,OpenGL), median toisto, datan varastointi(SQLite) ja nettiselaimet.

Kaikki kirjastot ovat kirjoitetut C/C++-kielillä ja ne tulevat näkymään kehittäjille Applikaatioiden ohjelmistokehyksen kautta.

Ajonaikaiset komponentit(Android runtime)-kerros sisältää Dalvik Virtual Machinen eli DVM:n ja sitä tukevat Java-kirjastot ja API:t. Kaikki kirjastot ja API:t ovat avoimia, hyvin dokumentoituja ja käytettävissä niitä tarvitseville kehittäjille.

Dalvik Virtual Machine on Googlen avoimen lähdekoodin versio Java SE:n Virtual Machinesta. DVM on varta vasten suunniteltu ottamaan mähdollisimman paljon irti mobiilipuhelimien kaltaisista vähän muistia ja prosessoritehoa omaavista laitteista ja silti maksimoimaan normaalit, Java-ohjelmoinnista saatavat hyödyt.

DVM:n suorituskyky ja koko mahdollistavat jokaisen Android-applikaation saamaan oman instanssinsa Dalvikista. Tämä mahdollistaa applikaatioiden paremman saatavuuden ja turvallisuuden koska ne eivät jaa muistia keskenään eivätkä täten ole yhtä herkkiä toisten applikaatioiden ajonaikaisille kaatumisille.

Linux-ydin(Linux Kernel)-kerros hoitaa järjestelmän keskeisimpien palvelujen kuten turvallisuuden, muistinhallinnan ja prosessienhallinnan toteutuksen.

Ydin toimii myös laitteistoabstraktiokerroksena suojaamalla kehittäjiä lukuisten eri Android-laitteiden erikoisuuksilta varmistamalla että kaikki laitteiden perusominaisuudet suoritetaan samalla tavalla laitteesta riippumatta.

3 TYYPILLINEN ANDROID APPLIKAATIO

3.1 Peruskomponentit

Android-applikaatio koostuu neljän tyyppisistä peruskomponenteista: aktiviteeteista, palveluista, sisällön tuottajista ja lähetyksien vastaanottajista.

Aktiviteetit (Activities) ovat applikaatiokomponentteja jotka koostuvat yleensä yhdestä näkymästä ja siihen liittyvästä käyttöliittymästä.

Aktiviteetit ovat Android-applikaatioiden keskeisimpiä komponentteja ja jokainen Android-applikaatio vaatiikin vähintään yhden aktiviteetin toimiakseen. Suurimmassa osassa sovelluksista käytetään kuitenkin useampia aktiviteetteja joihin kyseisen ohjelman eri toiminnallisuudet on jaoteltu. Kuitenkin yhden näistä aktiviteeteista on oltava määritelty applikaation pääaktiviteetiksi joka tullaan käynnistämään ensimmäiseksi sovellusta käynnistettäessä.

Androidin perussääntönä aktiviteettien käytössä on että yksi aktiviteetti keskittyisi toteuttamaan yhtä toimintaa esimerkiksi niin että sähköposti-applikaatiossa yksi aktiviteetti näyttäisi listauksen kaikista löytyneistä posteista ja toinen näyttäisi yksittäisen postin sisällön.

Aktiviteetit kommunikoivat keskenään Intentien eli Aikomusten välityksellä, joista kerrotaan lisää myöhemmässä kappaleessa.

Palvelut(Services) ovat myös eräänlaisia aktiviteetteja mutta ne eivät tarjoa käyttäjälle varsinaista käyttöliittymää ja ne on suunniteltu pyörimään muun ohjelman taustalla tai kun palvelun laukaisseen ohjelman käyttö on lopetettu kokonaan. Tyypillinen palvelu

voi tarjota esimerkiksi musiikin toistoa jonkin applikaation suorituksen ajaksi tai järjestelmän taustalle soitettavaksi käyttäjän hoitaessa muita asioita laitteellaan.

Sisällön tuottajat(Content Providers) ovat pääasiallinen tapa jakaa dataa applikaatioiden välillä. Ne kapsuloivat kaiken saamansa datan ja tarjoavat lisäksi mahdollisuuden määritellä datan turvallisuusasteen. Sisällön tuottajien asiakas applikaatiot kommunikoivat tuottajien kanssa applikaation kontekstin tarjoaman ContentResolverin avulla.

Tyypillinen esimerkki sisällön tuottajan käytöstä on puhelimen puhelinluettelon välittäminen eri applikaatioiden välillä.

Lähetyksien vastaanottajat (Broadcast Receivers) vastaanottavat järjestelmän laajuisia tai applikaatioiden sisäisiä viestejä ja reagoivat niihin määrätyillä tavoilla.

Tyypillisiä järjestelmän laajuisia viestejä ovat ilmoitukset mm. akun loppumisesta, puhelun tulosta tai näytön sammumisesta. Tyypillisiä applikaation sisäisiä viestejä ovat ilmoitukset mm. seuraavan aktiviteetin käynnistyksestä tai jo aloitetun, erillisellä palvelulla tuotettavan musiikintoiston lopetuksesta.

3.2 Prosessit ja säikeet

Android-applikaatioiden ja sen kaikkien komponenttien odotetaan toimivan erillisinä muista järjestelmästä löytyvistä applikaatioista minkä lisäksi kaikki kauan kestävät ja muun applikaation pysäyttävät toiminnat tulisi suorittaa erillisinä applikaation muista toiminnoista.

Näiden asioiden toteutuminen varmistetaan prosessien ja säikeiden avulla.

3.2.1 Prosessit (Processes)

Jokaisella Android-applikaatiolla on oma Linux-prosessinsa jossa kaikki applikaation komponentit oletusarvoisesti toimivat.

Android antaa kuitenkin mahdollisuuden määrittää osan applikaatioiden komponenteista suoritettavaksi erillisissä prosesseissa. Tämä määritys tapahtuu applikaation Android-Manifest.xml-tiedostossa.

Kaikki Androidissa pyörivät prosessit sijoitetaan erityiseen pinoon niin että viimeiseksi käynnistetty prosessi tulee aina pinon päällimmäiseksi. Pinon koolle ei varsinaisesti ole rajoitusta mutta Android-laitteiden ollessa mobiililaitteita niillä on rajatumpi määrä RAM-muistia esimerkiksi pöytätietokoneisiin verrattuna mikä johtaa siihen että jos jokin toinen prosessi vaatii lisää muistia niin Android voi joutua sulkemaan joitakin jo käynnissä olevia prosesseja tätä tarjotakseen.

Lisää aiheesta kerrotaan prosessien elinkaari-kappaleessa.

3.2.2 Säikeet (Threads)

Prosessien lisäksi Android-applikaatioilla on vähintään yksi säie käytössään jota kutsutaan applikaation pääsäikeeksi tai UI säikeeksi ja joka käynnistetään ensimmäiseksi ohjelman käynnistyessä.

Kaikkia operaatioita ei aina voi kuitenkaan suorittaa itse pääsäikeessä sillä pidemmät laskutoimitukset saattavat pysäyttää pääsäikeen niin pitkäksi aikaa että Android alkaa epäilemään ohjelman lopullista kaatumista ja pyytää käyttäjää pakkosulkemaan applikaation. Vaikka näin ei kävisikään on pitkät tauot ohjelman suorituksessa silti käyttäjäystävällisyyden kannalta huono asia.

Näitä ongelmia varten Android antaa kehittäjien luoda uusia, taustalla pyöriviä säikeitä joissa raskaimmat operaatiot voidaan suorittaa ja joiden tulokset palautetaan pääsäikeelle. Palauttaminen tapahtuu erityisten Handler- eli käsittelijä-luokkien kautta, joista puhutaan enemmän myöhemmin.

3.3 Sovelluksen elinkaari

3.3.1 Prosessin elinkaari

Prosessien sulkemisjärjestyksen määrittämistä varten ne on jaettu seuraavaan viisi-osaiseen tärkeyshierarkiaan.

Etualalla olevat prosessit on näkyviä ja toiminnassa olevia prosesseja joiden isännöimiä komponentteja käyttäjä tarvitsee suorittaessaan sen hetkistä toimintaansa.

Näkyvät prosessit ovat prosesseja jotka ovat käynnissä mutta joiden komponentit eivät juuri nyt näy käyttäjälle mutta jotka kuitenkin vaikuttavat tai saattaisivat vaikuttaa etualalla olevan prosessin toimintaan.

Palveluita suorittaviin prosesseihin kuuluvat kaikki palveluita suorittavat prosessit joilla ei ehkä ole suoraa vaikutusta etualalla tai näkyvissä oleviin prosesseihin mutta jotka kuitenkin suorittavat toimintoja joista käyttäjä saattaisi olla kiinnostunut kuten taustamusiikin toistoa tai datan lataamista netistä.

Taka-alalla olevat prosessit eivät vaikuta yllä olevissa kategoriossa mainittuihin prosesseihin millään lailla ja ne voidaan täten lopettaa milloin tahansa. Nämä prosessit jaetaan vielä niiden viimeisen käyttöajan mukaan omaan tärkeysjärjestykseensä niin että viimeiseksi käytetyt prosessit lopetetaan viimeisiksi.

Tyhjät prosessit ovat prosesseja joilla ei ole yhtään aktiivisia komponentteja ja ainoa syy minkä takia niitä saatetaan pitää vielä elossa on prosessien seuraavan käynnistyskertojen ajan lyhentämiseksi.

Se mihin yllämainituista luokista mikäkin prosessi kuuluu riippuu tämän prosessin isännöimien aktiviteettien, palvelujen tai lähetyksien vastaanottajien tiloista.

3.3.2 Aktiviteetin elinkaari

Aktiviteetin elinkaari koostuu neljästä vaiheesta aktiivinen, pysäytetty, tauotettu ja lopetettu.

Aktiviteetti on *aktiivisessa tilassa* silloin kun se on käynnissä ja se pyörii jäjestelmän etualalla eli sillä on fokus. Tämä koskee myös niitä aktiviteetteja jotka ovat aikaisemmin olleet tauotetussa tai pysäytetyssä tilassa mutta ovat nyt käynnistetty uudestaan ja siirretty jälleen järjestelmän etualalle. Tässä tilassa olevia aktiviteetteja isännöivät prosessit ovat etualalla olevassa tilassa.

Tauotetussa tilassa oleva applikaatio on käynnissä ja näkyvissä mutta se ei ole käytettävissä esimerkiksi koska käyttäjälle on esitetty notifikaatio joka on vienyt järjestelmän fokuksen ja johon hänen täytyy reagoida ennenkuin applikaatioon voidaan palata.

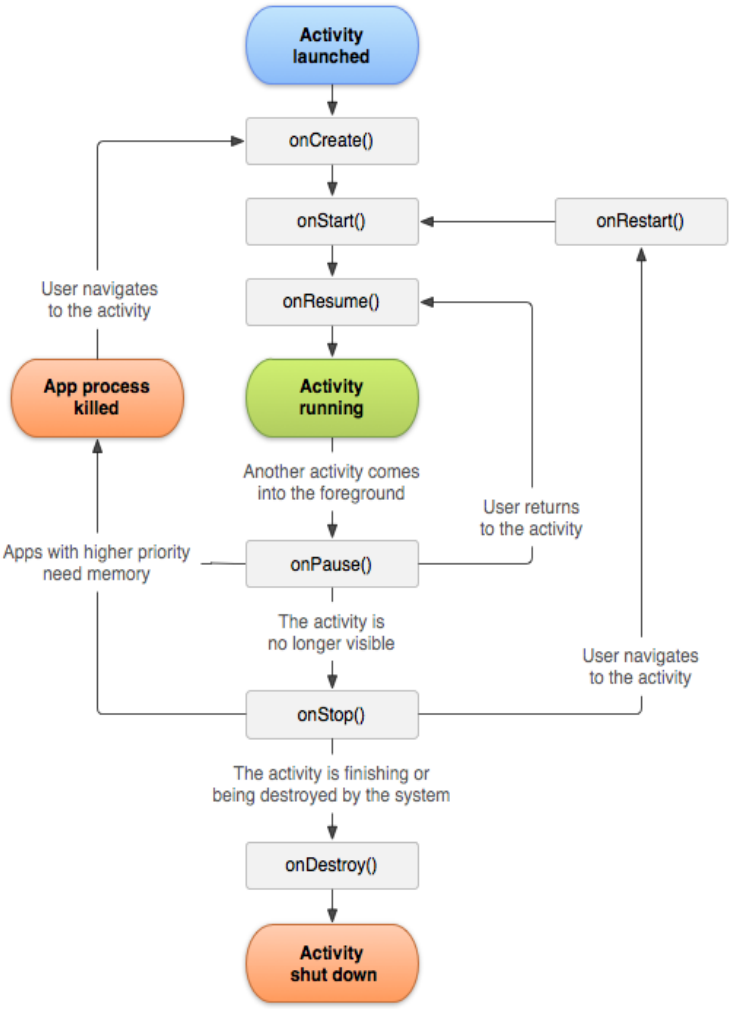
Tyypillinen applikaation tauottava notifikaatio on esimerkiksi puhelun tulo johon pitää joko vastata tai jättää huomiotta. Tauotettuja aktiviteetteja isännöivät prosessit ovat näkyvässä tilassa.

Pysäytetty tila on samankaltainen kuin tauotettu tilakin mutta erona on että applikaatio on nyt siirretty järjestelmän taka-alalle koska toinen applikaatio on vienyt sen fokuksen. Applikaatio voi kuitenkin vielä kommunikoida käyttäjän kanssa notifikaatioiden avulla.

Tässä tilassa olevaa aktiviteettia isännöivä prosessi lasketaan taka-alalla olevaksi prosessiksi.

Lopetetussa tilassa oleva applikaatio on nimensä mukaisesti sammutettu tai sitä ei ole aloitettu ollenkaan. Sammutettua aktiviteettia isännöivän prosessin katsotaan olevan tyhjässä tilassa.

Applikaatiot voidaan ohjelmoida ottamaan huomioon aktiviteettien eri tilat niiden elinkaaren takaisinkutsujen avulla jotka esitellään kuvassa 2.



Kuva 2: Aktiveetin elinkaari

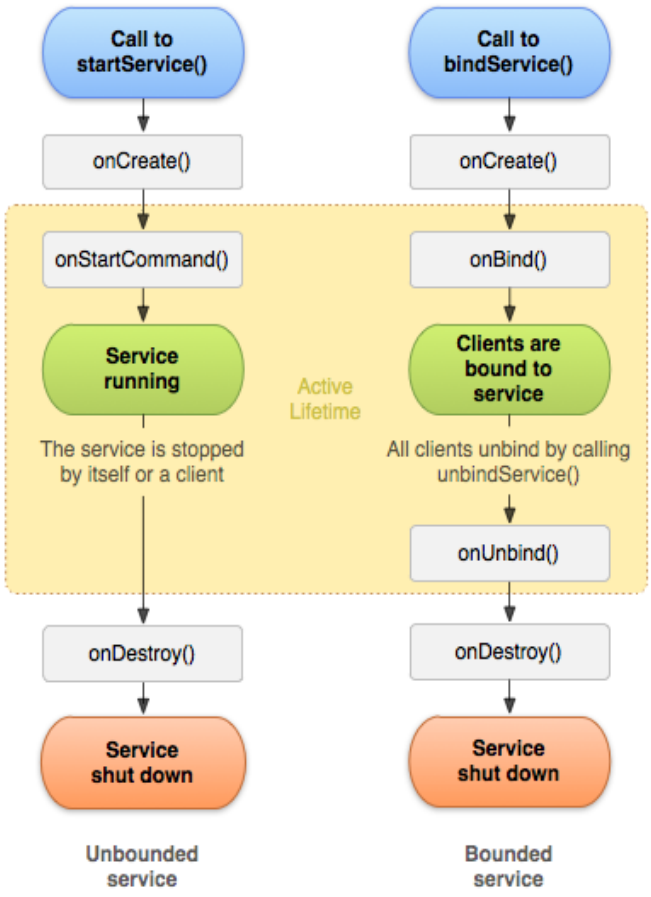
Aktiviteetin elinkaari alkaa onCreate()-takaisinkutsusta joka suoritetaan silloin kun uusi instanssi kyseisestä aktiviteetista luodaan ja päättyy onDestroy()-takaisinkutsuun jolloin aktiviteetti tuhotaan. Aktiviteetin näkyvä elinkaari alkaa kuitenkin onStart()-kutsusta ja päättyy onStop()-kutsuun. Tänä aikana aktiviteetti on näkyvissä käyttäjälle ja vastaa tämän pyyntöihin.

Etualalle palautetun aktiviteetin elinkaari ulottuu onResume()- tai onRestart()-kutsusta onPause()- tai onStop()-kutsuun joiden välisenä aikana aktiviteetille palautetaan ensisijainen fokus kaikista muista käynnissä olevista aktiviteeteista ja se vastaa taas käyttäjän pyyntöihin.

OnPause()- ja onStop()-kutsujen erona on että onPause():a kutsutaan silloin kun aktiviteetti menettää fokuksen vain hetkeksi ja onStop():ia silloin kun fokuksen menetys on pidempiaikaista.

3.3.3 Palvelun elinkaari

Palveluiden elinkaari selviää kuvasta 3.



Kuva 3: Palvelun elinkaari

Palvelujen elinkaari ulottuu siitä hetkestä kun ne ensimmäisen kerran luodaan startService()- tai bindService()-metodeja kutsuttaessa siihen hetkeen kun niiden stopService()-, stopSelf() tai unbindService()-metodeja kutsutaan.

StartService()-metodin ja bindService()-metodin erona on se että startService():ä kutsutaan silloin kun palvelun ja sen luoneen komponentin ei tarvitse kommunikoida keskenään ollenkaan palvelun luomisen ja tuhoamisen ohella ja bindService():ä kutsutaan silloin kun palvelua halutaan ohjata vielä silloinkin kun se on jo käynnistetty. Esimerkiksi musiikintoiston voi käynnistää startService():llä silloin kun halutaan soittimen toistavan jotakin tiettyä ennalta määrättyä kappaletta mutta jos toisto halutaan pysäyttää jossain kohtaa tai siirtää eteen- tai taaksepäin täytyy palvelu aloittaa bindService():llä.

Jos prosessi isännöi onCreate()- tai onStart()-metodejaan suorittavia palveluita lasketaan se silloin palveluita suorittavaksi prosessiksi ja se pyritään pitämään käynnissä niin kauan kunnes palvelu on lopetettu.

3.3.4 Lähetyksien kuuntelijan elinkaari

Lähetyksen kuuntelijat ovat aktiivisia vain sen ajan kun niiden viestit vastaanottavan onReceive()-metodin suorittaminen kestää. Prosessit jotka isännöivät onReceiver()-metodinsa sisältämää koodia suorittavia lähetyksien vastaanottajia ovat järjestelmän etualalla eikä niitä tulla lopettamaan ennenkuin kaikkein pahimmissa muistin loppumisesta johtuvista uhkatilanteissa.

3.4 Muita tärkeitä käsitteitä

3.4.1 Aikomukset(Intents)

Aikomukset ovat Androidin eri applikaatioiden, aktiviteettien ja palveluiden välinen kommunikointitapa.

Aikomusten tarkoitus on yksinkertaistaa Android-laitteissa navigointia ja datan käsittelyä tarjoamalla helpon ja yleisen kommunikointitavan laitteiston eri ohjelmien ja komponenttien väliselle viestinnälle.

Aikomukset koostuvat pääasiassa kahdesta, informaatiota sisältävästä osasta; toiminnan kuvauksesta ja datasta, johon toiminta kohdistuu.

Toiminta eli action on abstrakti kuvaus operaatiosta joka halutaan suorittaa. Tyypillisiä toimintoja ovat esimerkiksi ACTION_VIEW, ACTION_EDIT ja ACTION_MAIN.

Data-osuus taasen sisältää polun dataan jolle toiminta halutaan suorittaa ja joka usein kirjoitetaan URI-muotoon.

Tyypillinen toiminta/data-pari voisi olla esimerkiksi *ACTION_VIEW* ja *content://contacts/people/1* , jolla pyydetään näyttämään kontaktiluettelosta löytyvän henkilön 1 tiedot käyttäjälle.

Aikomukset voidaan lähettää kahdella tavalla, implisiittisesti tai eksplisiittivesti.

Ohjelman lähettäessä aikomuksen implisiittisesti aikomus annetaan Android-järjestelmälle käsiteltäväksi, joka tutkii kaikki laitteesta löytyvät applikaatiot, palvelut ja lähetysten vastaanottajat ja valitsee näistä sen joka pystyisi parhaiten halutun toiminnan suorittamaan.

Prosessin helpottamiseksi ohjelmilla on mahdollisuus rekisteröidä oma aikomusten suodattajansa(Intent Filter), joka kertoo mitä toimintoja ohjelma pystyy suorittamaan.

Eksplisiittistä tapaa käytetään silloin kun tiedetään tarkalleen mikä aikomuksen kohdeohjelma on ja se lähetetään suoraan kyseiseen kohteeseen.

3.4.2 AndroidManifest.xml

Jokaisella Android-applikaatiolla täytyy olla oma XML-muotoinen manifesti-tiedostonsa nimeltään Androidmanifest.xml joka ilmoittaa sitä lukevalle järjestelmälle mm. mitä komponentteja sovellus sisältää, mitä oikeuksia applikaatiolla on järjestelmän muihin ominaisuuksiin nähden kuten esimerkiksi pystyykö se muodostamaan internet-yhteyden sekä mikä on applikaation minimi API-taso ja käyttääkö applikaatio muita API:ja kuten esim. Googlen karttapalveluita.

Jos applikaatiolla on useita aktiviteetteja niin tämä tiedosto myös kertoo mikä näistä aktiviteeteista toimii pääaktiviteettina.

3.4.4 Käsittelijät (Handlers)

Handlerit eli käsittelijät mahdollistavat viestien ja ajettavien objektien lähetyksen ja vastaanoton niiden omistavien säikeiden viestijonon(MessageQueue) kautta.

Käsittelijöillä on kaksi tehtävää; ne mahdollistavat viestien käsittelyn tai ajettavien objektien ajon aikataulutuksen myöhemmäksi tai aikaisemmaksi ja niillä pystytään siirtämään samaiset tehtävät toisten säikeiden toteutettaviksi.

3.4.5 Resurssit

Resurssit ovat koodista erillään olevia applikaation elementtejä kuten kuvia, ikoneita tai äänitiedostoja. Resurssit säilytetään erillisissä kansioissa, joiden nimet kertovat mitä resursseja ne sisältävät ja jotka löytyvät projekti-hakemistosta res-kansion sisältä.

Jokaiselle kehittäjän määrittämälle tai lisäämälle resurssille luodaan oma ID:nsä jonka avulla kyseiseen resurssiin voidaan viitata koodista. Esimerkiksi jos kehittäjä luo uuden layout.xml tiedoston nimeltään main.xml luo järjestelmä uuden R.Layout.main nimellä tunnettavan ID:n.

Olennaisimpia resurssi-tyyppejä ovat mm. Layoutit, Values, Raw ja Drawabled-resurssit.

Layoutit ovat xml-tiedostoja jotka määrittävät applikaation käyttöliittymän visuaalisten elementtien kuten nappien ja tekstikenttien sijoittelun näytöllä. Yksi layoutti määrittää yhden näkymän mutta applikaatio voi vaihdella useiden eri layoutien välillä.

Vaikka näkymät voidaan määritellä ohjelmallisestikin niin xml-pohjaisia layouteja suositellaan silti käyttämään koska näkymien muokkaaminen keskitetysti omassa tiedostossaan on helpompaa kuin tehdä sama asia eri puolilla koodi-pohjaa.

Tyypillisiä layoutteja ovat mm. LinearLayout, RelativeLayout, TableLayout ja FrameLayout.

Values-resurssit kantavat tietoa applikaation stringeistä, väreistä, tyyleistä, ulottuvuuksista ja taulukoista(W. Frank Ableson, Robi Sen & Chris King & C. Enrique Ortiz 2012, 90).

Raw-resurssit tarkoittavat tiedostoja joita voidaan lukea streameinä applikaation suorituksen aikana. (Ableson ym. 2012, 90).

Drawables-resursseiksi katsotaan kaikki applikaation piirrettäväksi tarkoitetut elementit kuten PNG- tai JPG-kuvat, 9-Patch kuvat ja erilaiset gradientit joita voidaan mm käyttää applikaation käynnistysikoneina tai taustakuvina (Ableson ym. 2012, 90).

4 BLUETOOTH

Tämä kappale on kirjoitettu Bluetooth essentials for programmers(Albert S. Huang & Larry Rudolph, 2007)-kirjan pohjalta.

4.1 Yhteyden muodostuksen perusperiaatteet

Bluetooth on standardi sitä tukevien laitteiden langattoman kommunikoinnin mahdollistamiseen lähietäisyyksiltä.

Kahden laitteen muodostaessa Bluetooth-yhteyttä välilleen toinen laite tulee ensin muodostamaan ulospäin suuntautuvan yhteyden ja toinen sisäänpäin suuntautuvan yhteyden. Ulospäin suuntautuvan yhteyden muodostavaa laitetta kutsutaan asiakkaaksi eli clientiksi ja sisäänpäin menevää yhteyttä muodostavaa palvelimeksi eli serveriksi.

Kuitenkin palvelimella ja asiakkaalla tulee molemmilla olemaan omat sisäänpäin ja ulospäin suuntautuvat yhteytensä joten nimitykset tapahtuvat ainoastaan sen perusteella kumpi lähettää ensimmäisen, kommunikaation aloittavan data-paketin, tämän tullessa sen jälkeen tunnetuksi yhteyden asiakaspuolena.

Yhteyden muodostusta varten laitteiden täytyy olla tietoisia toistensa Bluetooth-osoitteista, käytettävästä lähetysprotokollasta sekä etsittävän palvelun palvelutunnuksesta.

4.2 Bluetooth-osoite

Bluetooth-osoite on Bluetooth-laitteiden Bluetooth-mikropiiriin painettu ja jokaiselle eri laitteelle ominainen 48-bittinen koodi. Osoitetta tullaan käyttämään Bluetooth-kommunikaation jokaisessa vaiheessa, alemman tason radio-protokollista korkeamman tason applikaatio-protokolliin.

Koska Bluetooth-osoite on 48-bittinen koodi jota ihmisen voi olla vaikea lukea tai edes muistaa käytetään osoiteena sen sijaan usein laitteelle annettavaa selkokielistä nimeä kuten MyBluetoothDevice, jonka laitteet kääntävät takaisin oikeaan muotoon samalla tavalla kuin internetissäkin sivujen käyttäjille näkyvät DNS-osoitteet(esim. www.google.com) käännetään koneiden lukemiksi IP-osoitteiksi.

Asiakaslaitteet eivät ole aina etukäteen tietoisia etsittävän laitteen osoitteesta joten käyttäjältä pyydetäänkin usein määrittämään kohdelaitteen selkokielinen nimi, jonka laitteet sitten muuttavat takaisin alkuperäiseksi 48-bittiseksi koodiksi kun ne alkavat etsiä oikeata palvelinlaitetta.

4.3 Protokollan valinta

Bluetooth-protokollia on useita erilaisia. Tyypillisiä protokollia ovat mm. RFCOMM, L2CAP,ACL ja SCO.

Protokollan valintaan vaikuttavat kaksi tärkeintä ominaisuutta ovat niiden vakuudet ja semantiikat.

Vakuudet kertovat kuinka peräänantamattomasti protokolla yrittää lähettää dataa laitteelta toiselle.

Luotettava protokolla lähettää dataa ”toimitus tai kuolema”-periaatteen mukaan eli dataa lähetetään niin kauan kunnes se varmasti saapuu kohteeseensa ja kaikki data on lähetetty tai muutoin yhteys katkaistaan kokonaan kun taas paras yritys protokollat lähettävät dataa ainostaan tiettyyn pisteeseen asti eivätkä sen saavuttaneena välitä onko kaikki data lähetetty tai edes onnistuneesti vastaanotettu.

Semantiikka kertoo millä tavalla data lähetetään; paketti-muotoisena vaiko yhtenä virtana.

Paketti-muotoisena lähetetty data siirtyy yhteyskanavaa pitkin määrätyn kokoisina palasina kun taas virtapohjaisena lähetetyssä datassa ei tehdä eroa sen välillä missä yksi data-paketti loppuu ja toinen alkaa.

Semantiikalla ei käytännössä ole väliä koska pienellä säädöllä pakettipohjainen data voidaan saada toimimaan kuin virtapohjainen ja päinvastoin.

4.4 Palvelutunnus ja SDP-serveri

Palvelutunnus eli Service ID kertoo minkälaista palvelua asiakaslaite on hakemassa ja se annetaan palvelinlaitteen Service Discovery Protocol-serverille eli SDP-serverille, joka pitää kirjaa kaikista palvelinlaitteesta löytyvistä palveluista. SDP-serveri palauttaa tämän jälkeen yhteyttä haluavalle laitteelle kaikki sen antaman hakuparametrin mukaiset palvelut jotka palvelinlaitteelta löytyvät.

Näistä hakutuloksista asiakas valitsee oman palvelunsa minkä jälkeen SDP-rekisteri palauttaa sitä vastaavan portin, jonka kautta asiakas voi luoda yhteyden varsinaiseen palveluun.

Asiakaslaite ei siis ota suoraan yhteyttä palvelinlaitteesta löytyvään palveluun vaan yhteys otetaan ensin palvelut listaavaan palveluserveriin, joka palauttaa halutun palvelun porttinumeron.

Tämä tehdään sen takia että Bluetooth voi näin pitää palvelun ja sen tarvitsevan portin erillään toisistaan niin kauan kunnes palvelua tarvitaan, jolloin se valitsee dynaamisesti yhden vielä vapaana olevista porteista palvelun käyttöön välttäen näin mahdolliset porttikonfliktit muiden palveluiden kanssa, jotka kovakoodattuja portteja käytettäessä saattaisivat varata samat portit toistensa kanssa.

Jotta haettavan palvelun tunnus löytyisi SDP-serveriltä täytyy palvelinlaitteen se rekisteröidä sinne. Palvelutunnus täytyy olla siis molempien laitteiden tiedossa ennen yhteyden muodostusta.

SDP-serverille rekisteröityminen vaatii palvelulta kahden parametrin välitystä. Parametrit määrittävät palvelun selkokielisen nimen ja tämän varsinaisen palvelutunnuksen mikä on palvelun itsensä muista palveluista erottava merkkijono. Siinä missä palvelun selkokielinen nimi voi olla pelkkä palvelun nimi kuten MyMusicPlayer niin Service ID:n pitää olla mahdollisimman erottuva minkä vuoksi ID:nä käytetäänkin usein UUID:tä.

4.5 Yhteyden muodostus asiakaslaitteen ja palvelinlaitteen välillä

Kommunikointi kahden Bluetooth-laitteen välillä tapahtuu pistokkeiden kautta.

Pistokkeiden luominen tapahtuu yksinkertaisesti Bluetoothin create()-komennolla, joka saattaa olla hiukan erilainen riippuen mitä kehityskieltä ja lähetysprotokollaa käytetään.

Myös pistokkeiden yhdistämisessä on eroavaisuuksia riippuen siitä onko kyseessä asiakaspistoke vai palvelinpistoke.

Asiakaspistokkeen yhdistäminen on verrattain yksinkertaista. Kun pistoke on luotu kutsutaan sen connect()-komentoa, joka ottaa parametreikseen kohdelaitteen osoitteen ja sen yhteydenottoja kuuntelevan portin. Laitteen oma BluetoothAdapter tulee hoitamaan kaiken muun prosessoinnin.

Palvelinpistokkeiden yhdistäminen tapahtuu kolmessa vaiheessa.

Ensimmäiseksi pistoke sidotaan paikallisiin Bluetooth-resursseihin kuten Adapteriin ja sille määrättyyn porttiin kutsumalla tämän bind()-komentoa.

Toiseksi kutsutaan pistokkeen listen()-komentoa joka asettaa pistokkeen kuuntelutilaan odottamaan asiakaslaitteiden yhteydenottoja.

Viimeiseksi obtain()-komennolla hankitaan yhdistetty pistoke jonka kautta dataa päästään siirtämään.

Erona asiakaspistokkeisiin, yhteyden muodostus vaiheen alussa luotuja palvelinpistokkeita ei kuitenkaan tulla itsessään käyttämään datan siirtoon tai vastaanottoon. Sen sijaan kun palvelinpistoke on havainnut uuden asiakaslaitteen luo se tälle oman Bluetooth-pistokkeensa joka tulee siitä lähtien hoitamaan kaiken asiakaslaitteen ja palvelinlaitteen välisen kommunikoinnin palvelinpistokkeen palatessa takaisin kuuntelu tilaan uusia yhteyspyyntöjä varten.

4.5 Kommunikointi ja yhteyden sulkeminen

Kommunikointi yhdistettyjen Bluetooth-pistokkeiden kautta tapahtuu yksinkertaisesti Bluetoothin write()- ja read()-komentoja kutsumalla. Write()-komento lähettää bittejä ulospäin ja read()-komennolla taasen vastaanotetaan bittejä.

Sen verran täytyy kuitenkin vielä muistaa että lähetys-komennolla lähetetyt bitit siirtyvät vain applikaation muistiosoitteesta järjestelmän puskureihin eivätkä ne siis ole välttämättä vielä lähteneet itse laitteelta. Vastaanotto-komento palauttaa aina bittejä ja jos yhteys on katkennut komento palauttaa 0 bittiä.

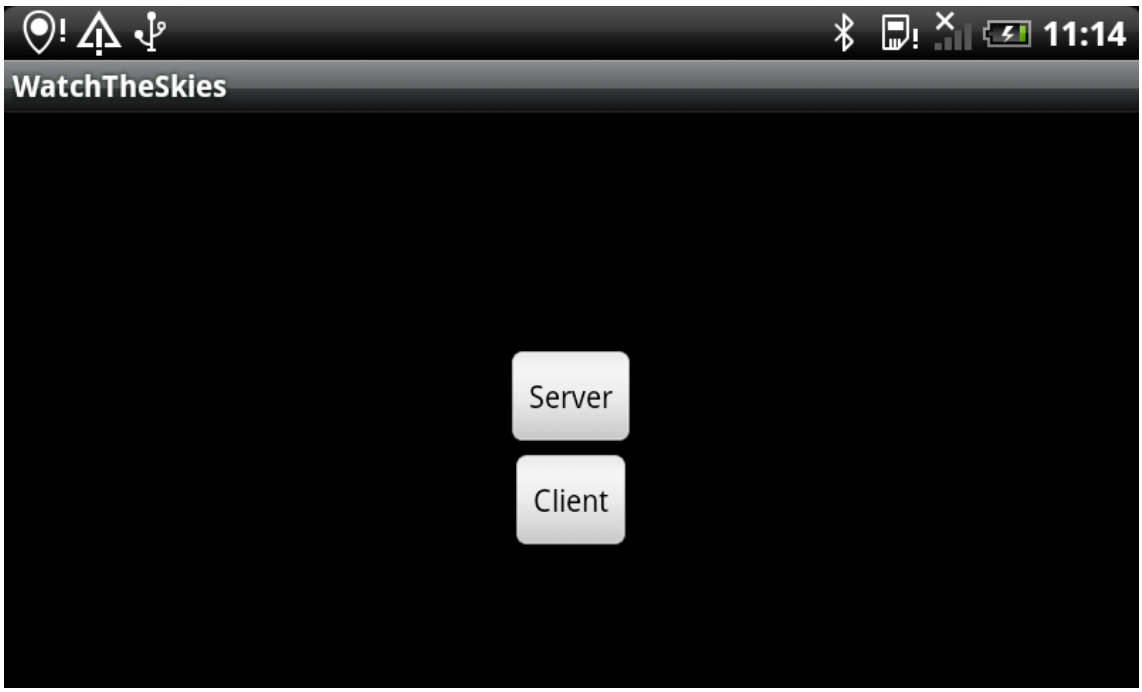
Yhteys suljetaan kutsumalla close()-komentoa, joka irrottaa luodut pistokkeet. Palvelin-pistokkeen sulkeminen merkitsee palvelinlaitteen portin vapauttamista ja sisäänpäin tulevien viestien kuuntelun lopetusta.

5 KOMPONENTTIEN KÄYTÄNNÖN TOTEUTUS

5.1 Esimerkki ohjelman esittely

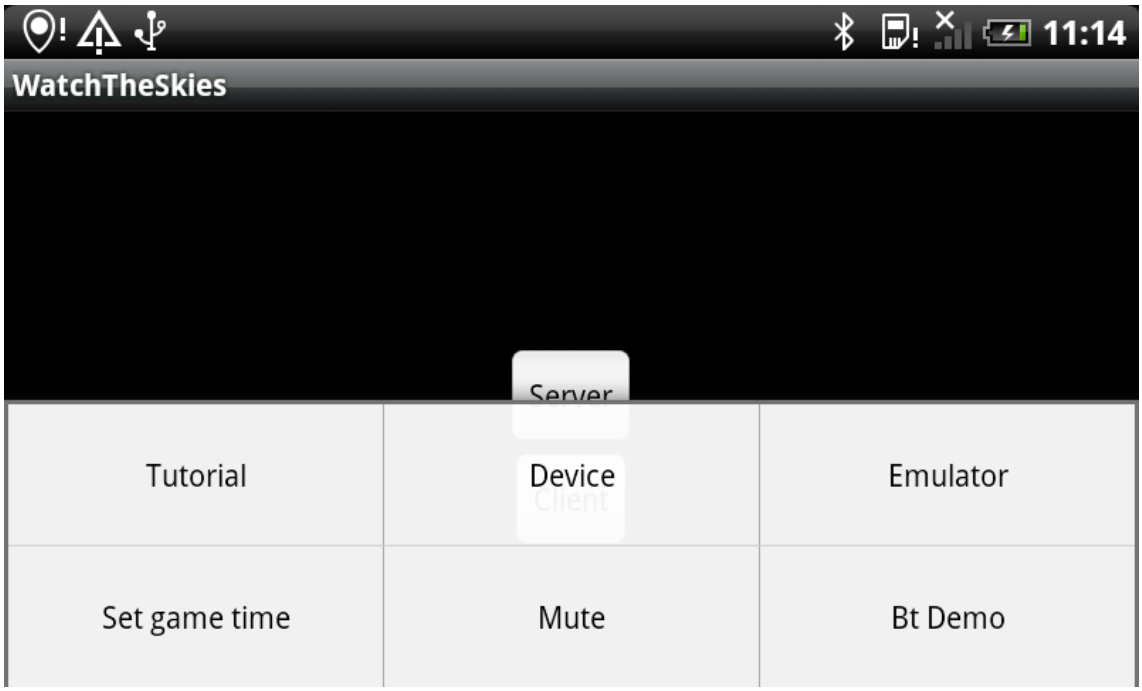
Ohjelma koostuu kahdesta päänäkymästä: alkuvalikosta ja varsinaisesta pelinäkymästä. Lisäksi applikaatiossa on yksi seekbar:in sisältämä näkymä ja kaksi Bluetooth-yhteyteen liittyvää näkymää.

Alkuvalikko koostuu kuvasta 4 näkyvistä kahdesta painonapista joista käyttäjä saa valita pelimuodon sekä erillisestä optionsmenusta joka avautuu Android-laitteen menunappia painamalla. Painonapeilla valitaan tuleeko laite toimimaan Bluetooth-yhteyden palvelin- vai asiakaspuolena mikä vaikuttaa myös siihen kumpaa roolia käyttäjä tulee itse pelissä pelaamaan, asiakkaan pelatessa aina tykkinä ja palvelimen ufona.



Kuva 4: Alkuvalikon painonapit

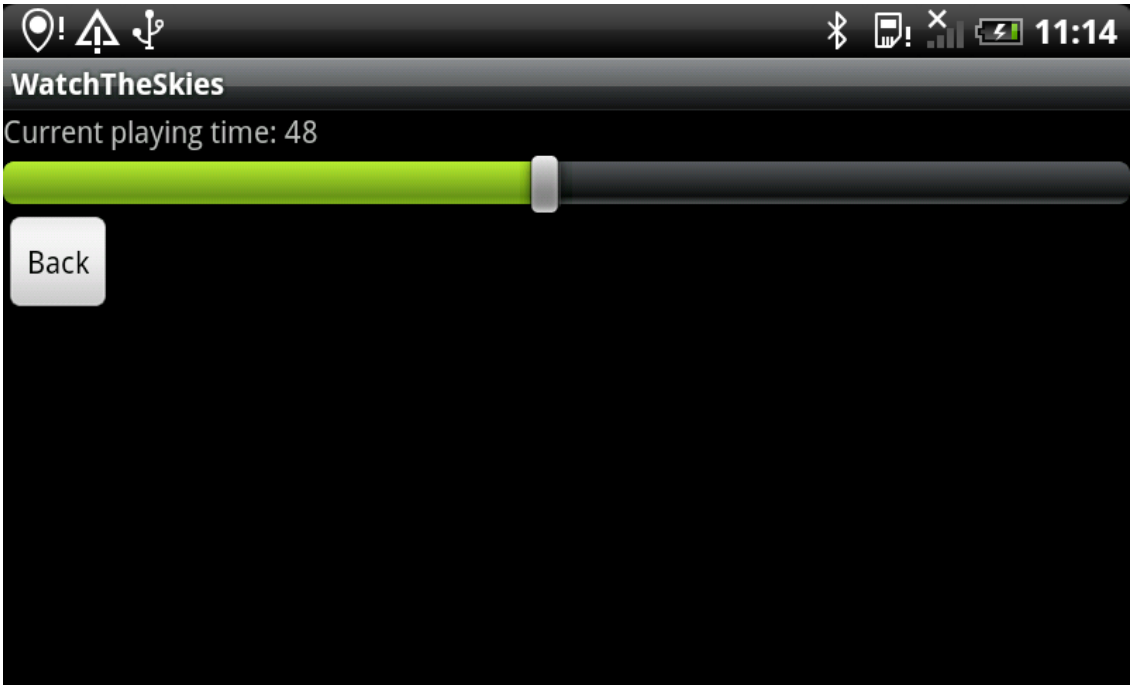
Painamalla Android-laitteen menu-nappia aukeaa kuvan 5 esittämä optionsMenu-valikko.



Kuva 5: OptionsMenu

Device ja Emulator vaihtoehdot määrittävät käynnistetäänkö peli Bluetooth-yhteydellä vai ei, Devicen vastatessa yhteydellistä tilaa ja Emulatorin yhteydetöntä. Lisäksi Emulator-tilassa ufo piirretään näytölle heti pelin käynnistyessä koska muuten ufo ei piirtyisi ruudulle ennenkuin sitä ohjastava pelaaja sitä ensimmäisen kerran liikuttaisi, millä tällä tavalla estetään toista pelaajaa ampumasta ufoa ennenkuin ensimmäinen pelaaja on valmis aloittamaan pelin.

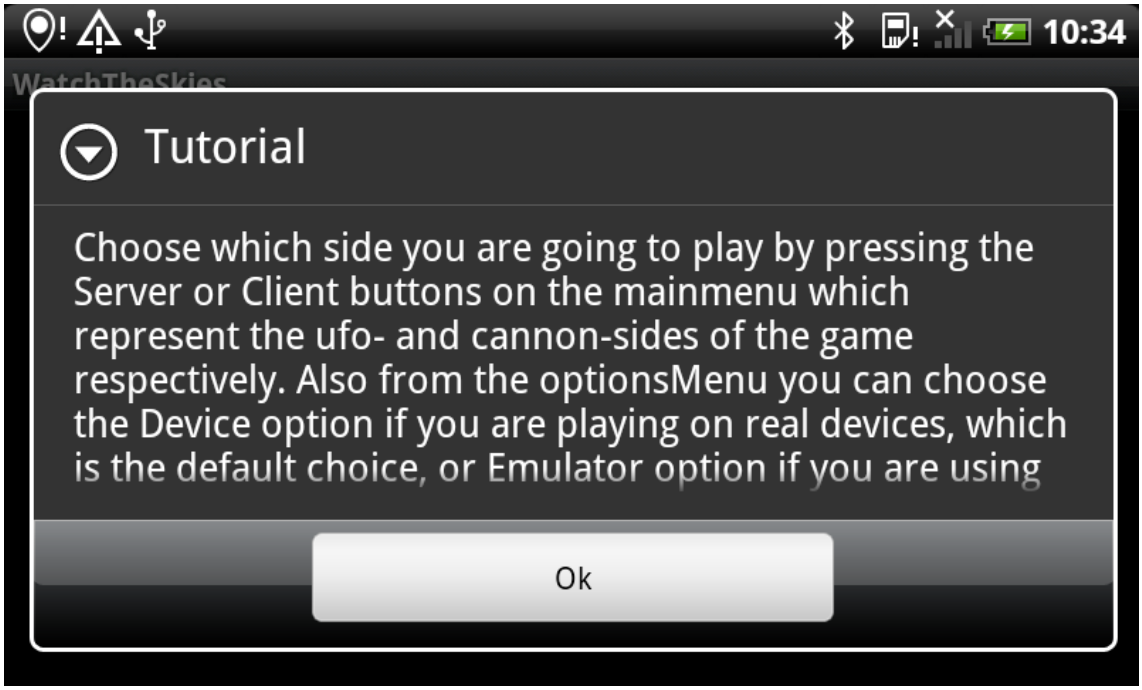
Set Game Time- vaihtoehdosta vaihtuu näkymä hetkeksi toiseen, kuvan 6 esittämään näkymään jossa käyttäjälle tarjotaan sormella asetettavaa seekbaria millä määritetään pelin kestoaika sekunneissa mitattuna. Takaisin päävalikkoon pääsee Back-nappia painamalla.



Kuva 6: Set Game Time-näkymä

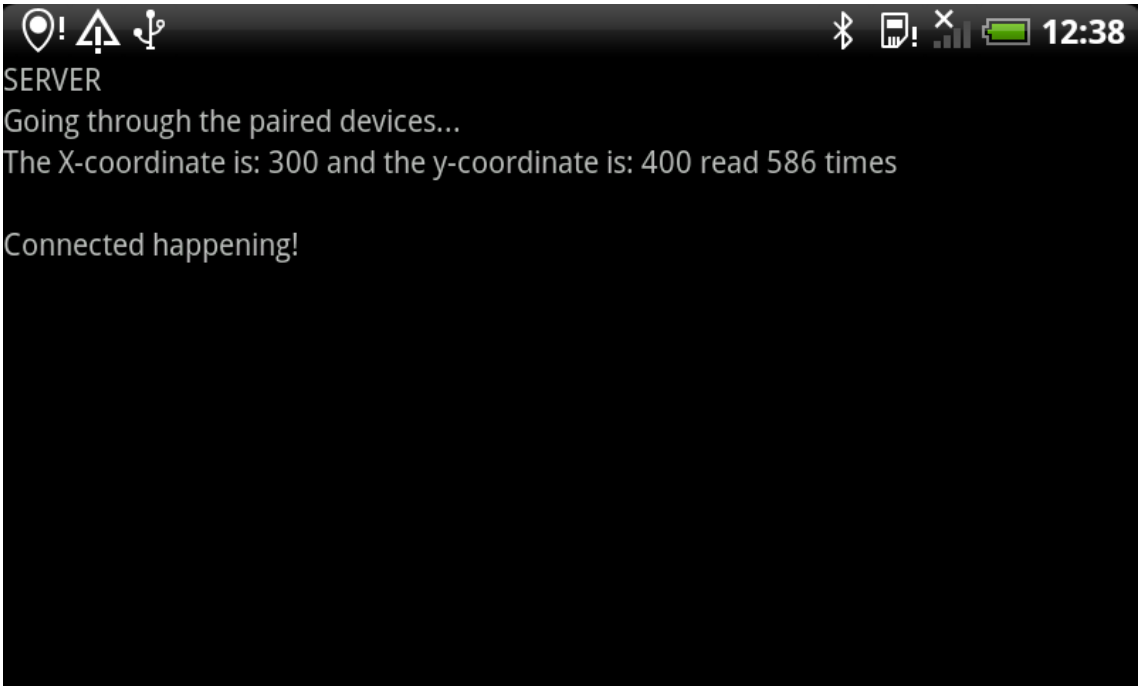
Mute-vaihtoehto yksinkertaisesti kytkee äänentoiston poispäältä.

Tutorial-vaihtoehto sisältää pelin käyttöohjeet jotka annetaan pelaajalle kuvassa 7 näkyvän yksinkertaisen AlertDialogin-sisällä. Painamalla OK pääsee takaisin alkuvalikkoon.

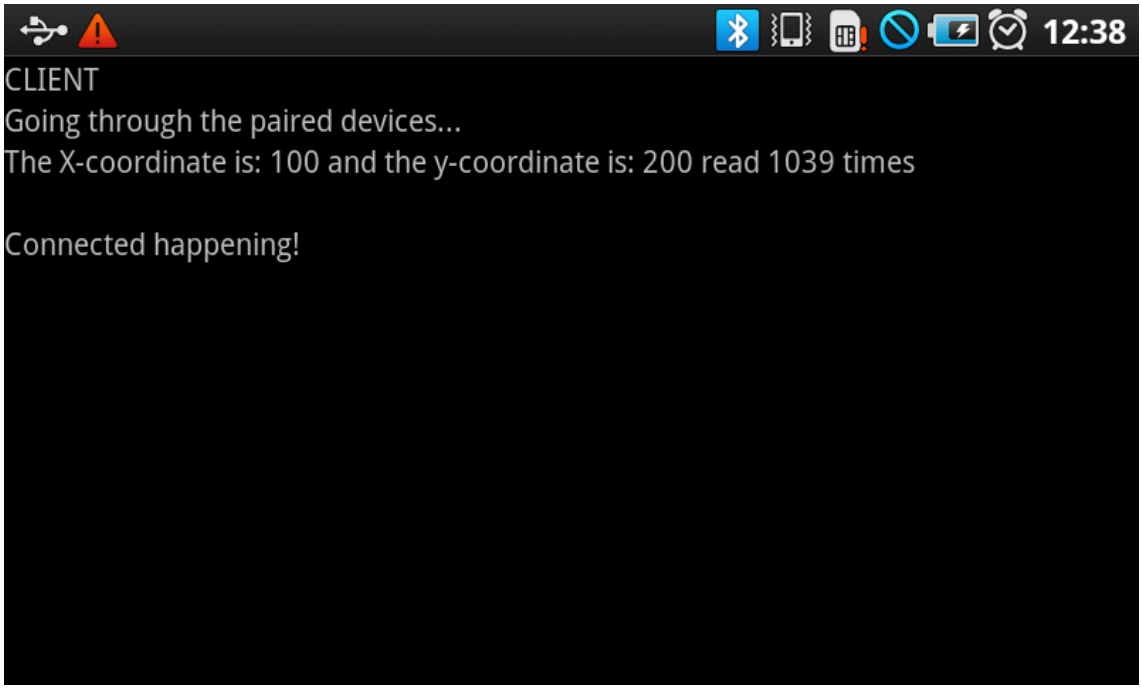


Kuva 7: Pelin tutoriaali

BT Demo-vaihtoehto on tarkoitettu Bluetooth-yhteyden testausta ja esittelyä varten. Valitsemalla tämän tilan peliä ei käynnistetä ollenkaan vaan Bluetooth-yhteyden yli lähetetään pelkkää testi-dataa, jonka tarkoitus on osoittaa että yhteys toimii ja sen yli pystytään lähettämään pelin tarkoituksia varten oikean muotoista informaatiota. BtDemo-moodi tapahtuu kuvien 8 ja 9 esittämissä näkymissä.



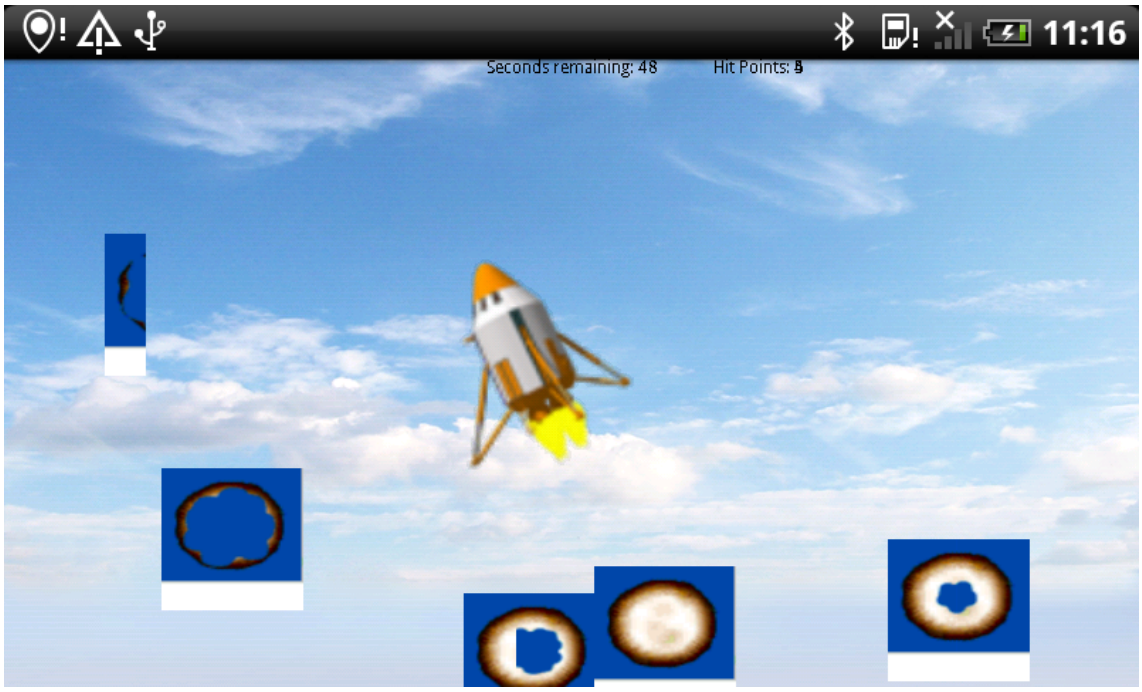
Kuva 7: BtDemo-näkymän palvelinpuoli



Kuva 8: BtDemo-näkymän asiakaspuoli

Lisäksi kun Bluetooth-yhteyttä muodostetaan kahden etälaitteen välille peliä pelattaessa käyttäjälle esitellään yksinkertainen muutamasta textvievistä koostuva näkymä joka kertoo missä vaiheessa Bluetooth-yhteyden muodostaminen on.

Ohjelman päänäkymä koostuu kuvasta 9 näkyvistä jpeg-muotoisesta taustasta, ufosta, räjähdyksistä(kun ne tapahtuvat), sekä alaspäin laskevasta kellosta ja ufon hitpointseista kertovista tekstikentistä.



Kuva 9: Pelinäkymä

Peliä pelataan kosketusnäytön avulla. Ufoa liikutetaan painamalla ensin sormi aluksen kuvakkeen päälle ja liuttamalla sitten sormea näyttöä pitkin aluksen seuratessa perässä.

Jos ufoa pelaava pelaaja koskettaa toisella sormellaan näyttöä jo yhden sormen ollessa painettuna piirretään näytölle valeufo jonka tarkoituksena on vaikeuttaa toisen pelaajan oikean ufon tunnistamista.

Tykillä ammutaan koskettamalla haluttua kohtaa ruudulla jolloin kyseisessä kohdassa tapahtuu räjähdystapahtuma tai jos kosketus tapahtuu ufon kuvakkeen sisäpuolella ufolta poistetaan yksi terveyspiste ja jos ne ovat kaikki jo poistettu niin alus tuhotaan.

Itse pelin aikana taustalla soi taustamusiikki minkä lisäksi tykin räjähdyksillä ja ufon tuhoutumisella on omat äänitehosteensa. Menussa ja päävalikon painonapeilla on myös omat ääniefektinsä jotka toistetaan näitä painettaessa.

5.2 Peruskomponentit

Ohjelma koostuu kahdesta aktiviteetista: MainMenu ja Game-aktiviteetista joista Main-Menu toimii applikaation aloitusaktiviteettina. Nimensä mukaisesti aktiviteetit edustavat alkuvalikkoa ja itse peliä.

Aktiviteettien lisäksi applikaatiolla on Bluetooth-komponentti jonka välityksellä applikaation palvelin- ja asiakaspuolet kommunikoivat toistensa kanssa.

5.3 Mainmenu-aktiviteetti

MainMenu-aktiviteetin koodi kokonaisuudessaan löytyy liitteestä 3.

Alkuvalikon painonapit ovat nimeltään server- ja clientButton.

```
private Button clientButton,serverButton;
```

Molemmat napit tarvitsevat omat kosketustenkuuntelijansa jotka rekisteröidään Button-luokan setOnTouchListener()-metodin avulla.

```
serverButton.setOnTouchListener(new OnTouchListener(){});
```

SetOnTouchListener:llä on onTouch()-niminen takaisinkutsu joka suoritetaan kun kuuntelija on havainnut kosketustapahtuman siihen rekisteröidyssä painonapissa.

```
public boolean onTouch(View v, MotionEvent e){}
```

Metodissa olevassa ehdossa parsitaan metodille parametrinä tulleesta MotionEvent e-oliosta sen havaitseman tapahtuman tyyppi.

Esimerkki applikaatio on ohjelmoitu reagoimaan ACTION_UP-nimellä tunnettavaan tapahtumaan joka ilmoittaa koska käyttäjä on nostanut sormensa näytöltä. Tapahtuman toteutuessa määritetään data-muuttujaan arvoksi 1 tai 2, toistetaan asianmukainen ääniefekti ja siirrytään activate()-metodiin joka käynnistää seuraavaan aktiviteetin.

```
if(v==clientButton && e.getAction()==MotionEvent.ACTION_UP){
          data=2;
          mSound.PlayButtonClick();
          activate();
          return true;
}
```

Data-muuttuja kantaa tiedon siitä kumpaa napeista on painettu 1 edustaessa palvelinta ja 2 asiakasta.

5.3.1 OptionsMenu

Painonappien tavoin jokaisella optionsmenun valinnalla on omat integer-muuttujansa jotka kertovat onko kyseinen valinta valittu vai ei. Esimerkiksi tieto siitä pelataanko peliä oikeilla laitteilla vai emulaattorilla kulkee hardware-muuttujan mukana, 1 edustaessa emulaattoria ja 2 oikeata laitetta.

Valikot määritellään onCreateOptionsMenu()-metodissa.

```
public boolean onCreateOptionsMenu(Menu menu);
```

Eri vaihtoehdot lisätään valikkoon kutsumalla menu-olion add()-metodia:

```
menu.add(0,MENU_TUTORIAL,0,R.string.tutorial);
```

MENU_TUTORIAL viittaa luokan alussa Tutorial-valinnalle määriteltyyn integer-tunnukseen ja R.string.tutorial resurssi-hakemiston string-alahakemistossa ja string-xml-tiedostossa määriteltyyn nimeen jolla valinta esitetään valikossa. Ensimmäinen parametri kertoo mihin joukkoon lisättävä vaihtoehto kuuluu ja toisella määritellään mikä järjerstysnumero sillä tulee olemaan valikossa. Koska molemmat ovat nollia tarkoittaa se että valinta ei kuulu mihinkään laajempaan joukkoon ja että valinnat sijoitetaan siihen järjestykseen valikkoon missä ne on lisättykin.

Tutoriaali näytetään käyttäjälle sitä varten luodussa AlertDialogissa, joka luodaan onOptionsItemSelected()-metodissa, MENU_TUTORIAL-casen kohdalla.

```
public boolean onOptionsItemSelected(MenuItem item){
        switch(item.getItemId()){
        case MENU_TUTORIAL:
        }
}
```

AlertDialog luodaan sen Builder-alaluokan avulla, joka sijoitetaan omaan Builder-olioonsa nimeltä builder. Parametrinä AlertDialogin builderille annetaan applikaation konteksti.

```
Builder builder = new AlertDialog.Builder(this);
```

SetTitle()-metodilla asetetaan ikkunalle otsikko joka tässä tapauksessa on siis "Tutorial".

```
builder.setTitle("Tutorial");
```

Varsinainen sisältö dialogille annetaan setMessage()-metodilla.

```
builder.setMessage("...");
```

SetPositiveButton()-metodilla asetetaan dialogin positiivisen napin teksti, joka tässä tapauksessa on "Ok" sekä sille oma kosketuksenkuuntelijansa.

```
builder.setPositiveButton("Ok", new DialogInterface.OnClickListener(){

            public void onClick(DialogInterface dialog, int id) {
                    dialog.cancel();
             }
});
```

Dialogi esitetään käyttäjälle show()-metodilla.

```
builder.show();
```

Optionsmenun Set Game Time-vaihtoehdosta aukeaa uusi näkymä jossa käyttäjä pääsee muuttamaan pelin aikarajaa näkymästä löytyvän SeekBar:in avulla.

Näkymä vaihdetaan setContentView()-metodilla.

```
setContentView(R.layout.seekbar);
```

Tämän jälkeen alustetaan SeekBar-olio sBar SeekBarin määrittämällä xml-tiedostolla timerbar ja säädetään palkin maksimipituudeksi 100-yksikköä ja aloituskohdaksi 10.

```
sBar=(SeekBar) findViewById(R.id.timerbar);
sBar.setMax(100);
sBar.setProgress(10);
```

SeekBar tarvitsee oman muutoksien kuuntelijansa joka tulee kutsumaan Seekbarin onStartTrackingTouch()-, onStopTrackingTouch()- ja onProgressChanged()- takaisinkutsuja havaitessaan metodien nimien mukaiset tapahtumat.

```
sBar.setOnSeekBarChangeListener(new OnSeekBarChangeListener() {

      @Override
      public void onStopTrackingTouch(SeekBar seekBar) {}
      public void onStartTrackingTouch(SeekBar seekBar) {}
      public void onProgressChanged(SeekBar seekBar, int progress,
                                    boolean fromUser) {}
});
```

5.3.2 Ääniefektit

Painonappeja ja optionsmenun eri vaihtoehtoja painettaessa toistetaan lyhyt klik-tyyppinen ääniefekti. Efektin toistamiseen käytetään Androidin SoundPool-soitinta jota hoitaa MenuSound-luokka.

SoundPool-luokasta tehdään sP-niminen olio jolle annetaan parametreiksi integeri määrittämään kuinka monta yhtäaikaista streamia soitin saa soittaa, itse stream-tyyppi jota soitin toistaa ja viimeiseksi tämän streamin laatu.

```
sP=new SoundPool(4, AudioManager.STREAM_MUSIC,10);
```

Stream-tyypiksi on valittu STREAM_MUSIC sen ollessa Androidin kehittäjien nimenomaan peleille suosittelema tyyli.

Äänitiedostot sijoitetaan omaan HashMappiinsa, jolle annetaan parametriksi kaksi integeria joista ensimmäinen toimii tunnuksena äänitiedostolle ja toinen on polku resurssikansioon josta tiedosto löydetään. Äänitiedoston tunnuksena toimii mClick-muuttuja jonka arvo on 0 ja itse äänitiedoston polku annetaan sP-olion load()-metodin palauttamana, jolle annetaan parametreiksi applikaation konteksti, äänitiedoston polku ja äänen prioriteetti.

```
soundsMap=new HashMap<Integer,Integer>();
soundsMap.put(mClick, sP.load(c,R.raw.sound4,1));
```

Efektit toistetaan kutsumalla sP-olion play()-metodia.

```
sP.play(soundsMap.get(mClick), 10, 10, 1, 0, 1);
```

Parametreinä metodille annetaan efektin äänitiedosto HashMap-olio soundsMapista, vasemman ja oikean kaiuttimen äänenvoimakkuus, toiston tärkeysjärjestys, integeri kertomaan toistetaanko ääniefekti uudestaan 0 ollessa kielteinen vastaus ja 1 myönteinen sekä toiston taso joka on 0.5:stä 2:teen.

5.4 Aktiviteettien vaihto

Painamalla jompaa kumpaa painonapeista siirtyy sovellus MainMenu-aktiviteetista Game-aktiviteettiin.

Vaihto tapahtuu Intentin eli aikomuksen avulla, joka lähetetään activate()-metodissa.

Uusi Intent-olio luodaan tällä tavalla:

```
Intent newIntent=newIntent(this.getApplicationContext(),Game.class);
```

Ensimmäinen parametri antaa metodille applikaation kontekstin ja toinen määrittää Intentin kohteen joka tässä tapauksessa on siis Game-luokan määrittämä aktiviteetti.

Tieto käyttäjän alkuvalikossa tekemistä valinnoista siirretään MainMenu-aktiviteetista Game-aktiviteetille käyttäen Bundle-karttaa, joka laitetaan Intentin-sisälle putExtras()-komennolla.

```
Bundle bundle=new Bundle();
bundle.putInt("data", data);
newIntent.putExtras(bundle);
```

Put*()-metodeissa ensimmäinen parametri on avain jolla toisena parametrinä annettava data tai informaatio pystytään löytämään Bundle-kartasta. Put*()-metodien variaatioita ovat mm. PutBoolean(), PutInt(), PutString() ja PutFloat().

5.5 Bluetooth-yhteyden luonti esimerkki applikaatiossa

Bluetooth-komponentin koodi löytyy kokonaisuudessaan liitteestä 4.

Mikäli pelitilaksi on valittu Device tai BtDemo aletaan aktiviteetin vaihdon jälkeen muodostaa Bluetooth-yhteyttä.

Ennenkuin applikaatio voi käyttää Bluetoothia täytyy sille määrätä Androidin vaatimat käyttöoikeudet applikaation AndroidManifest-tiedostossa.

```
<uses-permission android:name="android.permission.BLUETOOTH_ADMIN" />
<uses-permission android:name="android.permission.BLUETOOTH" />
```

Laitteen Bluetooth tuen olemassaolo varmistetaan samalla kun kutsutaan laitteen BluetoothAdapterin getDefaultAdapter()-komentoa, joka rekisteröi adapterin tämän applikaation käyttöön tai palauttaa null-arvon jollei laitteessa ole tukea Bluetoothille.

```
bAdapter=BluetoothAdapter.getDefaultAdapter();
```

Jos getDefaulAdapter() palauttaa null-arvon lopetetaan ohjelman suoritus tähän paikkaan mikä tapahtuu kutsumalla Androidin yleistä, aktiviteettien lopettamista varten tarkoitettua finish()-komentoa.

```
if(bAdapter==null){
        finish();
}
```

Seuraavaksi tarkastetaan onko laitteen Bluetooth-ominaisuudet kytkettynä päälle kutsumalla BluetoothAdapterin isEnabled()-metodia joka palauttaa true-arvon jos Bluetooth on päällä ja false-arvon jos Bluetooth ei ole päällä.

```
if(!bAdapter.isEnabled()){};
```

Jollei Bluetoothia ole kytketty päälle käynnistetään se nyt.

Bluetoothin kytkeminen päälle koodin tasolla tapahtuu lähettämällä uusi Intentti BluetoothAdapterille ACTION_REQUEST_ENABLE-toiminnalla varustettuna jonka tuloksena käyttäjältä pyydetään Adapterin itse luoman dialogin muodossa lupaa käynnistää laitteen Bluetooth-ominaisuudet.

```
Intent enableBt=new Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
startActivityForResult(enableBt,0);
```

StartActivityForResult()-metodi käynnistää sille ensimmäisenä parametrinä annetun Intentin määräämän aktiviteetin ja ottaa vastaan tämän aktiviteetin palauttaman datan onActivityResult()-takaisinkutsullaan kun käynnistetty aktiviteetti on suorittanut toimintansa. Toisella parametrillä yksilöidään tämä nimenomainen kutsu erotuksena muista mahdollisista aktiviteetin aloitus pyynnöistä.

Kun Bluetooth on käynnistetty onnistuneesti onActivityResult()-metodi vastaanottaa RESULT_OK-nimisen integerin minkä jälkeen aloitetaan laitteiden etsintä kutsumalla applikaation startDiscovery()-metodia.

```
protected void onActivityResult(int requestCode, int resultCode,
                                Intent data){

          if(resultCode==RESULT_OK && requestCode==0){
                    startDiscovery();
           }
};
```

Etsintä voi tapahtua kahdella eri tavalla; joko etsimällä jo aikaisemmin löydettyjä ja sidottuja laitteita tai etsimällä kokonaan uusia laitteita.

Kaikki laitteen jo muistissa olevat etälaitteet saadaan BluetoothAdapterin getBondedDevices()-metodin palauttamana ja ne sijoitetaan erityiseen Set-taulukkoon josta ne ovat helposti läpikäytävissä.

```
Set<BluetoothDevice> pairedDevices=bAdapter.getBondedDevices();
```

Setissä olevat laitteet käydään läpi for-loopissa jossa jokaisen laitteen kohdalla kutsutaan laitteiden getName()-metodia jonka palauttamaa nimeä verrataan etsittävän laitteen nimeen. Jos tällä nimellä oleva laite löytyy siirrytään yhteyden muodostukseen.

```
for(BluetoothDevice device: pairedDevices){
        if( device.getName().toString().equals("Q"){}}
```

Esimerkki ohjelma olettaa että etsittävän laitteen nimenä on Q, joten molempien peliä pyörittävien laitteiden nimeksi täytyy manuaalisesti vaihtaa Q laitteiden Bluetooth Settings-valikosta ennen kuin peliä pystytään pelaamaan.

Jollei oikeata laitetta löydy muistista aletaan niitä etsimään ympäristöstä.

Ensimmäiseksi määritetään oma laite toisten Bluetooth-laitteiden löydettäväksi mikä tapahtuu lähettämällä BluetoothAdapterille ACTION_REQUEST_DISCOVERABLE-pyynnöllä alustettu intentti.

```
Intent dIntent=
new Intent(BluetoothAdapter.ACTION_REQUEST_DISCOVERABLE);
```

Aikomuksen onnistunut lähetys palauttaa Androidin itse tekemän dialogin jossa käyttäjältä pyydetään lupaa tehdä laite etälaitteille näkyväksi.

Laitteiden etsintä alkaa BluetoothAdapterin startDiscovery()-komennolla, joka palauttaa booleanin ilmoittamaan oliko etsintä tuloksellinen vai ei.

Tätä varten aktiviteetti tarvitsee oman lähetysten vastaanottajansa, jonka onReceive()-metodi tulee nappaamaan kyseisen viestin ja käsittelemään sen asianmukaisella tavalla.

```
private final BroadcastReceiver bReceiver= new BroadcastReceiver(){
          @Override
          public void onReceive(Context c, Intent i){}
          }
```

Jotta oikea viesti pystyttäisiin nappaamaan kaikista mahdollisista viesteistä mitä järjestelmässä saattaa liikkua, täytyy applikaatiota varten luoda uusi IntentFilter, joka siivilöi kaikki lähetetyt viestit ja alustaa se kuuntelemaan haluttua lähetettä.

```
IntentFilter f=new IntentFilter(BluetoothDevice.ACTION_FOUND);
```

Suodatin annetaan BroadCastReceiverille samalla kun se rekisteröidään.

```
this.registerReceiver(bReceiver,f);
```

Napattava viesti on nimeltään ACTION_FOUND ja kun se on saatu kiinni käynnistetään joko BtServerThread tai BtClientThread, riippuen siitä onko laite palvelin- vai asiakastilassa.

```
public void onReceive(Context c, Intent i){
          String action=i.getAction();
          if(BluetoothDevice.ACTION_FOUND.equals(action)){}
}
```

5.5.1 Asiakasyhteyden muodostus

Asiakasyhteys muodostetaan BtClientThread-nimisessä säikeessä.

Pistoke palvelinlaitteeseen luodaan Game-luokasta saadusta BluetoothDevicestä löytyvän createRfcommSocketToServiceRecord()-metodin avulla, joka ottaa parametrikseen asiakas- ja palvelinlaitteille yhteiseksi määritellyn UUID:n.

```
bSocket=device.createRfcommSocketToServiceRecord(MY_UUID);
```

Yhteys palvelinlaitteelle muodostetaan pistokkeen connect()-metodilla ja pistoke suljetaan close()-metodilla joko silloin kun applikaatio lopetetaan tai jos connect()-metodi ei pystynytkään muodostamaan yhteyttä.

```
try {
          bSocket.connect();

}         catch (IOException e) {
                    e.printStackTrace();
                    try {
                           bSocket.close();

                     } catch (IOException e1) {
                              e1.printStackTrace();
                          }
};
```

Ennen connect()-metodin kutsua varmistetaan että BluetoothAdapterin etsintä-operaatiot ovat pysäytetyt kutsumalla vielä kerran adapterin cancelDiscovery()-metodia. Tämä tehdään sen takia että operaatiot syövät niin paljon Androidin resursseja että ne hidastaisivat liikaa varsinaisen yhteyden muodostusta.

```
bAdapter.cancelDiscovery();
```

BtClientThread käynnistää operaatioidensa onnistuneen suorituksen jälkeen BtManager-Thread-säikeen joka hoitaa varsinaisen yhteyden ylläpidon.

5.5.2 Palvelinyhteyden muodostus

Palvelinyhteys muodostetaan BtServerThread-säikeessä.

Palvelinpistoke luodaan listenUsingRfcommWithServiceRecord()-komennolla joka ottaa parametrikseen UUID:n lisäksi string muotoisen palvelunimen SDP-serveriä varten.

```
ServerSocket=
btAdapter.listenUsingRfcommWithServiceRecord(NAME,MY_UUID);
```

Yhteys muodostetaan BluetoothServerSocketin accept()-metodilla joka palauttaa onnistuessaan yhdistetyn BluetoothSocketin jonka kautta varsinainen viestien lähetys tulee tapahtumaan.

```
socket=serverSocket.accept();
```

Pistokkeen sulkeminen tapahtuu close()-metodilla.

BtServerThread käynnistää operaatioidensa onnistuneen suorituksen jälkeen BtManagerThread-säikeen joka hoitaa varsinaisen yhteyden ylläpidon.

5.5.3 Bluetooth-yhteyden ylläpito

BtManager-säie huolehtii viestien lähetyksestä ja vastaanotosta.

Viestien kuuntelu ja lähetys tapahtuu InputStream:in ja OutputStream:in avulla, jotka saadaan Bluetoothpistokkeen getInputStream()- ja getOutputStream()-metodien palauttamana.

```
InputStream tmpIn=null;
OutputStream tmpOut=null;

          try {
               tmpIn=bS.getInputStream();
               tmpOut=bS.getOutputStream();
             } catch (IOException e) {
               e.printStackTrace();
                }

iStream=tmpIn;
oStream=tmpOut;
```

Sisääntulevat viestit käsitellään InputStreamin read()-metodin avulla ja OutputStreamin write()-metodilla viestit lähetetään ulospäin.

5.5.4 Viestien muuttaminen biteiksi ja takaisin

Laitteiden välillä kulkevat viestit ovat kaksi alkioisia integer-taulukoita jotka sisältävät peliobjektien x- ja y-koordinaatit. Ennenkuin viestit voidaan lähettää täytyy ne kuitenkin muuttaa bittimuotoon ja vastaanoton jälkeen vastaavasti takaisin niiden alkuperäiseen muotoon.

Muuttaminen tapahtuu Game-luokasta löytyvien int2Byte()- ja byte2Int()-metodien avulla mutta koska metodien toiminta ei kuulu opinnäytetyön aihepiiriin ei niitä käsitellä raportissa tämän enempää.

5.5.5 Säikeiden välinen kommunikointi

Koska säikeet eivät voi suoraan kommunikoida toistensa kanssa niin tätä varten Game-aktiviteetille on tehty oma viestienkäsittelijänsä eli Handlerinsa joka tulee hoitamaan BtManager-säikeen ja pääsäikeen välisen viestinnän.

Ensimmäiseksi Game-luokkaan luodaan Handler-luokasta handler niminen olio ja sen jälkeen oma versio Handlerin viestit nappaavasta handleMessage()-metodista.

```
private final Handler handler=new Handler(){
          @Override
          public void handleMessage(Message msg){}
};
```

Handlerille tarkoitetuille viesteille annetaan omat integer-tunnuksensa joiden avulla ne voidaan erottaa toisistaan.

Viestit käsitellään käyttäen switch/case-rakennetta. Esimerkiksi START_THE_GAME-nimellä kulkeva viesti käsitellään näin:

```
public static final int START_THE_GAME=7;

@Override
public void handleMessage(Message msg){
            switch (msg.what) {
            case START_THE_GAME:
            startGame();
               break;
              }
}
```

Handleri tullaan antamaan parametrinä Bluetooth-säikeiden rakentajille ja viestit lähetetään käyttäen Handlerin obtainMessage()-metodia.

```
mHandler.obtainMessage(Game.COORD_GOT,bytes,-1,buffer).sendToTarget();
```

Ensimmäinen parametri ilmoittaa integerin muodossa mikä viesti on kyseessä. Toinen ja kolmas parametri ovat viestin mukana tulevia argumentteja ja viimeinen on objekti joka tässä tapauksessa on vastaanotettu viesti bittimuodossaan.

5.3 Game-aktiviteetin yleisarkkitehtuuri

Koska tämän opinnäytetyön tavoitteena on ensisijaisesti Android- ja Bluetooth-kehitykseen tutustuminen jätetään seuraavissa kappaleissa esimerkki ohjelman muut kuin näihin liittyvät osa-alueet käsittelemättä. Kuitenkin lukijalle voi olla helpompi seurata tekstiä kun hän ymmärtää esimerkki ohjelman koodin toiminnan laajemmallakin tasolla.

Tätä varten tässä kappaleessa käydään nopeasti läpi ohjelman yleinen arkkitehtuuri ja toimintatapa.

Tässä kappaleessa käsiteltävät luokat löytyvät liitteen 2 sisältämistä luokkakaavioista.

Panel- ja siihen kuuluva CanvasThread-luokka vastaavat applikaation näytön päivityksestä ja siihen kuuluvien objektien piirtokutsuista. Tämän lisäksi Panel-luokassa vastaanotetaan kaikki kosketusnäyttö-tapahtumat ja lähetetään ne edelleen GameObjectManagerille sekä alustetaan pelin ajanlaskin ja musiikkiraita, joita hoitavat GameTimer- ja SoundTrack-luokat. Panel-luokassa myös alustettava Painter-luokka vastaa pelin voittoilmoitusten maalaamisesta ruudulle.

GameInfo-luokka(Liite 6) välittää MainMenu-aktiviteetissa tehdyt valinnat muille luokille sekä Ufon ja räjähdysten paikkakoordinaatit BtManagerille joka lähettää ne toiselle laitteelle silloin kun peliä pelataan Bluetooth-yhteydellä.

GameObjectManager-luokka(Liite 10) huolehtii Panel-luokan ja peliobjektien eli Ufo- ja Explosion-luokkien välisestä kommunikoinnista välittämällä Panel-luokassa tapahtuneiden kosketustapahtumien koordinaatit asianmukaiselle peliobjektille.

Koska räjähdyksiä tulee tapahtumaan useita ja ne päivittyvät eri tahtiin toistensa suhteen on niitä varten vielä luotu oma manageri-luokkansa nimeltä ExplosionManager(Liite 12) joka hoitaa eri räjähdysten luonnin ja päivityksen.

Ufolle ja Räjähdyksille kuuluvat omat ääniefektinsä joiden toistosta vastaavat UfoSoundEffects- ja ExplosionSound-luokat. ExplosionSound-luokkaa kutsutaan ExplosionManagerin puolelta silloin kun uusi räjähdysalustetaan kun taas UfoSoundEffects-luokkaa kutsutaan Ufo-luokasta silloin kun alukseen on osuttu tai se on tuhoutunut.

5.4 Game-aktiviteetti

Game-luokan koodi löytyy kokonaisuudessaan liitteestä 5.

Kun Bluetooth-yhteys on onnistuneesti luotu tai jos sitä ei luoda ollenkaan siirrytään varsinaiseen pelitilaan.

Ensimmäiseksi määritetään Game-aktiveetin näkymä pelinäkymäksi antamalla setContentView()-metodille parametriksi resurssien layout osiosta löytyvä game.xml-tiedosto.

```
setContentView(R.layout.game);
```

Varsinaista pelinäkymää ei määritellä kuitenkaan game.xml:ssä vaan se tehdään dynaamisesti Panel nimisessä luokassa johon Game.xml viittaa.

5.4.1 Pelinäkymä

Panel-luokan koodi löytyy kokonaisuudessaan liitteestä 7.

Panel-luokka periytyy SurfaceView:stä joka on View-luokan alaluokka ja tarkoitettu erityisesti nopeasti päivittyvien näkymien piirtoon. Panel käyttää myös SurfaceHolder-.callBack nimistä abstraktia käyttöliittymää joka mahdollistaa näkymän ja sen sisällön muokkaamisen.

Panel-luokkaan tehdään omat versiot View-luokan onDraw(), surfaceCreated(), surfaceChanged()- ja surfaceDestroyed()-metodeista. SurfaceCreated(),-Changed() ja Destroyed()-metodeja kutsutaan kun näkymä on piirretty, muutettu tai tuhottu. OnDraw()-metodi huolehtii sisällön piirtämisestä itse näkymään.

OnDraw()-metodia kutsutaan erillisestä säikeestä nimeltä CanvasThread, josta luodaan olio surfaceCreated()-metodissa. OnDraw():in kutsuminen erillisestä säikeestä on tarpeen koska pelien ja varsinkin arcade-pelien tyyliin kuuluu nopea ruudunpäivitys, joka ei saisi pätkiä silloinkaan kun ohjelma suorittaa toisia laskutoimituksia.

CanvasThreadissa luodaan myös Canvas johon piirtotapahtumat kohdistuvat ja joka annetaan onDraw():lle.

Canvasta ei kuitenkaan luoda suoraan Canvas-luokasta vaan se tehdään Panel-luokan surfaceHolderin ja sen lockCanvas()-metodin avulla. Tällä varmistetaan se ettei mikään toinen säie tai komponentti pysty piirtämään kyseiselle kankaalle ennenkuin se on asianmukaisesti vapautettu unlockCanvasAndPost()-komennolla.

```
Canvas c;
c = _surfaceHolder.lockCanvas(null);
_surfaceHolder.unlockCanvasAndPost(c);
```

5.4.2 Kosketusnäyttö

Kosketustapahtumat napataan View-luokkaan kuuluvan onTouchEvent()-metodin avulla. Metodin implementaatio löytyy Panel-luokasta. OnTouchEvent()-metodi ottaa parametrikseen MotionEvent muuttujan jonka avulla saadaan tietää mm. minkä tyyppinen liike on suoritettu ja missä kohdin näyttöä se on tapahtunut.

```
public boolean onTouchEvent(final MotionEvent ev){};
```

MotionEvent:ien käsittely tapahtuu switch/case-rakenteen avulla ja tarkasteltavia tapahtumia ovat ACTION_DOWN, ACTION_UP ja ACTION_MOVE, jotka kertovat koska sormi on ensimmäisen kerran koskenut näyttöä, koska se on siitä nostettu ja milloin sormea liutetaan näytön pinnalla.

```
switch (ev.getAction() & MotionEvent.ACTION_MASK) {

            case MotionEvent.ACTION_DOWN:{}
             case MotionEvent.ACTION_UP:{}
            case MotionEvent.ACTION_MOVE:{}

}
```

ACTION_MASK liittyy useamman yhtäaikaisen kosketuksen rekisteröimiseen näytöltä. Sitä käytetään erottamaan itse tapahtuma sen saamasta pointer-arvosta joka pitää lukua kaikista näytöllä tapahtuvista kosketustapahtumista.

Androidin havaitessa toisen kosketustapahtuman näytöllä jo yhden sormen ollessa painettuna lähettää se ACTION_POINTER_DOWN-ilmoituksen kertomaan tästä ACTION_DOWN:in sijaan. Esimerkki applikaatiossa toisen kosketustapahtuman tapahtuessa edellisen ollessa vielä voimassa käsketään GameObjectManageria piirtämään valeufo näytölle jollei sitä ole jo piirretty.

```
case MotionEvent.ACTION_POINTER_1_DOWN:{
     if(mirrorUfo==false){
        gManager.CreateMirrorUfo();
        mirrorUfo=true;
      }
}
```

5.4.3 Musiikki ja ääniefektit

Pelin taustamusiikin toisto käynnistyy heti kun Panel-luokan surfaceCreated()-metodia on kutsuttu millä varmistetaan että musiikintoisto alkaa nätisti samaan aikaan kun pelinäkymäkin on luotu. Toistosta huolehtii SoundTrack-luokka, joka alustetaan Panel-luokan rakentajassa.

Taustamusiikin toistossa käytetään Androidin omaa MediaPlayer-toistinta ja musiikkina on netistä löydetty The Thing-elokuvan pääteema mp3-muotoisena.

MediaPlayer alustetaan sen rakentajassa toistamaan haluttu tiedosto käyttämällä MediaPlayer-luokan create()-metodia, jolle annetaan toisena parametrinä tiedoston sijainti joka on tässä tapauksessa resurssien raw-kansio.

```
private static MediaPlayer mp;
new MediaPlayer();
mp=MediaPlayer.create(c, R.raw.music);
```

Musiikin toisto alkaa kun mp-olion start()-metodia kutsutaan mikä tapahtuu SoundTrack-luokan omassa start()-metodissa jota taasen kutsutaan Panel-luokan surfaceCreated()-metodissa.

Ääniefekteistä vastaavat UfoSoundEffects- ja ExplosionSound-luokat, jotka hoitavat nimiensä mukaisesti räjähdyksiin ja ufoon liittyvät ääniefektit.

Ufon ääniefekti toistetaan kun ufo on tuhoutunut ja se tapahtuu Ufo-luokan Destroyed()-metodissa jonka toteutus löytyy liitteestä 11.

Räjähdysten efektit toistetaan ExplosionManager-luokassa sen newExplosionEvent()-metodissa heti kun uusi räjähdys on luotu. ExplosionSound-luokan toteutus löytyy liitteestä 12.

Molemmista ääniefekteistä vastaa Androidin SoundPool-mediatoistin, jonka alustus ja suoritus toteutetaan samalla tavalla kuin MainMenu-kappaleessa käsitelty valikkoefektienkin toisto, joten näitä ei tässä kohdassa käsitellä sen tarkemmin.

5.4.4 Peliobjektit

Ufon ja räjähdysten piirrosta ja tilojen päivityksistä huolehtivat Ufo- ja Explosion-luokat.

Ufo-luokan koodi löytyy kokonaisuudessaan liitteestä 11.

Ufolla on kaksi png-kuvaa, jotka edustavat “tervettä ” ufoa ja tuhottua ufoa.

Kun ufon viisi terveyspistettä ovat kulutetut vaihdetaan kuvan 10 terveen ufon kuva kuvan 11 tuhoutuneeseen ufoon.

Kuva 10: Terveen Ufon png-kuva

Kuva 11: Tuhotun Ufon png-kuva

Ufon Drawablen alustus tapahtuu terveen ufon osalta Ufo-luokan rakentajassa ja tuhoutuneen ufon osalta luokasta löytyvässä Destroyed()-metodissa, jossa tapahtuu myös terveyspisteiden vähennys.

Molemmissa tapauksissa Drawablen alustus tapahtuu kutsumalla applikaation kontekstin getResources()- ja tämän getDrawable()-metodia, jolle annetaan parametriksi haettavan resurssin osoite ja nimi.

```
mDrawable=
cntxt.getResources().getDrawable(R.drawable.app_lunar_lander);
```

Drawableissa ei itsessään ole mitään tapaa rekisteröidä niihin tapahtuvia kosketuksia tai muita tapahtumia joten koskeeko pelaajat ufoa vai ei lasketaan vertaamalla GameObjectManagerin newRect()-metodin kautta välittämiä, kosketustapahtumien x- ja y-koordinaatteja ufon ufoRect-muuttujaan ja selvittämällä näiden avulla tapahtuivatko kosketukset tämän neliön sisä- vai ulkopuolella.

Ufo piirretään näytölle ufo-luokan draw()-metodissa kutsumalla Drawablen omaa draw()-metodia jolle annetaan parametriksi pelin yhteinen Panel-luokalta tullut piirtotapahtumia isännöivä Canvas. Ennen piirtoa Drawablelle täytyy antaa neliö jonka sisälle kuva piirretään. Neliönä käytetään aikaisemmin mainittua ufoRect:iä.

```
mDrawable.setBounds(ufoRect);
mDrawable.draw(canvas);
```

Explosion- ja ExplosionManager-luokkien koodit löytyvät kokonaisuudessaan liitteestä 12.

Räjähdystapahtumat ovat 9:stä eri png-kuvalla alustetusta Drawablesta koostuvia animaatioita. Png-kuvat ovat kaikki sijoitettu yhteen yksiulotteiseen integer-taulukkoon nimeltään exArray[] ja niiden toisto hoidetaan Explosion-luokan draw()-metodissa. Animaatio suoritetaan index-muuttujan avulla jota kasvatetaan yhdellä joka kerran kun metodia kutsutaan niin että metodi piirtää aina taulukossa seuraavana olevan kuvan.

```
private static int[] exArray={R.drawable.explosion1,R.drawable.explo-
sion2,R.drawable.explosion3,R.drawable.explosion4,R.drawable.explo-
sion5,R.drawable.explosion6,R.drawable.explosion7};
```

```
if(index<exArray.length){

            explosion=c.getResources().getDrawable(exArray[index]);
            explosion.setBounds(posX-50, posY-50, posX+50, posY+50);


            explosion.draw(canvas);
            frameIndex++;
}
```

Animaatioon kuuluvat eri kuvat näkyvät kuvassa 12.

Kuva 12: Räjähdysanimaation eri kuvat

Kun kaikki kuvat taulukosta ovat käyty läpi muutetaan boolean muuttuja done todeksi , mikä kertoo että räjähdystapahtuma on suoritettu loppuun eikä tätä räjähdystä tarvitse enää piirtää ruudulle.

Uudet räjähdys-oliot luodaan explosionManagerin newExplosionEvent()-metodissa ja ne säilötään explList-nimiseen ArrayList:iin. Listin sisältö käydään läpi update()-metodissa jossa suoritetaan olioiden draw()-metodien kutsut.

```
explList=new ArrayList<Explosion>();
explList.add(new Explosion(context,sHolder,gInfo));

for(int k=0;k<explList.size();k++){
          if(explList.get(k)!=null){
             explList.get(k).draw(c);
          }
}
```

5.4.5 Ajastin

Tässä kappaleessa käsiteltävien GameTimer- ja Painter-luokkien koodit löytyvät liitteestä 13.

Pelin ajastimesta vastaa GameTimer-luokka joka alustetaan Panel-luokan surfaceCreated()-metodissa.

Ajastimena käytetään Androidin CountDownTimer-luokkaa, josta luodaan timer niminen olio. CountDownTimerin rakentaja ottaa kaksi parametria jotka kertovat mistä luvusta aletaan laskea alaspäin ja kuinka monen millisekunnin välein vähennys tapahtuu. Alkuvalikossa määritelty pelinaikaraja saadaan GameInfo-luokalta kutsumalla sen getTimerData()-metodia.

```
gameTime=gInfo.getTimerData()*1000;
```

Ajastimen päivitysten väliseksi ajaksi määritetään 1000 millisekuntia eli yksi sekunti. Tämä tarkoittaa että gameTimerin onTick()-takaisinkutsua kutsutaan aina yhden sekunnin välein. OnTick()-metodissa alustetaan time-muuttuja takaisinkutsun palauttamalla, ajastimessa vielä jäljellä olevalla ajalla. Kun ajastin on suorittanut operaationsa loppuun kutsutaan gameTimerin onFinish()-metodia, jossa time-muuttuja alustetaan varmuuden vuoksi nollalla merkitsemään peliajan loppua.

```
timer= new CountDownTimer(gameTime, 1000){

          public void onTick(long millisUntilFinished) {

              if(millisUntilFinished<gameTime ||
                 millisUntilFinished==gameTime){
               time=millisUntilFinished;
                 }
            }
```

```
        public void onFinish() {
                    time=0;
                }
};
```

Ajastimen piirto näytölle tapahtuu Painter-luokassa, jota kutsutaan Panel-luokan onDraw()-metodissa. Painter-olion drawTimer()-metodi ottaa parametrikseen applikaatiossa käytettävän canvaksen ja sen hetkisen peliajan joka saadaan GameTimer-luokan getTime()-metodin palauttamana.

```
painter.drawTimer(mCanvas,timer.getTime());
```

Ajastin piirretään käyttäen Canvaksen drawText()-metodia. Piirtoa varten määritellään luokan rakentajassa uusi Paint-olio paint ja määritetään sen väriksi musta. Koska ajastin halutaan piirtää keskelle näytön yläreunaa kutsutaan Paint-luokan setTextAling()-metodia ja annetaan sille parametriksi CENTER mikä tarkoittaa että piirrettävä elementti sijoittuu horisontaalisesti drawText()-metodille annettujen x ja y-koordinaattien keskelle.

```
paint=new Paint();
paint.setColor(Color.BLACK);
paint.setTextAlign(Align.CENTER);

public void drawTimer(Canvas c,long time){
c.drawText("Seconds remaining: "+time/1000,x/2,10, paint);
}
```

Ajastin käynnistetään Panel-luokan onDraw()-metodissa kutsumalla Timer-luokan Start()-metodia joka taasen kutsuu CountDownTimer-olio timer:in omaa start()-metodia.

```
timer.Start();

public void Start(){
timer.start();
}
```

6. JATKOKEHITYS AJATUKSIA

Ajanpuutteesta ja tekijän omasta kokemattomuudesta johtuen kaikkia vaatimusmäärittelyssä annettuja ominaisuuksia ei pystytty toteuttamaan eikä ohjelman koodia ehditty optimoimaan loppuun asti. Tässä kappaleessa tullaan listaamaan puuttuneita ominaisuuksia ja mahdollisia ohjelmaan tehtäviä parannuksia.

Suurimpana ja kaikkein ilmeisimpänä puutteena on tietysti se että peli kaatuu sitä yritettäessä pelata Bluetooth-yhteyden yli. Kuitenkin opinnäytetyön tavoitteisiin nähden jotka liittyivät Android- ja Bluetooth-kehitykseen tutustumiseen ja Bluetooth-yhteyden kuitenkin toimiessa BtDemo-moodia käytettäessä voi asiaa katsoa siltä kannalta että Bluetoothin toimimattomuus itse pelin kanssa ei ole työn kaatava puute, koska se liittyy enemmän pelilogiikan heikkouksiin kuin applikaation Android tai Bluetooth puolen toteutukseen. Tämän ja ajanpuutteen vuoksi ongelma päätettiin jättää ratkaisematta.

Esimerkki applikaatiota ajavat etälaitteet tunnistavat toisensa niiden Bluetooth-nimiensä perusteella Bluetooth-yhteyttä muodostattaessa. Tällä hetkellä applikaatio vaatii että nimi pitää olla valmiiksi määriteltynä laitteiden Bluetooth-settings valikossa. Nimen voisi asettaa myös koodin kautta kutsumalla laitteen setName()-metodia mutta koska tämä asia tuli tietoon niin myöhäisessä työprosessin vaiheessa ja koska näinkin yksinkertaisen koodin lisäämisessä tälläiseen monimutkaiseen ohjelmaan saattaa olla kauaskantoiset seuraukset, päätettiin tämä jättää toteuttamatta.

Bluetooth-yhteyden luonnin aloittava koodi pitäisi sijoittaa omaan luokkaansa Game-luokan sijaan koodin luettavuuden parantamiseksi.

BtManager-luokka olisi voitu toteuttaa palveluna jotta tähänkin puoleen Android-kehitystä oltaisiin voitu tutustua.

Esimerkki ohjelma kaatuu kun siitä pyrkii poistumaan sen jälkeen kun Bluetooth-yhteys on muodostettu. Tämä johtunee juurikin Bluetooth-yhteydestä jonka sulkemista ei ole luultavasti toteutettu täysin oikein. Ajan puutteen vuoksi tätäkään ei kuitenkaan ehditty korjaamaan.

Ajan puutteen vuoksi pelin gyroskooppi-ominaisuudet päätettiin myös jättää toteuttamatta mikä johtui osittain myös testilaitteiden huonosta saatavuudesta.

Peli olisi hyvä optimoida toimimaan eri kokoisilla näytöillä varustetuilla latteilla niin että esimerkiksi alkuvalikon painonapit ja peliobjektit skaalautuisivat isommiksi tai pienemmiksi näytön koon mukaan.

Valikko äänissä on epäsynkkaa äänentoiston ja napin painon välillä.

Esimerkki applikaatiossa pitäisi ottaa huomioon Android-applikaation elinkaaren esittämät rajoitteet ja lisätä tarvittavat onPause()-, onResume()-, onStop()- ja onStart()-metodit, jotka esimerkiksi hoitaisivat muistin loppumisesta tai puhelun tulosta johtuvat pysähdykset.

Ufon liikuttamisessa on takkuilua mikä johtunee siitä että ufoa päivitetään applikaation pääsäikeessä kaikkien muiden laskutoimitusten ohessa. Ufon päivitys pitäisi siirtää tulevaisuudessa omaan säikeeseensä jotta sen liikuttamisesta tulisi sulavampaa.

Räjähdysanimaatioiden eri kuvat olisi pitänyt leikata paremmin irti taustastaan ja sinisen taustan pitäisi olla läpinäkyvä. Nämä toiminnot kuuluvat kuitenkin enemmän kuvankäsittelyn puoleen ja jätettiin sen takia toteuttamatta.

7. LOPPUYHTEENVETO

7.1 Android vs. iOS

Koska tämän opinnäytetyön raportin kirjoittajalla on aikaisempaa kokemusta Applen iPhonelle ohjelmoinnista keskittyy tämä kappale vertailemaan näitä kahta alustaa keskenään. Tämä on soveliasta siinäkin mielessä että Android- ja iPhone ovat tämän hetken markkinoiden suosituimmat mobiilialustat.

Iphoneen verrattuna Android-applikaatioiden ohjelmoiminen tuntui aluksi vaikeammalta ja näkyvien tulosten saanti niiden ideoinnista laitteiden näytölle saakka kesti kauemmin. Yksi syy tähän on Applen Interface Builder-ohjelma joka antaa kehittäjien hoitaa monien käyttöliittymien elementtien sijoittelun ja toiminnallisuuden määrittämisen graafisesti ennen kuin heidän tarvitsee kirjoittaa varsinaista koodia ollenkaan.

Toinen syy liittyy Android applikaatioiden rakenteeseen joka koostuu useasta erillisestä mutta toistensa kanssa yhteistyötä tekevistä komponenteista joiden välisestä kommunikoinnista huolehtiminen saattaa olla aluksi sekavalta tuntuvaa.

Android antaa kuitenkin kehittäjille vapaat kädet järjestelmän eri resurssien käyttöön mikä tarkoittaa että applikaatioiden on mahdollista olla monipuolisempia Applen vastaaviin, joiden toiminnallisuutta rajoittavat Applen kolmansien osapuolien kehittäjille osoitetut rajoitukset laitteiden eri ominaisuuksien käyttöönottoon ja muokkaamiseen.

Javaa ennestään tunteville kehittäjille Androidiin siirtyminen on varmasti luontevaa sen pohjautuessa vahvasti kyseiseen kieleen. Harvinaisempaa Objective-C:tä käyttävät Applen laitteet lisäävät yhden esteen jonka laitteelle siirtyvät kehittäjät joutuvat ylittämään ennen kuin he pääsevät rakentamaan applikaatioitaan. Sinänsä uuden kielen oppimisen ei pitäisi olla ylivoimaisen vaikeaa kellekään asiantuntevalle ohjelmoijalle.

Lisäksi lisenssien saaminen Applelta applikaatioiden testaamiseen oikeiden laitteiden kanssa puhumattakaan niiden julkaisusta AppStoressa on hyvin vaikeaa ja aikaa vievää koska Apple haluaa erikseen hyväksyä jokaisen sen laitteille tehtävän applikaation. Androidilla vastaavia ongelmia ei ole.

Applen lisenssit myös maksavat(99$ tällä hetkellä peruslisenssistä) kun Android-kehitys on ilmaista.

7.2 Yleisiä huomioita kehityksestä

Haasteellisinta esimerkki applikaation tekemisessä oli järjestää applikaation eri komponenttien välinen yhteistyö ja kommunikointi parhaimmalla mahdollisella tavalla joka muuttui vaikeammaksi sitä mukaa kun ohjelman monimutkaisuus lisääntyi.

Androidin eri ominaisuudet ja vaatimukset ovat kuitenkin kattavasti dokumentoitu Android-kehittäjien kotisivuilla jossa tarjotaan myös monia hyviä esimerkkiohjelmia joiden avulla pääsee hyvin selville Android-kehityksen eri käytännöistä.

Bluetooth-yhteyden muodostus ja datan lähetys sen yli osoittautui sen sijaan helpommaksi kuin oli odotettu vaikka sen saaminen toimimaan itse pelin kanssa ei onnistunutkaan. Silti Bluetoothin perusperiaatteiden ymmärtäminen ja implementoiminen oli helppoa ja nopeaa. Vaikeuksia oli vain viestien muuttamisessa bitti-muotoon ja takaisin mutta siitäkin selvittiin internetin avustuksella.

Näin jälkeenpäin ajateltuna kokonaisen pelin tekeminen oli liian kunnianhimoinen tavoite ja sen sijaan esimerkki applikaatio olisi voitu kasata joukosta pienempiä teknologiademoja jotka olisivat esitelleet näitä samoja ominaisuuksia kuin pelinkin on tarkoitus.

7.3 Loppumietteet

Androidille ohjelmoimisen aloittaminen on varmasti helppoa kaikille ohjelmoinnin perustaidot omaaville kehittäjille. Vaikeuksia tuottaa lähinnä edellämainittu Androidin tapa koostaa applikaatiot monista erillisistä mutta toistensa kanssa yhteistyötä tekevistä komponenteista joka vaatii kehittäjältä enemmän tarkkaavaisuutta applikaatioiden arkkitehtuuria mietittäessä. Kuitenkin kun perusperiaatteet selviävät on applikaatioiden kehittäminen Androidille yhtä helppoa kuin Java- ja mobiiliohjelmointi yleensä voi olla.

LÄHTEET

Wikipedia. Luettu 18.3.2012
http://fi.wikipedia.org/wiki/Ohjelmointirajapinta

Wikipedia. Luettu 18.3.2012
http://en.wikipedia.org/wiki/Application_software

Wikipedia. Luettu 18.3.2012
http://fi.wikipedia.org/wiki/Graafinen_k%C3%A4ytt%C3%B6liittym%C3%A4

Wikipedia. Luettu 18.3.2012
http://en.wikipedia.org/wiki/Java_(programming_language)

Wikipedia. Luettu 18.3.2012
http://en.wikipedia.org/wiki/Mobile_device

Wikipedia. Luettu 18.3.2012
http://fi.wikipedia.org/wiki/Keskusmuisti

Wikipedia. Luettu 18.3.2012
http://fi.wikipedia.org/wiki/K%C3%A4ytt%C3%B6liittym%C3%A4

Wikipedia. Luettu 18.3.2012
http://en.wikipedia.org/wiki/UUID

Wikipedia. Luettu 18.3.2012
http://fi.wikipedia.org/wiki/URI

Wikipedia. Luettu 18.3.2012
http://fi.wikipedia.org/wiki/XML

Wikipedia. Luettu 10.1.2012
http://fi.wikipedia.org/wiki/Android

Developer.android.com: What is Android? 2011. Luettu 17.12.2011
http://developer.android.com/guide/basics/what-is-android.html

Developer.android.com: Dev Guide. 2011. Luettu 10.1.2012.
http://developer.android.com/guide/index.html

Denieks, Z., Dornin, L., Meike, G.B. & Nakamura M. 2011. Programming Android. Sebastopol: O'Reilly Media Inc.

Lee W. 2011. Beginning Android Application Development, In Full Color. Indianapolis Indiana: Wiley Publishing Inc.

Ableson, W.F., Sen, R., King, C. & Ortiz, C.E. 2012. Android in Action, Third Edition. Shelter Island, NY: Manning Publications Co.

Huang, A.S & Rudolph, L. 2007. Bluetooth Essentials for Programmers. New York, USA : Cambridge University Press.

LIITTEET

1. Johdanto

1.1 Tarkoitus ja kattavuus

Työn tarkoituksena on tutustua Android- ja tablet-ympäristöihin ja näissä työskentelyyn yhden esimerkki ohjelman kautta. Ohjelmassa on tarkoitus käyttää yleisempiä androidin ja tablettien ominaisuuksia kuten kosketusnäyttöä, valikkoja, bluetooth-kommunikointia ja muita jotka määritellään tarkemmin muaalla tässä dokumentissa.

1.2 Tuote ja ympäristö

Esimerkki ohjelma on eräänlainen pelimuotoinen teknologiademo jota pelataan kahdella android-pohjaisella taulutietokoneella.

1.3 Määritelmät, termit ja lyhenteet

Android

Googlen kehittämä mobiililaitteille tarkoitettu ohjelmistopino joka sisältää käyttöjärjestelmän, väliohjelmistoja ja perusohjelmistoja.

Bluetooth

Protokolla laitteiden lähietäisyydellä tapahtuvaan langattomaan kommunikointiin.

Honeycomb

Erityisesti tableteille kehitetty versio Androidista.

2. Yleiskuvaus

2.1 Ympäristö

Ohjelma toimii android 3.0 eli Honeycomb ja sitä suuremmissa android-pohjaisissa taulutietokoneissa.

(Jatkuu)

2.2 Toiminta

Peliä pelataan kahdella tablet-laitteella ja pelin tarkoituksena on toisen pelaajan osalta väistellä ufollaan toisen pelaajan ampumia ammuksia. Jos ”tykkiä” ohjastava pelaaja osuu viisi kertaa ufoon voittaa silloin tämä pelaaja pelin ja jos taas ufoa pelaava pystyy väistelemään näitä ammuksia tietyn ajan tulee hän vastaavasti voittamaan.

Pelin alkuvalikossa käyttäjät valitsevat kumman tabletti toimii serverinä ja kumman clienttinä. Tämä vaikuttaa myös siihen kumpaa roolia pelaajat pelaavat.

2.3 Käyttäjät

Peli on kahden ihmisen, kahdella taulutietokoneella pelattava peli. Toista pelaajaa ei voi korvata tietokoneella.

2.4 Yleiset rajoitteet

Peli toteutetaan ensin kahdessa erillisessä osassa niin että toinen tabletti saa tykin puolen ja toinen ufon puolen ilman että niissä olisi mahdollisuutta valita kummalla kyseisellä tabletilla pelaa. Myöhemmin nämä kaksi puolta yhdistetään sitten yhdeksi kokonaiseksi ohjelmaksi jotka sisältävät molemmat puolet pelistä. Tämä tehdään jotta turvattaisiin ainakin ohjelmiston perustoiminnallisuuden valmistuminen ennen määrä ajan loppua.

Lisäksi, samasta syystä, pelin ravistus- ja aluksen peilikuvaominaisuudet määritellään optionaalisiksi jotka lisätään ohjelmaan jos aikaa riittää.

2.5 Oletukset ja riippuvuudet

Peli on tarkoitus pelattavaksi mahdollisimman isoruutuisella android-pohjaisella tabletilla.

Tabletti voi olla joko 3.0-versioinen tai sitten uudempi ja sen tarvitsee omata bluetooth-ominaisuudet.

3. Toiminnot

3.1 Kosketusnäyttö

Pelaajat pelaavat peliä kosketusnäytön avulla niin että toinen pelaaja ohjastaa alustaan painamalla sitä sormellaan ja raahaten sitä ympäri näyttöään ja toinen pelaaja yrittää osua siihen omalla sormellaan omalla näytöllään.

3.2 Bluetooth

Tablettien välinen kommunikointi hoidetaan bluetoothilla. Toinen laitteista toimii silloin serverinä ja toinen clientinä niin että serveri-laitteella pelataan ufolla ja clientillä tykillä. Ufon sijainti sitä pelaavan laitteilla lähetetään toiselle tabletille ruudulle piirrettäväksi samaten kuin tykin ammusten räjähdyksetkin niin että molemmilla pelaajilla on samat kokoajan näkymät.

3.3 Grafiikka

Ufoa edustaa yksinkertainen netistä poimittu bittikartta ja tykkiä edustaa tykin ammusten ilmaan jättämät bittikartta räjähdykset. Tykin osuessa alukseen alus käy läpi pienen animaation jossa sen nähdään räjähtävän.

Pelialueen taustana toimii myöskin netistä löydetty taivasta esittävä bittikartta.

3.4 Ajastin

Pelissä on ajastin joka laskee aikaa alaspäin minuutista ja jollei ufoa ole siihen mennessä ammuttu alas voittaa ufoa-pelaava pelaaja.

3.5 Musiikki ja äänet

(Tämä ominaisuus on optionaalinen ja se toteutetaan jos on aikaa jäljellä)

Taustalla soi musiikkiraita ja tykin räjähdyksillä ilmassa ja itse aluksen räjähdyksillä on omat ääniefektinsä.

3.6 Valikot

Pelissä on alkuvalikko josta käyttäjä saa valita kumpi hänen tablettinsa on, serveri vai clienttti, näin ollen valiten myös pelaako hän pelissä tykillä vai ufolla. Valikkojen kielenä on englanti.

3.7 Gyroskooppi

(Tämä ominaisuus on optionaalinen ja se toteutetaan jos on aikaa jäljellä)

Tykkiä pelaava pelaaja saa kolme mahdollisuutta ravistella näyttöä 10 sekunnin välein niin että laitteen liikkuminen vaikuttaa ufon liikkeisiin ja mahdollisesti näin herpaannuttaa kyseisen pelaajan otteen aluksestaan.

3.8 Aluksen peilikuva ominaisuus

Tykin gyroskooppi ominaisuutta vastaavasti alusta ohjastavalle pelaajalle annetaan kolme mahdollisuutta luoda aluksestaan peilikuva-alus koskettamalla näyttöä jollain toisista sormistaan silloin kun yksi hänen sormistaan on painettuna ufoon. Nimensä mukaisesti pei

likuva alus peilaa oikean aluksen liikkeitä niin että aluksen mennessä vasemmalle peilikuva liikkuu oikealle jne. hämäten näin tykkiä joka ei (toivottavasti) tiedä kumpi on oikea alus.

3.9 Ufo

Ufo on 2D-bittikartalla toteutettu ja sillä on viisi yksikköä terveyttä minkä jälkeen se on ammuttu alas.

3.10 Tykki

Tykkinä toimii käyttäjän sormi. Kohtaan mihin pelaaja koskettaa näyttöä ilmestyy hetkeksi jonkinlainen savupilvi joka ilmoittaa ohi menneestä ammuksesta ja pelaajan osuessa ufoon ufossa tapahtuu pieni tai iso räjähdys riippuen menettääkö ufo vain terveyttä vai onko se ammuttu alas.

4. Ulkoiset liittymät

5.1 Laitteistoliittymät

Ulkoisia laitteistoliittymiä ei ole.

5.2 Ohjelmistoliittymät

Ulkoisia ohjelmistoliittymiä ei ole.

5.3 Tietoliikenneliittymät

Kommunikointi hoidetaan tablettien sisäisellä bluetooth-protokollalla.

5. Muut ominaisuudet

5.1 Suorituskyky ja vasteajat

Peli on reealiaikainen ja koska siinä on tarkoitus osua suhteellisen pieneen ja nopeasti liikkuvaan maaliin
jonka on myös pystyttävä väistelemään nopeasti ja joiden molempien liikkeet ja toiminnot on mallinnettava yhtäaikaisesti kahdelle eri ruudulle täytyy vasteaikojen olla mahdollisimman pieniä.

5.2 Saavutettavuus ("käytettävyys"), toipuminen, turvallisuus, suojaukset

Peli ei saa kaatua kesken pelaamisen ja näytön päivitykset täytyy tapahtua yhtäaikaisesti molemmissa tableteissa.

Peli ei voi alkaa ennenkuin tabletit ovat saaneet yhteyden toisiinsa.

5.3 Ylläpidettävyys

Ohjelmalla ei ole tule olemaan ylläpitoa sen valmistumisen jälkeen.

5.4 Siirrettävyys ja yhteensopivuus

Peli on tarkoitus pelattavaksi tablet-laitteilla joten sen voi portata mihin tahansa eri valmistajien eri pohjaisille laitteille mutta sitä ei saisi pelata normaaleilla tai älypuhelimilla näiden kuvaruutujen pienuuden takia.

6. Suunnittelurajoitteet

6.1 Standardit

Ohjelmiston teossa pyritään noudattamaan android-alustan yleisiä suunnittelu periaatteita.

6.2 Laitteistorajoitteet

Peli rajoitetaan android-pohjaisilla tableteilla toimivaksi.

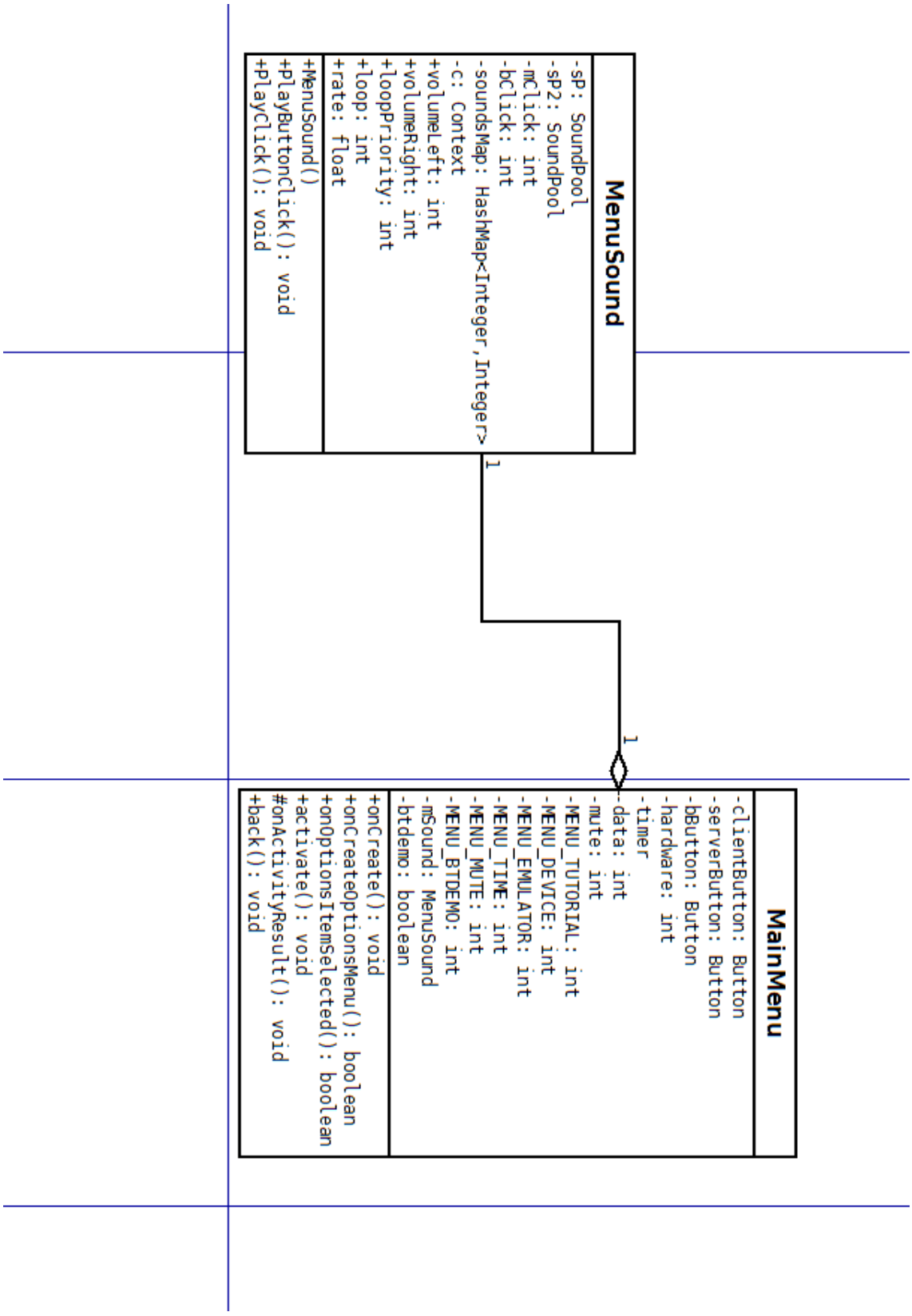
6.3 Ohjelmistorajoitteet

Tableteissa on oltava bluetooth-valmiudet.

6.4 Muut rajoitteet

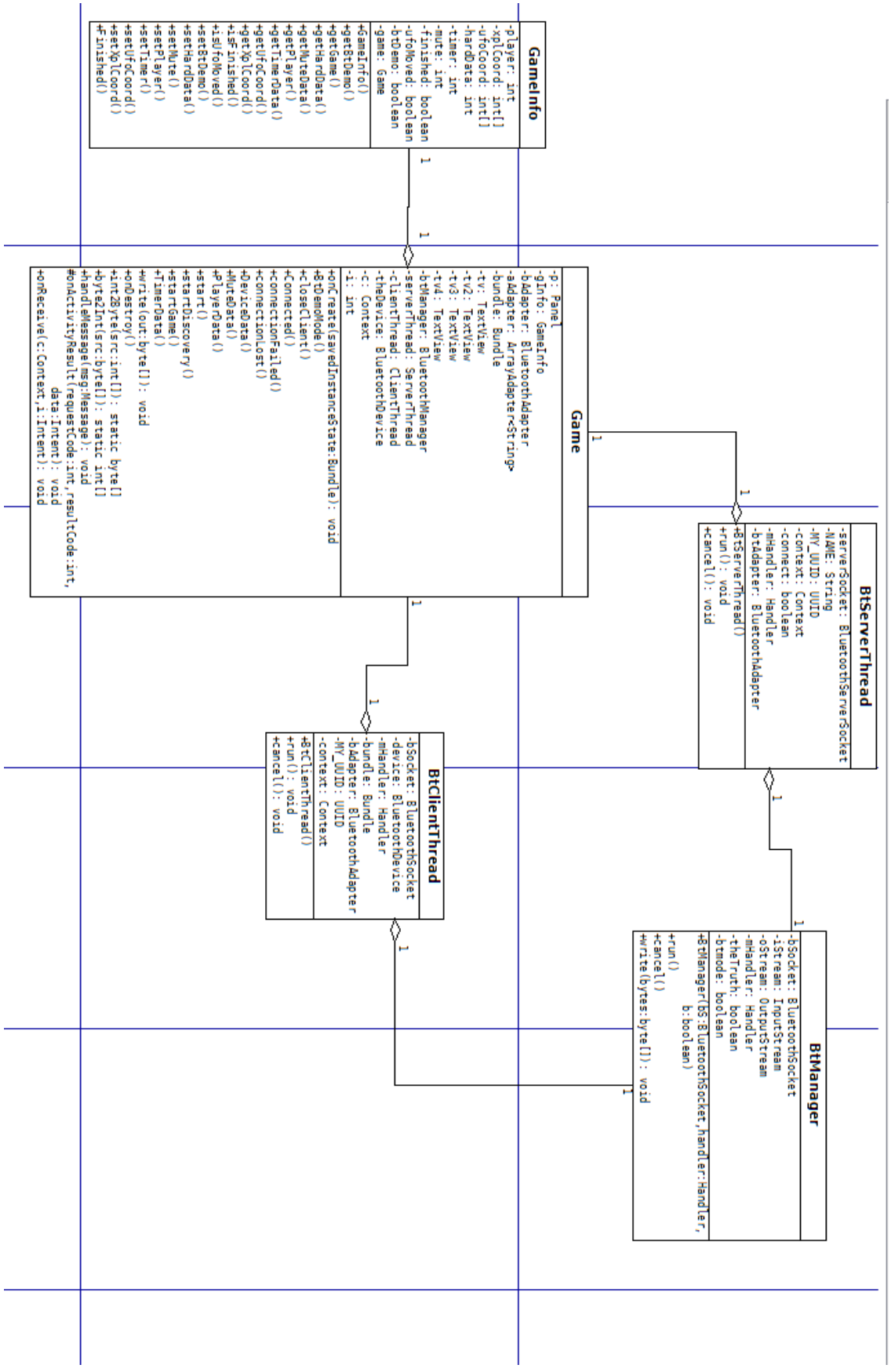
Muita rajoitteita ei ole.

Liite 2: Esimerkki ohjelman luokkakaaviot Liite 2:1(5)

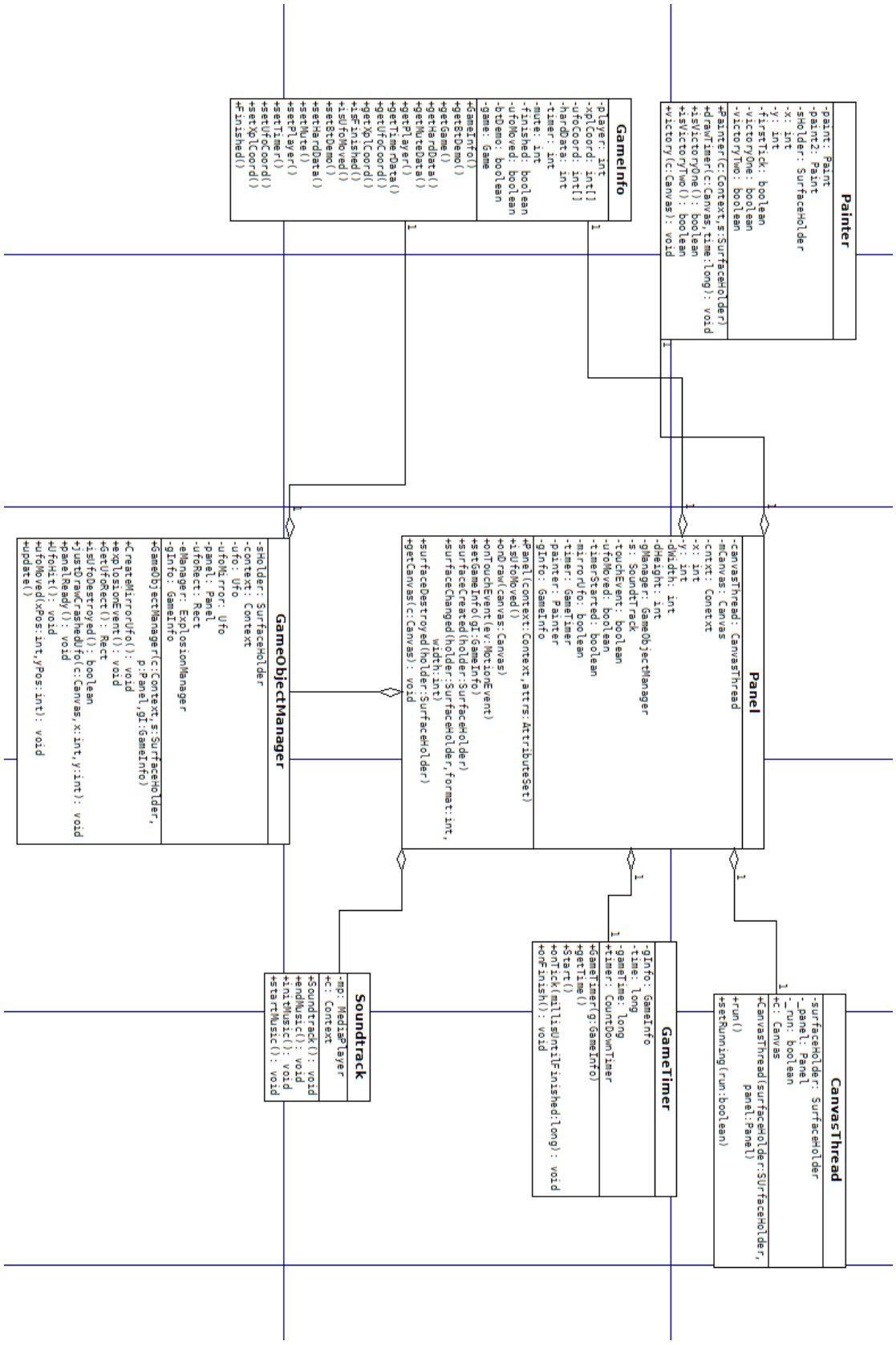


(Jatkuu)

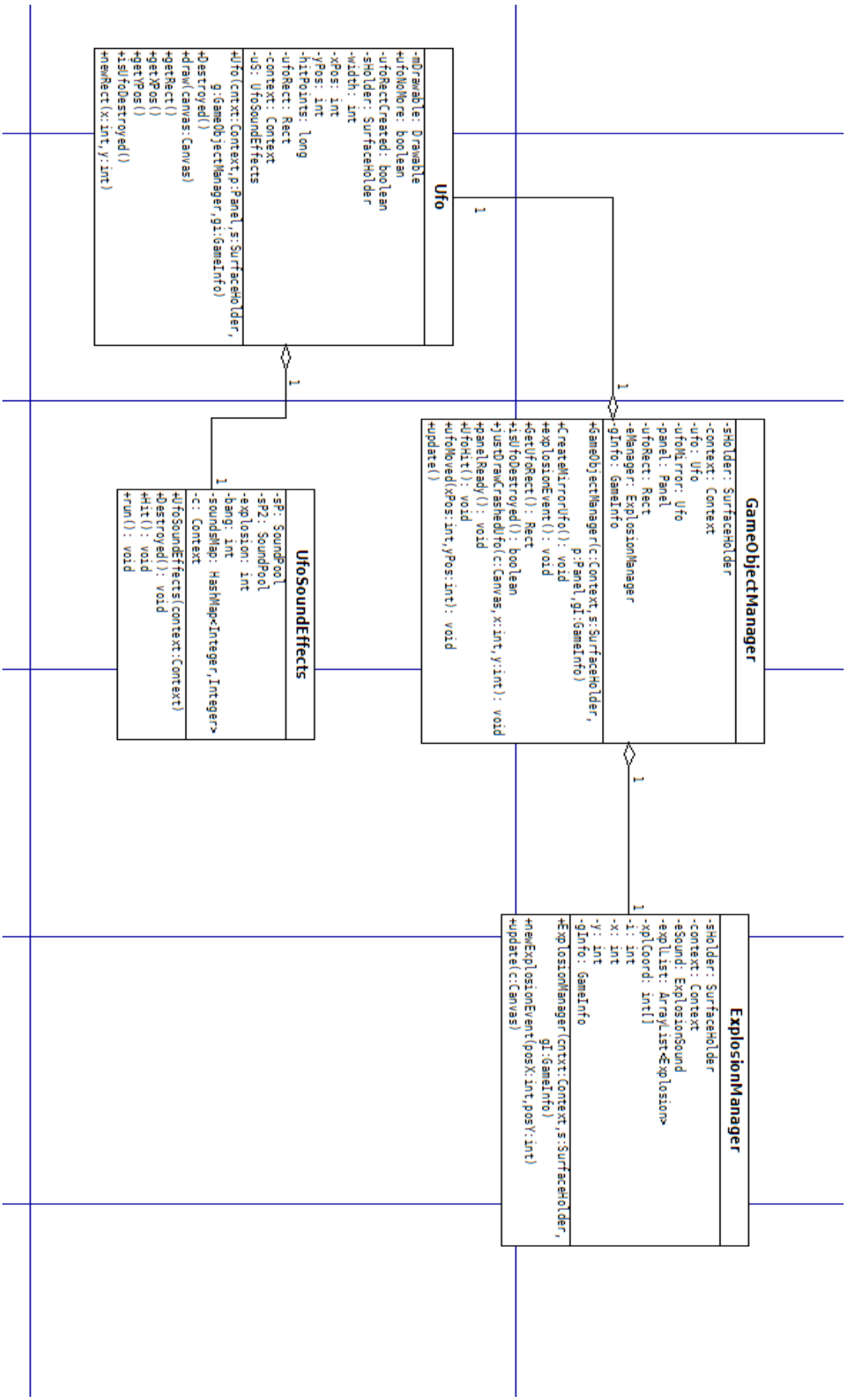
Liite 2:2(5)


GameInfo
-player: int
-xplCoord: int[]
-ufoCoord: int[]
-hardData: int
-timer: int
-mute: int
-finished: boolean
-ufoMoved: boolean
-btDemo: boolean
-game: Game
+GameInfo()
+getBtDemo()
+getGame()
+getHardData()
+getMuteData()
+getPlayer()
+getTimerData()
+getUfoCoord()
+getXplCoord()
+isFinished()
+isUfoMoved()
+setBtDemo()
+setHardData()
+setMute()
+setPlayer()
+setTimer()
+setUfoCoord()
+setXplCoord()
+finished()
Game
-p: Panel
-gInfo: GameInfo
-bAdapter: BluetoothAdapter
-aAdapter: ArrayAdapter<String>
-bundle: Bundle
-tv: TextView
-tv2: TextView
-tv3: TextView
-tv4: TextView
-btManager: BluetoothManager
-serverThread: ServerThread
-clientThread: ClientThread
-theDevice: BluetoothDevice
-c: Context
-i: int
+onCreate(savedInstanceState:Bundle): void
+BtDemoMode()
+closeClient()
+Connected()
+connectionFailed()
+connectionLost()
+DeviceData()
+MuteData()
+PlayerData()
+start()
+startDiscovery()
+startGame()
+TimerData()
+write(out:byte[]): void
+onDestroy()
+int2Bytes(src:int[]): static byte[]
+byte2Int(src:byte[]): static int[]
+handleMessage(msg:Message): void
#onActivityResult(requestCode:int,resultCode:int,
data:Intent): void
+onReceive(c:Context,i:Intent): void
BtServerThread
-serverSocket: BluetoothServerSocket
-NAME: String
-MY_UUID: UUID
-context: Context
-connect: boolean
-mHandler: Handler
-btadapter: BluetoothAdapter
+BtServerThread()
+run(): void
+cancel(): void
BtClientThread
-bSocket: BluetoothSocket
-device: BluetoothDevice
-mHandler: Handler
-bundle: Bundle
-bAdapter: BluetoothAdapter
-MY_UUID: UUID
-context: Context
+BtClientThread()
+run(): void
+cancel(): void
BtManager
-bSocket: BluetoothSocket
-iStream: InputStream
-oStream: OutputStream
-mHandler: Handler
-theTruth: boolean
-btmode: boolean
+BtManager(bS:BluetoothSocket,handler:Handler,
b:boolean)
+run()
+cancel()
+write(bytes:byte[]): void
1

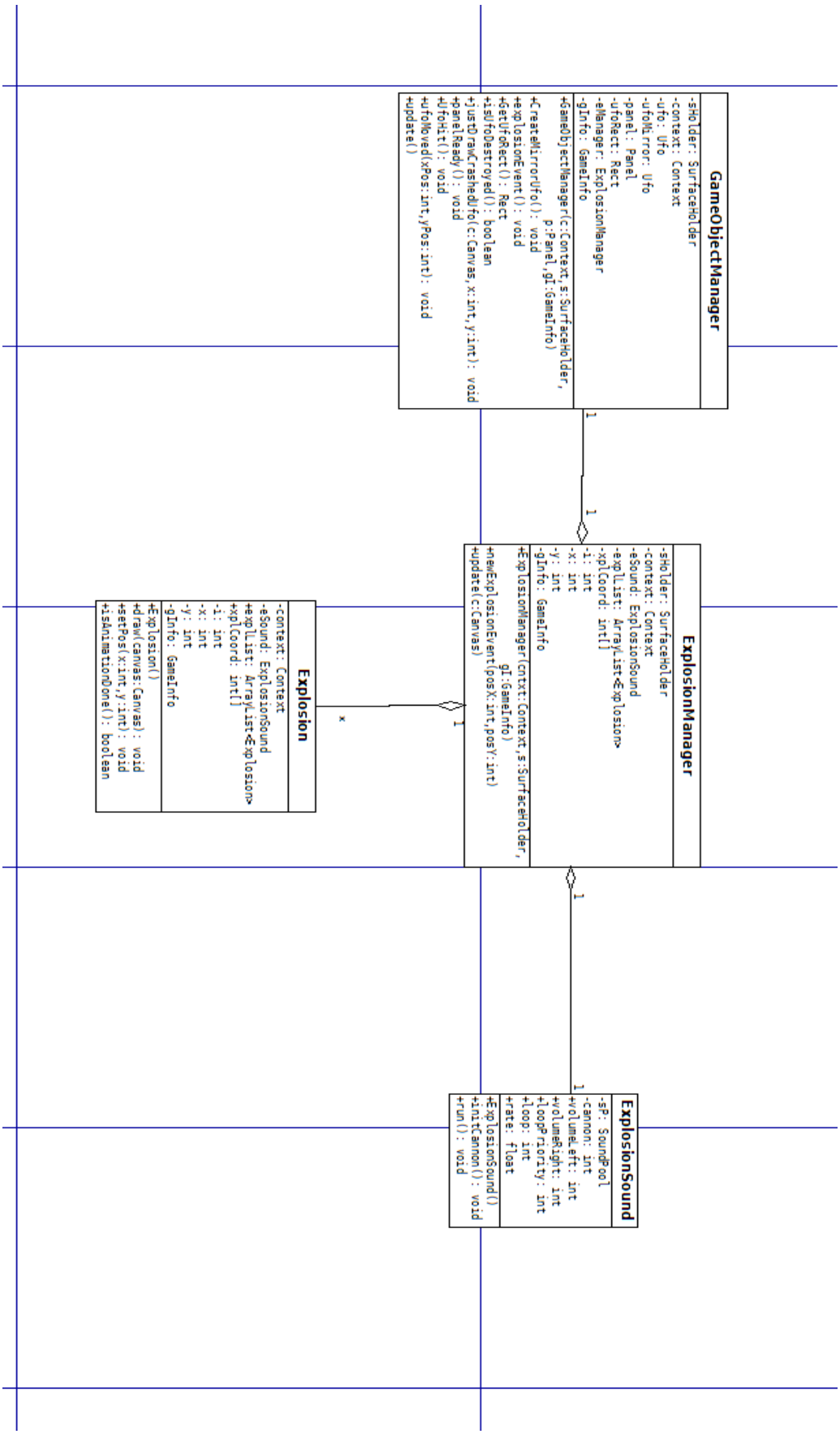
Liite 2:3(5)


Painter
-paint: Paint
-paint2: Paint
-sholder: SurfaceHolder
-x: int
-y: int
-firstTick: boolean
-victoryOne: boolean
-victoryTwo: boolean
+Painter(c:Context,s:SurfaceHolder)
+drawTimer(c:Canvas,time:long): void
+isVictoryOne(): boolean
+isVictoryTwo(): boolean
+victory(c:Canvas): void
GameInfo
-player: int
-xplCoord: int[]
-ufoCoord: int[]
-hardData: int
-timer: int
-mute: int
-finished: boolean
-ufoMoved: boolean
-btDemo: boolean
-game: Game
+GameInfo()
+getBtDemo()
+getGame()
+getHardData()
+getMuteData()
+getPlayer()
+getTimeData()
+getUfoCoord()
+getXplCoord()
+isFinished()
+isUfoMoved()
+setBtDemo()
+setHardData()
+setMute()
+setPlayer()
+setTime()
+setUfoCoord()
+setXplCoord()
+Finished()
Panel
-canvasThread: CanvasThread
-mCanvas: Canvas
-cntxt: Context
-x: int
-y: int
-dWidth: int
-dHeight: int
-gManager: GameObjectManager
-s: Soundtrack
-touchEvent: boolean
-ufoMoved: boolean
-timerStarted: boolean
-mirrorUfo: boolean
-timer: GameTimer
-painter: Painter
-gInfo: GameInfo
+Panel(context:Context,attrs:AttributeSet)
+isUfoMoved()
+onDraw(canvas:Canvas)
+onTouchEvent(ev:MotionEvent)
+setGameInfo(gI:GameInfo)
+surfaceCreated(holder:SurfaceHolder)
+surfaceChanged(holder:SurfaceHolder,format:int,
width:int)
+surfaceDestroyed(holder:SurfaceHolder)
+getCanvas(c:Canvas): void
CanvasThread
-surfaceHolder: SurfaceHolder
-_panel: Panel
-_run: boolean
-c: Canvas
+CanvasThread(surfaceHolder:SurfaceHolder,
panel:Panel)
+run()
+setRunning(run:boolean)
GameObjectManager
-sHolder: SurfaceHolder
-context: Context
-ufo: Ufo
-ufoMirror: Ufo
-panel: Panel
-eManager: ExplosionManager
-gInfo: GameInfo
+GameObjectManager(c:Context,s:SurfaceHolder,
p:Panel,gI:GameInfo)
+CreateMirrorUfo(): void
+explosionEvent(): void
+GetUfoRect(): Rect
+isUfoDestroyed(): boolean
+JustDrawCrashedUfo(c:Canvas,x:int,y:int): void
+panelReady(): void
+UfoHit(): void
+ufoMoved(xPos:int,yPos:int): void
+update()
GameTimer
-gInfo: GameInfo
-time: long
-gameTime: long
+timer: CountDownTimer
+GameTimer(g:GameInfo)
+getTime()
+Start()
+onTick(millisUntilFinished:long): void
+onFinish(): void
Soundtrack
-mp: MediaPlayer
+c: Context
+Soundtrack(): void
+endMusic(): void
+initMusic(): void
+startMusic(): void

Liite 2:4(5)


Ufo
-mDrawable: Drawable
+ufoNoMore: boolean
-ufoRectCreated: boolean
-sHolder: SurfaceHolder
-width: int
-xPos: int
-yPos: int
-hitPoints: long
-ufoRect: Rect
-context: Context
-uS: UfoSoundEffects
+Ufo(cntxt:Context,p:Panel,s:SurfaceHolder,
g:GameObjectManager,gi:GameInfo)
+Destroyed()
+draw(canvas:Canvas)
+getRect()
+getXPos()
+getYPos()
+isUfoDestroyed()
+newRect(x:int,y:int)
GameObjectManager
-sHolder: SurfaceHolder
-context: Context
-ufo: Ufo
-ufoMirror: Ufo
-panel: Panel
-ufoRect: Rect
-eManager: ExplosionManager
-gInfo: GameInfo
+GameObjectManager(c:Context,s:SurfaceHolder,
p:Panel,gI:GameInfo)
+CreateMirrorUfo(): void
+explosionEvent(): void
+GetUfoRect(): Rect
+isUfoDestroyed(): boolean
+JustDrawCrashedUfo(c:Canvas,x:int,y:int): void
+panelReady(): void
+UfoHit(): void
+ufoMoved(xPos:int,yPos:int): void
+update()
UfoSoundEffects
-sP: SoundPool
-sP2: SoundPool
-explosion: int
-bang: int
-soundsMap: HashMap<Integer,Integer>
-c: Context
+UfoSoundEffects(context:Context)
+Destroyed(): void
+Hit(): void
+run(): void
ExplosionManager
-sHolder: SurfaceHolder
-context: Context
-eSound: ExplosionSound
-explList: ArrayList<Explosion>
-xplCoord: int[]
-i: int
-x: int
-y: int
-gInfo: GameInfo
+ExplosionManager(cntxt:Context,s:SurfaceHolder,
gI:GameInfo)
+newExplosionEvent(posX:int,posY:int)
+update(c:Canvas)
1
1
1
1
1
1
1
1

Liite 2:5(5)


GameObjectManager
-sHolder: SurfaceHolder
-context: Context
-ufo: Ufo
-ufoMirror: Ufo
-panel: Panel
-ufoRect: Rect
-eManager: ExplosionManager
-gInfo: GameInfo
+GameObjectManager(c:Context,s:SurfaceHolder,
p:Panel,gI:GameInfo)
+createMirrorUfo(): void
+explosionEvent(): void
+getUfoRect(): Rect
+isUfoDestroyed(): boolean
+justDrawCrashedUfo(c:Canvas,x:int,y:int): void
+panelReady(): void
+ufoHit(): void
+ufoMoved(xPos:int,yPos:int): void
+update()
1
1
ExplosionManager
-sHolder: SurfaceHolder
-context: Context
-eSound: ExplosionSound
-expList: ArrayList<Explosion>
-xplCoord: int[]
-i: int
-x: int
-y: int
-gInfo: GameInfo
+ExplosionManager(context:Context,s:SurfaceHolder,
gI:GameInfo)
+newExplosionEvent(posX:int,posY:int)
+update(c:Canvas)
1
*
Explosion
-context: Context
-eSound: ExplosionSound
+expList: ArrayList<Explosion>
+xplCoord: int[]
-i: int
-x: int
-y: int
-gInfo: GameInfo
+Explosion()
+draw(canvas:Canvas): void
+setPos(x:int,y:int): void
+isAnimationDone(): boolean
1
1
ExplosionSound
-sP: SoundPool
-cannon: int
+volumeLeft: int
+volumeRight: int
+loopPriority: int
+loop: int
+rate: float
+ExplosionSound()
+initCannon(): void
+run(): void

Liite 3: MainMenu-aktiviteetti Liite 3:1 (8)

```
//Tämä on applikaation alkuvalikko johon käyttäjä tuodaan ensimmäisenä 
//applikaation käynnistyessä. Näkymässä käyttäjälle tarjotaan kaksi, 
kosketukseen reagoivaa nappia joita painamalla hän voi valita toimiiko 
käytössä oleva laite pelin serverinä vai clienttinä.
//Käyttäjällä on myös mahdollisuus avata applikaation  optionsmenu 
jossa hän voi valita suoritetaanko applikaatio emulaattori- vai 
devicetilassa, käynnistetäänkö pelin sijaan pelkkä Bluetooth-yhteyttä 
demoava moodi, onko äänentoisto käytössä vai ei, määrittää peli 
ajan(10-100s) ja lukea pelin tutoriaalin

public class MainMenu  extends Activity{


    //Päävalikossa olevat napit
    private Button clientButton,serverButton;
    //SeekBar-näkymään tuleva painonappi, jolla palataan takaisin 
päänäkymään
    private Button bButton;
    //Muuttujat jotka kertovat päävalikossa tehdyistä valinnoista
    //ja jotka tullaan välittämään Game-aktiviteetille
    private int hardware,timer,data,mute;
    private boolean emulator,btdemo;
    private SeekBar sBar;
    //Määritetään tunnukset eri optionsMenu valinnoille
         private final int MENU_TUTORIAL=0;
         private final int MENU_DEVICE=1;
         private final int MENU_EMULATOR=2;
         private final int MENU_TIME=3;
         private final int MENU_MUTE=5;
         private final int MENU_BTDEMO=6;
         private int progress;
         private MenuSound mSound;
         @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.mainmenu);
        //Peliä pelataan oletusarvoisesti oikeilla laitteilla joten
          hardware-muuttuja
        //alustetaan 2:lla
        hardware=2;
        emulator=false;
           //Luodaan uusi MenuSound-olio joka hoitaa päänäkymään
             // liittyvien äänitehosteiden toiston
        mSound=new MenuSound(this);
        //Peliaika on oletusarvoisesti 10 sekunttia
        timer=10;
        data=1;
        mute=1;
        progress=0;
        btdemo=false;
```

(Jatkuu)

Liite 3:2 (8)

```
        //Sijoitetaan nappien määritykset XML-layoutista niiden id:en
            //avulla Java-muuttujiin
          serverButton=(Button) findViewById(R.id.serverbutton3);
          clientButton=(Button) findViewById(R.id.clientbutton3);

          //Rekisteröidään onTouchListener serverButtonille
            //kosketustapahtumien kuuntelemista varten
          serverButton.setOnTouchListener(new OnTouchListener(){
            //Kosketustapahtuman tapahtuessa suoritetaan onTouch()-metodi
            @Override
               public boolean onTouch(View v, MotionEvent e) {
      if(v==serverButton && e.getAction()==MotionEvent.ACTION_UP){
    //Käyttäjän päästäessä irti napista määritetään datan arvoksi 1
   //joka kertoo että pelaaja on valinnut serverin roolin
                      data=1;
            //Soitetaan asianmukainen ääniefekti
                      mSound.PlayButtonClick();

//Siirrytään activate funktioon, joka tulee käynnistämään itse pelin
                     activate();
                     return true;
                     }
                     else
                     return false;
               }
          });

          //Tehdään sama clientButtonille
          clientButton.setOnTouchListener(new OnTouchListener(){
            @Override
               public boolean onTouch(View v, MotionEvent e) {
         if(v==clientButton && e.getAction()==MotionEvent.ACTION_UP){
                              data=2;
                              mSound.PlayButtonClick();
                             activate();
                     return true;
                     }


                     else
                     return false;
               }
          });


}
```

Liite 3:3 (8)

```
//Tämä luo applikaation optionsmenun
@Override
public boolean onCreateOptionsMenu(Menu menu){
          super.onCreateOptionsMenu(menu);
          //Lisätään valikkoon seuraavat vaihtoehdot
          //R.string.* kertoo mistä  valikkoon lisättävien valintojen
nimet löydetään
          menu.add(0,MENU_TUTORIAL,0,R.string.tutorial);
          menu.add(0,MENU_DEVICE,0,R.string.device);
          menu.add(0,MENU_EMULATOR,0,R.string.emulator);
          menu.add(0,MENU_TIME,0,R.string.shortgame);
          menu.add(0,MENU_MUTE,0,R.string.mute);
          menu.add(0,MENU_BTDEMO,0,R.string.btdemo);

          return true;
}

//Käyttäjän valitessa yhden optionsmenun valinnoista suoritetaan tamä
metodi
@Override
public boolean onOptionsItemSelected(MenuItem item){
          switch(item.getItemId()){
          //Jos tutorial on valittu
          case MENU_TUTORIAL:
                    //Soitetaan asiaankuuluva ääniefekti
                    mSound.PlayClick();


   //Alertdialogi rakennetaan Builder-luokan avulla
        Builder builder = new AlertDialog.Builder(this);
   //Dialogin otsikko
        builder.setTitle("Tutorial");

   //Itse dialogin viesti
        builder.setMessage("Choose which side you are going to play by
pressing" +" the Server or Client buttons on the mainmenu which rep-
resent the ufo- and cannon-sides of the game respectively. Also from
the optionsMenu " +"you can choose the Device option if you are play-
ing on real devices, which is the default choice, or Emulator option
if you are using emulator(Note: if you are playing on an emulator and
you choose the device-option " +"you will not be able to see the ufo
on the client-screen!). If you don't want to play the game but want to
see if the Bluetooth-connection is working choose the BtDemo-option.
From the menu you can also set the game time with Set Time-option as
well as mute the sound with the Mute-option.");

      //Määrittää positiivisen napin asetukset
        builder.setPositiveButton("Ok", new DialogInterface.OnClick-
Listener()
        {           @Override
    public void onClick(DialogInterface dialog, int id) {
          //Ok-nappia painettaessa tuhotaan dialogi
          dialog.cancel();
        }
        }
        );
    //Näyttää dialogin
        builder.show();
          return true;
```

Liite 3:4 (8)

```
          //Jos emulaattori valittu
          case MENU_EMULATOR:
                    mSound.PlayClick();
         //Ilmoitetaan käyttäjälle että Emulaattori-tila on valittu
Toastin avulla
                    //Toastit ovat lyhyitä viestinäkymiä jotka
ilmestyvät lyhyeksi aikaa ruudulle
                    //kertomaan jotakin käyttäjälle ja katoavat sen
jälkeen omatoimisesti
Toast toast2=Toast.makeText(getApplicationContext(),
"Emulator selected", 0);
                    toast2.show();
                    //sijoitetaan hardware-muuttujaan  1
                    hardware=1;

                    return true;
                    //Jos device on valittu
          case MENU_DEVICE:
                    mSound.PlayClick();

Toast toast3=Toast.makeText(getApplicationContext(),
"Device selected", 0);
                    toast3.show();
           //Sijoitetaan muuttujaan 2
                    hardware=2;
                    return true;
          //Jos Set Gametime-vaihtoehto on valittu
          case MENU_TIME:
                    mSound.PlayClick();

          //Vaihdataan näkymä mainmenu.xml:stä seekbar.xml:ään
          setContentView(R.layout.seekbar);
//Luodaan handleri timerbar-näkymään ja määritetään palkin maksimi
                //pituudeksi 100-yksikköä ja aloituspisteeksi 10.
              sBar=(SeekBar) findViewById(R.id.timerbar);
                sBar.setMax(100);
                sBar.setProgress(10);
                //Rekisteröidään palkki muutoksenkuuntelijalle
sBar.setOnSeekBarChangeListener(new OnSeekBarChangeListener() {
              //TextView joka tulee kertomaan mikä palkin arvo on
               TextView tv=(TextView) findViewById(R.id.text);


             @Override
             public void onStopTrackingTouch(SeekBar seekBar) {}


   //Tämä metodi suoritetaan kun palkkia aletaan liikuttamaan
             @Override
             public void onStartTrackingTouch(SeekBar seekBar) {
   //Sijoitetaan palkin tämän hetkinen sijainti progress-muuttujaan
                       progress=seekBar.getProgress();


             }
```

Liite 3:5 (8)

```
//Tätä metodia kutsutaan aina kun palkin tilanne muuttuu
@Override
public void onProgressChanged(SeekBar seekBar, int progress,
boolean fromUser) {

             tv.setText("Current playing time: "+ progress);
    //Jos progress-muuttujan arvo on suurempi kuin aloitusarvo
    //sijoitetaan uusi arvo timer-muuttujaan
                    if(progress>10){
                    timer=progress;
                    }
                    else timer=10;

            }
           });
      //Seekbar-näkymässä on yksi painonappi jota painamalla palataan
     //takaisin MainMenu-näkymään
               bButton=(Button) findViewById(R.id.backbutton);
               bButton.setOnTouchListener(new OnTouchListener(){


         @Override
 public boolean onTouch(View v, MotionEvent e) {
if(v==bButton && e.getAction()==MotionEvent.ACTION_UP){
         //Siirrytään back()-metodiin
                       back();
                     return true;
                           }
                           else
                           return false;
                                    }

               });



Toast toast4=Toast.makeText(getApplicationContext(),timer+" second
countdown set!", 0);
                    toast4.show();

                    return true;
         //Jos mute on valittu
         case MENU_MUTE:
                    mSound.PlayClick();

                    if(mute==1){
                    mute=2;
                    Toast toast6=Toast.make-
Text(getApplicationContext(),"Silent mode on", 0);
                    toast6.show();
                    return true;
                    }
                    else if(mute==2){
                              mute=1;
                    Toast toast6=Toast.make-
Text(getApplicationContext(),"Silent mode off", 0);
                    toast6.show();
                    return true;
                    }
                    return true;
```

Liite 3:6 (8)

```
		//Jos BtDemo valittu
		case MENU_BTDEMO:

				if(btdemo==false){
				btdemo=true;
				Toast toast7=Toast.make-
Text(getApplicationContext(),"Bluetooth demo mode set", 0);
				toast7.show();
				return true;

				}
				else if(btdemo==true){
					btdemo=false;
Toast toast7=Toast.makeText(getApplicationContext(),"Bluetooth demo
mode set", 0);
				toast7.show();
				return true;
				}
				return true;
		}


		return false;
}
//Tämä metodi tulee hoitamaan aktiviteettien vaihdon
		public void activate(){


//Luodaan uusi Game-aktiviteetille osoitettu intentti
		Intent newIntent=new In-
tent(this.getApplicationContext(),Game.class);
//Luodaan bundle jonka sisälle sijoitetaan informaatio alkuvalikossa
tehdyistä valinnoista
		Bundle bundle=new Bundle();
		bundle.putInt("data", data);
		bundle.putInt("hardware", hardware);
		bundle.putInt("timer", timer);
		bundle.putInt("mute", mute);
		bundle.putBoolean("btdemo", btdemo);

		//Annetaan bundle intentille
		newIntent.putExtras(bundle);

//Käynnistetään newIntentin osoittama aktiviteetti paluu arvolla
FLAG_ACTIVITY_REORDER_TO_FRONT. StartActivityForResult() varmistaa
että applikaatio palaa tähän näkymään kun peli on lopetettu

startActivityForResult(newIntent,Intent.FLAG_ACTIVITY_REORDER_TO_FRONT
);

		}


@Override
protected void onActivityResult(int requestCode, int resultCode, In-
tent data)
{
		super.onActivityResult(requestCode, resultCode, data);
}
```

Liite 3:7 (8)

```
//Tämä metodi luo MainMenu-näkymän toiminnallisuuden uudestaan kun
//käyttäjä palaa Seekbar-näkymästä
          public void back(){
          setContentView(R.layout.mainmenu);

    serverButton=(Button) findViewById(R.id.serverbutton3);
    clientButton=(Button) findViewById(R.id.clientbutton3);

    serverButton.setOnTouchListener(new OnTouchListener(){
          @Override
        public boolean onTouch(View v, MotionEvent e) {
                    if(v==serverButton && e.getAction()==Mo-
tionEvent.ACTION_UP){
                                 data=1;
                                 mSound.PlayButtonClick();

                                activate();
                    return true;
                    }

                    else
                    return false;
        }
    });


    clientButton.setOnTouchListener(new OnTouchListener(){
          @Override
        public boolean onTouch(View v, MotionEvent e) {
                    if(v==clientButton && e.getAction()==Mo-
tionEvent.ACTION_UP){
                                 data=2;
                                 mSound.PlayButtonClick();
                                activate();
                    return true;
                    }


                    else
                    return false;
        }
    });

}



}
```

MenuSound-luokka Liite 3:8 (8)

```
//Tämä luokka huolehtii päävalikossa kuuluvien ääniefektien toistosta
          public class MenuSound {

                    private SoundPool sP,sP2;
                    //Muuttujat molemmille ääniefekteille
                    private int mClick,bClick;
                    private HashMap<Integer,Integer> soundsMap;
                    private Context c;

          public MenuSound(Context cntxt){
                    //Applikaation konteksti
                    c=cntxt;
//Määritetään kaksi eri SoundPool-oliota sP-olio hoitaa OptionsMenuun
kuuluvat ääniefektit ja sP2-olio painonappien efektit
                     sP=new SoundPool(4, AudioManager.STREAM_MUSIC,10);
                     sP2=new SoundPool(4,AudioManager.STREAM_MUSIC,10);

                     //Ääniefektit sijoitetaan HashMappiin
                soundsMap=new HashMap<Integer,Integer>();
                soundsMap.put(mClick, sP.load(c,R.raw.sound4,1));
                soundsMap.put(bClick, sP2.load(c,R.raw.sound57,1));

          }

//Tämä metodi toistaa optionsmenun vaihtoehtoihin kuuluvan ääniefektin
          public void PlayClick(){

//VolumeLeft ja -right määrittävät vasemman ja oikein kanavan äänen-
voimakkuuden
                    int volumeLeft=10;
                     int volumeRight=10;
//LoopPriority määrittää toistettavan äänen prioriteetin joka on tässä
määritelty yhdeksi eli korkeimmaksi mahdolliseksi.
                     int loopPriority=1;
// loop määrittää toistetaanko ääniefekti useasti vai ei tässä tapauk-
sessa arvo on 0 eli ääni toistetaan vain kerran.
                     int loop=0;
//Rate määrittää toistetaanko ääniefekti sillä nopeudella kuin se on
alun perin luotu vai kenties nopeammin tai hitaammin. Arvon ollessa
yksi ääni toistetaan alkuperäisessä muodossaan
                     float rate=1;
/Ääniefekti toistetaan tällä käskyllä.
sP.play(soundsMap.get(mClick), volumeLeft, volumeRight, loopPriority,
loop, rate);
          }
//Tämä metodi toistaa painonappeihin kuuluvan ääniefektin
          public void PlayButtonClick(){

          int volumeLeft=10;
          int volumeRight=10;
          int loopPriority=1;
          int loop=0;
          float rate=1;
sP2.play(soundsMap.get(bClick), volumeLeft, volumeRight, loopPriority,
loop, rate);
          }
}
```

Liite 4: Bluetooth-komponentti Liite 4:1(7)

BtClientThread-luokka

```
//Tämä luokka huolehtii Bluetooth-yhteyden asiakaspuolen muodostamis-
esta

public class BtClientThread extends Thread {

          private final BluetoothSocket bSocket;
          private final BluetoothDevice device;
          private Handler mHandler;
          private Bundle bundle;
          private final BluetoothAdapter bAdapter;
          private  final UUID MY_UUID=UUID.fromString("4ba79700-15af-
11e1-be50-0800200c9a66");
          private Context context;

public BtClientThread(Context c,BluetoothDevice d, Handler handler){

                    BluetoothSocket tmp=null;
                    mHandler=handler;
          //Haetaan handleri laitteen BluetoothAdapteriin
                    bAdapter=BluetoothAdapter.getDefaultAdapter();
//Device edustaa Game-luokassa löydettyä Bluetooth-etälaitetta
                    device=d;
                    context=c;
                    try{
//Pyydetään BluetoothDevicea luomaan uusi ulospäin suuntautunut
Bluetooth-pistoke joka tullaan yhdistämääm MY_UUID:tä kantavaan
etälaitteeseen.
                              tmp=device.createRfcommSocketToServiceRe-
cord(MY_UUID);

                    }
                    catch(IOException e){

                    }
                    bSocket=tmp;
                    Log.d("cThread","RFCOMM CREATED!!!");
          }


          @Override
          public void run(){
//Lopetetaan laitteiden etsintä jottei se veisi resursseja
//pois itse yhteyden muodostukselta
                    bAdapter.cancelDiscovery();

                    Log.d("cThread","discovery canceled");

                    try {
                    Log.d("cThread","tries to connect");
            //Yritetään yhdistää pistoke etälaitteeseen
                              bSocket.connect();

                    } catch (IOException e) {
                              e.printStackTrace();
                              Log.d("cThread","closing...");
```

(Jatkuu)

Liite 4:2(7)

```
                    try {
                              Log.d("cThread","closing...");
//Jollei yhteyden muodostus onnistunut suljetaan pistoke
                              bSocket.close();
                    } catch (IOException e1) {

                              e1.printStackTrace();
                    }

//Pistokkeen tullessa onnistuneesti suljetuksi lähetetään pääsäikeelle
//CONNECTION_FAILED-viesti joka aloittaa yhteyden muodostuksen
uudestaan.

mHandler.obtainMessage(Game.CONNECTION_FAILED).sendToTarget();

                    return;
                    }

                    synchronized(this){
//Yhteyden tullessa muodostetuksi suljetaan asiakassäie lähettämällä
//pääsäikeelle CLOSE_THREAD-viesti...

          mHandler.obtainMessage(Game.CLOSE_CTHREAD).sendToTarget();
                    }
//... ja ilmoitetaan että asiakasyhteys on muodostettu CONNECTED-vi-
estillä
        mHandler.obtainMessage(Game.CONNECTED,bSocket).sendToTarget();

          }
//Metodi jolla suljetaan pistoke
          public void cancel(){
                    try {
                              bSocket.close();
                    } catch (IOException e) {
                              e.printStackTrace();
                    }
          }
}
```

BtServerThread-luokka Liite 4:3(7)

```
//Tämä luokka hoitaa Bluetoothyhteyden palvelin puolen alustuksen
public class BtServerThread extends Thread {

          private final BluetoothServerSocket serverSocket;
          private final String NAME="Q";
          private  final UUID MY_UUID=UUID.fromString("4ba79700-15af-
11e1-be50-0800200c9a66");
          private Context context;
          //private BluetoothManager bManager;
          private boolean connect;
          private Handler mHandler;
          private BluetoothAdapter btAdapter;


          public BtServerThread(Context c, Handler handler){
          //Luodaan väliaikainen BluetoothServerSocket
                    BluetoothServerSocket tmp=null;
                    mHandler=handler;
                    context=c;
                    connect=false;
           //Hankitaan handleri laitteen BluetoothAdapteriin
                     btAdapter=BluetoothAdapter.getDefaultAdapter();

                    try{
//Luodaan RFCOMM-kanavaa kuunteleva pistoke, jota ohjeistetaan kuun-
telemaan NAME ja MY_UUID-tunnuksia kantavilta laitteilta tulevia
yhteyspyyntöjä. Onnistuessaan metodi palauttaa uuden BluetoothServer-
Socketin
tmp=btAdapter.listenUsingRfcommWithServiceRecord(NAME,MY_UUID);

                    }

                    catch(IOException e){
                              }
//Sijoitetaan tmp-olio varsinaiseen BluetoothServerSocket-olioon

                    serverSocket=tmp;
                    Log.d("aThread","listenRfcommSocketToServiceRe-
cord");


          }
```

Liite 4:4(7)

```
        @Override
        public void run(){
                //Luodaan uusi BluetoothSocket olio
                BluetoothSocket socket=null;

                while(true){
                        try{
        Log.d("aThread","Trying to accept serverSocket");

//Aloitetaan kanavan kuuntelu kutsumalla serverSocketin
accept()-metodia. Metodi palauttaa yhdistetyn BluetoothSocketin heti
kun yhteys on luotu.
                socket=serverSocket.accept();

                Log.d("aThread","Accepted serverSocket");

                        }
                catch(IOException e){
                                                Log.e("aTh-
read","ServerSocket not acceptet");
                e.printStackTrace();
                        break;
                                }

        Log.d("aThread","Checking if socket is not null");
                if(socket!=null){
                                                Log.d("aThread","Call-
ing connected");
//Ilmoitetaan pääsäikeelle että yhteys on luotu
                                        mHandler.obtainMessage(Game.-
CONNECTED,socket).sendToTarget();

Log.d("aThread","Connected called, moving on to try catch ");

                try {

                                                                Log.d("aTh-
read","Closing serverSocket");

//Suljetaan palvelinpistoke. Tämä ei sulje kuitenkaan jo luotua
//Bluetooth-pistoketta
                                                                serverSock-
et.close();
                                                                Log.d("aTh-
read","Closed serverSocket");
                break;
                }
                   catch (IOException e) {

Log.e("aThread","server socket couldn't be closed or bManager
created!");

                e.printStackTrace();
                break;
                        }
                        }
                        }
                }
```

Liite 4:5(7)

```
//Tällä metodilla suljetaan yhteys
        public void cancel(){
                try{
                        serverSocket.close();
                }
                catch(IOException e){};
        }

        }
```

BtManagerThread-luokka Liite 4:6(7)

```
//Tämä luokka tulee hoitamaan Bluetooth-yhteyden ylläpidon ja viestin-
nän
public class BtManager extends Thread{

          private final BluetoothSocket bSocket;
          private final InputStream iStream;
          private final OutputStream oStream;
          private Handler mHandler;
          private boolean loop,btmode;

          public BtManager(BluetoothSocket bS,Handler handler,boolean b
){
                    bSocket=bS;
                    loop=true;
                    mHandler=handler;
                    btmode=b;
//Hankitaan asianmukaiset input- ja outputstreamit joiden avulla
//voidaan vastaanottaa ja lähettää viestejä
                    InputStream tmpIn=null;
                    OutputStream tmpOut=null;

                    try {
                              tmpIn=bS.getInputStream();
                              tmpOut=bS.getOutputStream();
                    } catch (IOException e) {
                              e.printStackTrace();
                    }
                    iStream=tmpIn;
                    oStream=tmpOut;

          Log.d("bManager","Here we are; at the BluetoothManager!");
          }



          @Override
          public void run(){

//Luettu viesti säilötään ensin buffer-taulukkoon ja sen jälkeen
bytes-muuttujaan
                    byte[] buffer=new byte[1024];
                    int bytes;
                    Log.d("bManager","Running...");

//Käynnistetään peli tässä vaiheessa jos BtDemo-moodi ei ole päällä
//Tämä ei ole ehkä oikea aika tehdä tätä mutta en ole ajanpuutteen
vuoksi ehtinyt miettimään tätä enempää
                    if(btmode==false){
                    mHand-
ler.obtainMessage(Game.START_THE_GAME).sendToTarget();

                    }
                    int i=0;
                    i++;
```

Liite 4:7(7)

```
                    while(loop==true){

//Lähetetään oman laitteen peliobjektin koordinaatit etälaitteelle

                                mHandler.obtainMessage(Game.WRITE_CO-
ORD).sendToTarget();

                                try {
                                        i++;
                    Log.d("bManager","reading... "+i);
//Luetaan iStreamin avulla  Bluetoothsocketin kautta tulleita viestejä

                    bytes=iStream.read(buffer);
                    Log.d("bManager","read... "+i);

//Kerrotaan pääsäikeelle että koordinaatit ovat vastaanotetut.
mHandler.obtainMessage(Game.COORD_GOT,bytes,-1,buffer).sendToTarget();


                                }
                                 catch (IOException e) {
                    Log.e("bManager","message not read....");

//Jos lukeminen ei onnistu lähetetään CONNECTION_LOST viesti pääsäi-
keelle
mHandler.obtainMessage(Game.CONNECTION_LOST).sendToTarget();

                    e.printStackTrace();
                                break;
                                }
                    }

          }


//Tämä metodi lähettää oStreamin avulla viestin bluetoothsocketin
kautta etälaitteelle
public void write(byte[] bytes){
          try {
                    oStream.write(bytes);

                    Log.d("bManager","message written....");

          } catch (IOException e) {
                    e.printStackTrace();
                    Log.e("bManager","message not written....");

          }
}

//Poistaa bSocketin käytöstä
public void cancel(){
          try {
                    bSocket.close();
          } catch (IOException e) {
                    e.printStackTrace();
          }
}
}
```

Liite 5: Game-luokka Liite 5:1(11)

```
//Tämä on applikaation kolmas aktiviteetti joka suorittaa varsinaisen
pelin käynnistyksen ja ajamisen. Tässä luokassa alustetaan itse
pelinäkymä, puretaan edelliseltä aktiviteetiklta tullut bundle luokan
omaan bundleen jonka tietosisältö siirretään GameInfo-luokkaan
joka hoitaa näiden olennaisten peliin liittyvien tietojen välittämisen
muille luokille.

public class Game extends Activity {
    /** Called when the activity is first created. */

          private Panel p;

          private GameInfo gInfo;
          private BluetoothAdapter bAdapter;
          private ArrayAdapter<String> aAdapter;
          private Bundle bundle;
          private TextView tv,tv2,tv3,tv4,tv5,tv6,tv7;

         private BtManager btManager;
          private BtServerThread serverThread;
          private BtClientThread clientThread;
          private BluetoothDevice theDevice;
           public static final int MESSAGE_READ = 2;
           public static final int WRITE_MESSAGE = 1;
           public static final int CONNECTION_FAILED = 3;
           public static final int CONNECTION_LOST = 4;
           public static final int CONNECTED = 5;
           public static final int CLOSE_CTHREAD=6;
           public static final int START_THE_GAME=7;
           public static final int COORD_GOT=8;
           public static final int WRITE_COORD=9;
          private Context c;
         private int i=0;



public static byte[] int2byte(int[]src) {
    int srcLength = src.length;
    byte[]dst = new byte[srcLength << 2];

    for (int i=0; i<srcLength; i++) {
        int x = src[i];
        int j = i << 2;
        dst[j++] = (byte) ((x >>> 0) & 0xff);
        dst[j++] = (byte) ((x >>> 8) & 0xff);
        dst[j++] = (byte) ((x >>> 16) & 0xff);
        dst[j++] = (byte) ((x >>> 24) & 0xff);
    }
    return dst;
}
```

(Jatkuu)

Liite 5:2(11)

```
public static int[] byte2int(byte[]src) {
    int dstLength = src.length >>> 2;
    int[]dst = new int[dstLength];

    for (int i=0; i<dstLength; i++) {
        int j = i << 2;
        int x = 0;
        x += (src[j++] & 0xff) << 0;
        x += (src[j++] & 0xff) << 8;
        x += (src[j++] & 0xff) << 16;
        x += (src[j++] & 0xff) << 24;
        dst[i] = x;
    }
    return dst;
}
          private final Handler handler=new Handler(){
                    @Override
                    public void handleMessage(Message msg){
                      switch (msg.what) {

                      case MESSAGE_READ:



                         break;
                      case WRITE_MESSAGE:


                Log.d("BtDeviceSetup","WRITE MESSAGE");

                          break;
//Tämä viesti kertoo että yhteys on katkennut
                      case CONNECTION_LOST:
              Log.d("BtDeviceSetup","CONNECTION_LOST MESSAGE CAUGHT");
              //Aloitetaan yhteyden luonti uudestaan
                                  start();
                         tv3.setText("Connection lost "  );

                                  break;
  //Tämä viesti kertoo että yhteyttä ei onnistuttu luomaan
                      case CONNECTION_FAILED:
  Log.d("BtDeviceSetup","CONNECTION_FAILED MESSAGE CAUGHT");

                     //Aloitetaan yhteyden luonti uudestaan
                                  start();
                        tv3.setText("Connection failed"  );

                          break;
    //Tämä viesti kertoo että yhteys on muodostettu
                      case CONNECTED:
  Log.d("BtDeviceSetup","CONNECTED MESSAGE CAUGHT");
//Parsitaan Bluetoothpistoke viestin obj-parametristä ja sijoitetaan
se socket olioon joka lähetetään connected()-metodille.
           BluetoothSocket socket=(BluetoothSocket)msg.obj;

          connected(socket,socket.getRemoteDevice());
                      break;
```

Liite 5:3(11)

```
//Tämä viesti kertoo että asiakassäie voidaan sulkea
                    case CLOSE_CTHREAD:
                              closeClientThread();
                              break;
                    //Tämä viesti aloittaa pelin
                    case START_THE_GAME:

                              startGame();

                              break;
//Tämä viesti kertoo että laite on saanut uudet koordinaatit
//Bluetooth-yhteyden kautta
                    case COORD_GOT:

//Siirretään viestin mukana tullut bittitaulukko readBufX-taulukkoon
                    byte[] readBufX =(byte[])msg.obj ;
                     i++;

//Muunnetaan readBufX-taulukko biteistä takaisin integer-taulukoksi
                     int[] message=byte2int(readBufX);
                     //Jos pelataan ufolla
                    if(gInfo.getPlayer()==1 ){
//Jollei demo-moodi ole päällä asetetaan uudet räjähdys-koordinaatit

                     if(gInfo.getBtDemo()==false){

          gInfo.setXplCoord(message[0],message[1]);
                                                    }
//Kerrotaan käyttäjälle saatujen koordinaattien arvot

tv3.setText("The X-coordinate is: "+ message[0]+" and the y-coordinate
is: "+message[1]+" read "+i+" times" );
Log.e("BtDeviceSetup"," Explosion Coordinate caught and they are x:
"+message[0]+" and "+message[1]);

                              }
                    //Jos pelataan tykillä
                              if(gInfo.getPlayer()==2    ){
//Jollei demo-moodi ole päällä asetetaan uudet ufo-koordinaatit
            if(gInfo.getBtDemo()==false){
                                                  gInfo.setUfoCoord(mes-
sage[0],message[1]);
                              }
//Kerrotaan käyttäjälle saatujen koordinaattien arvot
tv3.setText("The X-coordinate is: "+ message[0]+" and the y-coordinate
is: "+message[1]+" read "+i+" times" );

Log.e("BtDeviceSetup","Ufo Coordinate caught and they are x: "+mes-
sage[0]+" and "+message[1]);

                              }
                              else


                              break;
```

Liite 5:4(11)

```
//Tämä viesti kertoo että on aika lähettää uudet koordinaatit etälait-
teelle
                    case WRITE_COORD:
                                Log.i("WatchTheSkies","writing y");
//Luodaan uusi integer-taulukko johon koordinaatit sijoitetaan
                                int[] coords=new int[2];
          if(gInfo.getPlayer()==1 && gInfo.getBtDemo()==false){
         //Haetaan ufon koordinaatit gInfo-luokasta
                     coords=gInfo.getUfoCoord();
                                                Log.e("BtDevice-
Setup","Ufo Coordinate written and they are x: "+coords[0]+" and "+co-
ords[1]);

                     }
        else if(gInfo.getPlayer()==1 && gInfo.getBtDemo()==false){
         //Haetaan räjähdysten koordinaatit gInfo-luokalta
                          coords=gInfo.getXplCoord();
                                                Log.e("BtDevice-
Setup","Explosion Coordinate written and they are x: "+coords[0]+" and
"+coords[1]);

                     }
//Jos demo-moodi on päällä kovakoodataan koordinaatit taulukkoon
        else if(gInfo.getPlayer()==1 && gInfo.getBtDemo()==true){
                    coords[0]=100;
                       coords[1]=200;
                  }
        else if(gInfo.getPlayer()==2 && gInfo.getBtDemo()==true){
                    coords[0]=300;
                       coords[1]=400;
                  }
//Muutetaan coords-taulukko integereista biteiksi
                  byte[] bytes=int2byte(coords);
//Annetaan bitti-taulukko bytes write()-metodille
                                write(bytes);
                                break;
                   }
    }
         };


//Tämä metodi pyytää btManageria lähettämään out-taulukon määräämät
         //bitit etälaitteelle
         public void write(byte[] out){
                  BtManager b;
         //       Log.e("BtDeviceSetup","Writing a message...");
                  synchronized(this){
                           b=btManager;
                  }
                  b.write(out);

         }
```

Liite 5:5(11)

```
        @Override
          public void onCreate(Bundle savedInstanceState) {
              super.onCreate(savedInstanceState);
              Handler handler=new Handler();
              //Luodaan GameInfo instanssi
            gInfo=new GameInfo(this,handler);
             //Luodaan handleri aktiviteetille


//Alustetaan GameInfo mainmenu-aktiviteetista tulleilla peli-asetuk-
silla

              gInfo.setPlayer(PlayerData());
              gInfo.setHardData(DeviceData());
              gInfo.setTimer(TimerData());
              gInfo.setMute(MuteData());
              gInfo.setBtDemo(BtDemoMode());
              c=this;

              //Jos peliä ei pelata oikeilla laitteilla
              if(bundle.getInt("hardware")==1){
                   startGame();

              }
              //Jos peliä pelataan oikeilla laitteilla
              else if(bundle.getInt("hardware")==2){

             //Vaihdetaan näkymä bt-näkymään
              setContentView(R.layout.btsetup);

              bundle=this.getIntent().getExtras();

               setContentView(R.layout.btsetup);

//Joukko tekstinäkymiä kertomaan käyttäjälle Bluetooth-yhteyden
edistymisestä
               tv=(TextView) findViewById(R.id.progressview);
               tv2=(TextView) findViewById(R.id.progressview2);
               tv3=(TextView) findViewById(R.id.progressview3);
               tv5=(TextView) findViewById(R.id.progressview5);
               //Rekisteröidään applikaatio adapterille
               bAdapter=BluetoothAdapter.getDefaultAdapter();

              if(bundle.getInt("data")==1){
                  tv.setText("SERVER");

              }
              else
              tv.setText("CLIENT");
               //Jollei adapteria löydy lopetetaan aktiviteetti
               if(bAdapter==null){
                 finish();
               }
          if(!bAdapter.isEnabled()){
```

Liite 5:6(11)

```
        //Pyydetään Androidia käynnistämään Bluetooth-ominaisuudet
  Intent enableBt=new Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
                startActivityForResult(enableBt,0);
                            }
           if(bAdapter.isEnabled()){
              //Aloitetaan etälaitteiden etsintä
                    startDiscovery();
           }

           }

            }


         public void startGame(){
 //Näkymänä käytetään game.xml-tiedoston määrittelemää näkymää
                setContentView(R.layout.game);
                p=(Panel) findViewById(R.id.SurfaceView01);
                //Annetaan panelille GameInfo instanssi
               p.setGameInfo(gInfo);
         }


          @Override
protected void onActivityResult(int requestCode, int resultCode, In-
tent data)
                  {
//Kun BluetoothAdapterille mennyt intentti palauttaa RESULT_OK:n
                   //siirrytään startDiscovery()-metodiin
         if(resultCode==RESULT_OK && requestCode==0){

                  startDiscovery();

                           }


                  }
                  //Lähetyksien vastaanottaja
private final BroadcastReceiver bReceiver= new BroadcastReceiver(){
                  @Override
                  public void onReceive(Context c, Intent i){

         //Parsitaan intentistä toiminnan kuvaus
                  String action=i.getAction();
         //Jos toiminta on ACTION_FOUND
                                    if(Bluetooth-
Device.ACTION_FOUND.equals(action)){

//Parsitaan intentistä löydetyn BT-laitteen tiedot
BluetoothDevice device=i.getParcelableExtra(Bluetooth-
Device.EXTRA_DEVICE);
```

Liite 5:7(11)

```
//Jos etälaitteen nimi vastaa "Q":ta

if( device.getName().toString().equals("Q") |  device.getName().to-
String().equals("Q") ){

                    theDevice=device;
                   //Jos pelataan serveri-laitteella

          if(bundle.getInt("data")==1){


                    serverThread=new BtServerThread(c,handler);
                    serverThread.start();
          }


//Jos pelataan asiakaslaitteena
                    if(bundle.getInt("data")==2){

                    clientThread=new BtClientThread(c,device,handler);
                    clientThread.start();

                    }

                              }
                    }
                    }

          };



          public void startDiscovery(){
//Haetaan getBondedDevice():llä kaikki muistista löytyvät etälaitteet
//ja sijoitetaan ne pairedDevices-taulukkoon
Set<BluetoothDevice> pairedDevices=bAdapter.getBondedDevices();

          if(pairedDevices.size()>0){
tv2.setText("Going through the paired devices...");
   for(BluetoothDevice device: pairedDevices){

if( device.getName().toString().equals("Q") | device.getName().to-
String().equals("Q") ){

//Riippuen kummaksi osapuoleksi laite on valittu käynnistetään joko
palvelimen alustava säie serverThread tai asiakas yhteyden muodostama
clientThread. Molemmissa tapauksissa säikeen run()-metodi käyn-
nistetään suorittamalla niiden start()-komennon.
                                                            if(bundle.-
getInt("data")==1){

          serverThread=new BtServerThread(c,handler);
          serverThread.start();
                                                            }
```

Liite 5:8(11)

```
            if(bundle.getInt("data")==2){
            clientThread=new BtClientThread(c,device,handler);
            clientThread.start();
                                        }
                                        }
                                }
                                }

//Jollei pariutuneita laitteita ole löytynyt etsitään uusia laitteita
ympäristöstä.
            else if(pairedDevices.size()==0){

tv.setText("No paired devices found. Trying to find new ones....");

//Tällä intentillä pyydetään lupaa tehdä laite etälaitteille näkyväk-
si.
Intent dIntent=new Intent(BluetoothAdapter.ACTION_REQUEST_DISCOVER-
ABLE);

//Extralla määritellään kuinka pitkäksi ajaksi laite tehdään näkyväksi
//ajan ollessa tässä tapauksessa 300 sekuntia.
                        dIntent.putExtra(BluetoothAdapter.EXTRA_DIS-
COVERABLE_DURATION,300);
                        startActivity(dIntent);

                    if (bAdapter.isDiscovering()) {
                    bAdapter.cancelDiscovery();
                                }

                        bAdapter.startDiscovery();

//Luodaan IntentFiltteri joka päästää nimenomaan ACTION_FOUND-viestin
lävitse
            IntentFilter f=new IntentFilter(BluetoothDevice.ACTION_FOUND);
                    this.registerReceiver(bReceiver,f);
                                }

                    }

                    //Käynnistää säikeet uudestaan
                    public synchronized void start(){

                //Sammutetaan säikeet jolleivät ne ole jo sammutettu
                  if(clientThread!=null){
                    clientThread.cancel();
                    clientThread=null;
                                }
                  if(btManager!=null){
                    btManager.cancel();
                    btManager=null;
                  }
                  if(serverThread!=null){
                    serverThread.cancel();
                    serverThread=null;
                          }
```

Liite 5:9(11)

```
            //Käynnistetään säikeet uudestaan
                if(bundle.getInt("data")==1){
                if(serverThread==null){
                  serverThread=new BtServerThread(this,handler);
                  serverThread.start();
                }
                }
               else if(bundle.getInt("data")==2){
                if(clientThread==null){
          clientThread=new BtClientThread(this,theDevice,handler);
                  clientThread.start();
                }
                }
                  }
                  //Jos yhteys on katkennut
                  public void connectionFailed(){
                            start();
                  }

                  //Tämä metodi käynnistää btManagerin
public synchronized void connected(BluetoothSocket socket, Bluetooth-
Device device){

          Log.d("BtDeviceSetup","connected called!");
          tv5.setText("Making a connection now...");

//Sammutetaan säikeet jotka voivat vielä olla päällä

                if(clientThread!=null){
                  clientThread.cancel();
                  clientThread=null;
                  }

                if(serverThread!=null){
                  serverThread.cancel();
                  serverThread=null;
                }
                if(btManager!=null){
                  btManager.cancel();
                  btManager=null;
                }
          //Käynnistetään btManageri
                  btManager=new BtManager(socket,handler,BtDemoMode());
                  btManager.start();

                  }



          public void connectionLost(){

                    start();

                  }
          public void closeClientThread(){
          clientThread=null;
          }
```

Liite 5:10(11)

```
//Näillä metodeilla haetaan bundlesta alkuvalikossa tehdyt peliasetuk-
set

          //Pelataanko tykillä vai ufolla.
           public int PlayerData(){
                    //Haetaan bundlesta data-avaimen osoittama data
                   bundle=this.getIntent().getExtras();
                   int data=bundle.getInt("data");

                   return data;
                   }

  //Pelataanko oikealla laitteella vai emulaattorilla
           public int DeviceData(){
 //Haetaan bundlesta hardware-avaimen osoittama data

          bundle=this.getIntent().getExtras();
          int devicedata=bundle.getInt("hardware");

          return devicedata;
                   }

            //Peliaika
              public int TimerData(){
          bundle=this.getIntent().getExtras();
          int timerData=bundle.getInt("timer");

          return timerData;
                   }

            //Onko applikaatio hiljaisessa tilassa vai ei?
            public int MuteData(){
          bundle=this.getIntent().getExtras();
          int muteData=bundle.getInt("mute");

          return muteData;
                   }

            public boolean BtDemoMode(){
//Sijoitetaan intentin extroissa mukana ollut data informaatio aktiv-
iteetin omaan bundleen ja sieltä data muuttujaaan
          bundle=this.getIntent().getExtras();
          boolean data=bundle.getBoolean("btdemo");

          return data;
                   }


//Pelaajan painaessa paluu-painiketta suoritetaan tämä metodi
//jonka tarkoitus on tuhota Game-aktiviteetti
           @Override
          public void onDestroy() {
              super.onDestroy();

              if(clientThread!=null){
                    clientThread.cancel();
                     clientThread=null;
                    }
```

Liite 5:11(11)

```
                    if(btManager!=null){
                      btManager.cancel();
                      btManager=null;
                    }

                    if(serverThread!=null){
                      serverThread.cancel();
                      serverThread=null;
                    }
                      if(bAdapter!=null){
                      bAdapter.cancelDiscovery();
                       }
                  unregisterReceiver(bReceiver);
        this.finishActivity(Intent.FLAG_ACTIVITY_PREVIOUS_IS_TOP);
                       }
}
```

Liite 6: GameInfo-luokka Liite 6:1(3)

```
//Tämä luokka välittää aikaisemmalta aktiviteetilta tulleet peli-
asetukset eteenpäin. sekä hoitaa osan muiden luokkien välisestä kom-
munikoinnista

public class GameInfo {

          private Handler mHandler;
          private int player;
          private int[] xplCoord,ufoCoord;
          private int hardData,timer,mute,xPos, yPos,w,h;
          private boolean finished,ufoMoved,btdemo;
          private Game game;

          public GameInfo(Context c, Handler handler){
                    player=0;
                    btdemo=false;
                    finished=false;
                    mute=1;
                    mHandler=handler;
//Taulukot jotka tulevat sisältämään ufon ja räjähdysten
//paikkakoordinaatit
                    xplCoord=new int[2];
                    ufoCoord=new int[2];
                    ufoMoved=false;
          }


          public void setPlayer(int i){

                    player=i;
          }

          public int getPlayer(){
                    return player;
          }

          //Pelataanko peliä oikeilla laitteilla vai ei
          public void setHardData(int d){

                    hardData=d;
          }

          //Pelin aikaraja
          public void setTimer(int d){

                    timer=d;
          }

          //Äänentoisto päällä vai ei
          public void setMute(int d){

                    mute=d;
          }
```

(Jatkuu)

Liite 6:2(3)

```
            //Ufo koordinaatit

            public void setUfoCoord(int x, int y){

                        ufoCoord[0]=x;
                        ufoCoord[1]=y;

            if(getPlayer()==1){
            //
            mHandler.obtainMessage(WatchTheSkies.WRITE_COORD,ufoCoord);
            }



}

            //Räjähdys koordinaatit
            public void setXplCoord(int x, int y){
                        xplCoord[0]=x;
                        xplCoord[1]=y;
                        if(getPlayer()==2){
            //
            mHandler.obtainMessage(WatchTheSkies.WRITE_COORD,xplCoord);
            }

}

//Palauttaa tiedon pelataanko oikeilla laitteilla vai ei
            public int getHardData(){

                        return hardData;
            }

            //Palauttaa pelin aikarajan
            public int getTimerData(){

                        return timer;
            }
//Palauttaa tiedon onko äänentoisto päällä vai ei
            public int getMuteData(){

            return mute;
            }
            //Palauttaa handlerin Game-luokkaan
            public void getGame(Game w){
                        game=w;
            }

            //Palauttaa ufo-koordinaatit
            public int[] getUfoCoord(){
                        return ufoCoord;
            }

            //Palauttaa räjähdys-koordinaatit
            public int[] getXplCoord(){
                        return xplCoord;
            }
```

Liite 6:3(3)

```
//Palauttaa tiedon onko btdemo-moodi päällä vai ei
        public boolean getBtDemo(){
                return btdemo;
        }

        //Onko btdemo-moodi päällä
        public void setBtDemo(boolean b){
                btdemo=b;
        }

        public void Finished(){
                //game.setResult(game.RESULT_OK);
                finished=true;
                //game.Finish();
        }

        //Palauttaa tiedon onko ufoa liikutettu
        public boolean isUfoMoved(){
                return ufoMoved;
        }
        //Palauttaa tiedon onko peli ohi
        public boolean isFinished(){
                return finished;
        }


}
```

Liite 7: Panel- ja CanvasThread-luokat Liite 7:1(7)

```
//Tässä luokassa määritellään pelinäkymä ja käsitellään näytöllä
tapahtuvat kosketustapahtumat.

public class Panel extends SurfaceView implements SurfaceHolder.Call-
back{

          private CanvasThread canvasthread;
          private Canvas mCanvas;
          private Context cntxt;
          private int x,y,dWidth,dHeight;
          private float initialX,initialY;
          private GameObjectManager gManager;
          private Soundtrack s;
          private boolean touchEvent,ufoMoved,timerStarted,mirrorUfo;
          private GameTimer timer;
          private Painter painter;
          private GameInfo gInfo;
          private String direction,directionY;

    public Panel(Context context, AttributeSet attrs) {
                    super(context, attrs);
               getHolder().addCallback(this);
   //CanvasThread tulee hoitamaan pelinäkymän päivittämisen
               canvasthread = new CanvasThread(getHolder(), this);
               //Annetaan näkymälle fokus
               setFocusable(true);
               cntxt=context;
               direction=null;
               directionY=null;
//Tämä boolean kertoo koska palvelinlaitteella pelaava pelaaja on
liikuttanut ufoaan ensimmäisen kerran mikä on merkki siitä että ufon
voi piirtää näkyviin asiakaslaitteenkin ruudulle
               ufoMoved=false;
               timerStarted=false;
               mirrorUfo=false;




          //gObjectRect=new Rect(x/2,y/2, x+100,y+100);
//Luodaan uusi taustamusiikin toistoon tarkoitetty Soundtrack-olio
               s=new Soundtrack(cntxt);
//Luodaan painter-olio joka tulee huolehtimaan tekstien piirtämisestä
          painter=new Painter(cntxt,this.getHolder());

    }
```

(Jatkuu)

Liite 7:2(7)

```
        public void setGameInfo(GameInfo gI){
                //Haetaan handleri näyttöön
                 Display display = ((WindowManager) cntxt.getSys-
temService(Context.WINDOW_SERVICE)).getDefaultDisplay();
//Sijoitetaan näytön leveys ja korkeus muuttujiin dWidth ja dHeight
                 dWidth = display.getWidth();
                 dHeight = display.getHeight();
//Määritetään koordinaatit näytön keskikohdalle ja miinustetaan siitä
//puolet ufon leveydestä
                 x=display.getWidth()/2-50;

                 y=display.getHeight()/2-50;
                 gInfo=gI;
                 //if(gInfo.getPlayer()==1){
//Annetaan ufon aloituskoordinaatit GameInfo-luokalle jonka kautta ne
//lähetetään etälaitteelle


                gInfo.setUfoCoord(x, y);
                // }
//Luodaan uusi GameObjectManager, joka tulee huolehtimaan peliobjek-
tien luonnista ja päivityksestä
 gManager=new GameObjectManager(cntxt,this.getHolder(),this,gInfo);

//Alustetaan ufo piirrettäväksi keskelle näyttöä
                if(gInfo.getPlayer()==1){
                 gManager.ufoMoved(x, y);
                 }
        }


    //Palauttaa SurfaceHolderin canvaksen
        public void getCanvas(Canvas c){
                 mCanvas=c;
        }


//Metodi joka huolehtii piirtotapahtumista
        @Override
        public void onDraw(Canvas canvas) {
                Log.d("ondraw", "lefutott");

                mCanvas=canvas;
//Piirretään tausta joka kerta uudestaan kun onDraw():ta kutsutaan
//mikä vastaa samalla näytön alustamista
                Drawable sky=
        cntxt.getResources().getDrawable(R.drawable.sky_ex01_1600);

                sky.setBounds(0,0,dWidth,dHeight);
               sky.draw(mCanvas);

//Jollei voittoehdot ole täyttyneet kutsutaan GameObjectManagerin up-
date()-metodia mikä huolehtii loppujen peliobjektien piirrosta
if(painter.IsVictoryOne()==false && painter.IsVictoryTwo()==false){

                 gManager.update(mCanvas);
                                        }
```

Liite 7:3(7)

```
//Jos ufo on tuhottu pyydetään GameObjectManageria piirtämään tuhottu
//ufo näytölle, x- ja y-koordinaattien määrittämään paikkaan
          if(painter.IsVictoryTwo()==true){
                                                                    gManager-
.justDrawCrashedUfo(mCanvas,x,y);

                    }

//Jos ufoa on liikutettu ensimmäisen kerran ja pelataan oikeilla lait-
teilla käynnistetään ajastin nyt
if(ufoMoved==true && timerStarted==false && gInfo.getHardData()==2){

                              timer.Start();

//Määritetään timerStarted-boolean todeksi ettei ehtoa suoriteta
toiseen kertaan
                              timerStarted=true;

                    }

//Jos pelataan emulaattorilla ja ajastinta ei ole vielä käynnistetty
//käynnistetään se nyt
          else if( gInfo.getHardData()==1 && timerStarted==false){

                              timer.Start();
                              timerStarted=true;
                    }

//Jollei voittoehdot ole vielä täyttynyt piirretään ajastin näytölle
if(painter.IsVictoryOne()==false && painter.IsVictoryTwo()==false &&
ufoMoved==true || gInfo.getHardData()==1){

//Piirretään ajastin mCanvaksen edustamalle kankaalle ja asetetaan
ajaksi timer-olion getTime()-metodin palauttama aika
                                                                    painter.draw-
Timer(mCanvas,timer.getTime());

                              }

//Kutsutaan painter-olion Victory()-metodia joka piirtää voittotekstin
näytölle jos asianmukaiset ehdot täyttyvät
                    painter.Victory(mCanvas);
                    Log.d("ondraw", "done onDraw(Panel)");

          }



 //Tätä metodia kutsutaan kun surfacessa on tapahtunut muutoksia
          @Override
public void surfaceChanged(SurfaceHolder holder, int format, int
width,int height) {


          }
```

Liite 7:4(7)

```
        @Override
        public void surfaceCreated(SurfaceHolder holder) {
//Aloitetaan musiikintoisto jos äänentoisto on päällä
                if(gInfo.getMuteData()==1){
                        s.initMusic();
                    s.startMusic();
                   }
                //Aloitetaan canvasThread
            canvasthread.setRunning(true);
            canvasthread.start();
//Ilmoitetaan GameObjectManagerille että pelinäkymä
//on alustettu jotta GameObjectManager voi luoda ufon ja räjähdykset
            gManager.panelReady();

          //Pelin ajastin
            timer=new GameTimer(gInfo);


        }


        //Palauttaa tiedon onko ufoa liikutettu
        public boolean IsUfoMoved(){
                return ufoMoved;
        }



        @Override
        public void surfaceDestroyed(SurfaceHolder holder) {
                s.endMusic();
                boolean retry = true;
                canvasthread.setRunning(false);
                while (retry) {
                        try {
                                canvasthread.join();
                                retry = false;
                        } catch (InterruptedException e) {

                        }
                }

        }

        @Override
        public boolean onTouchEvent(final MotionEvent ev) {

                switch (ev.getAction() & MotionEvent.ACTION_MASK) {
                case MotionEvent.ACTION_DOWN: {

                int posX =(int) ev.getX();
                int posY = (int) ev.getY();

                //Suoritetaan jos peliä pelataan ufolla
                if(gInfo.getPlayer()==1){
```

Liite 7:5(7)

```
//Jos kosketustapahtuma on sattunut ufon sisältämän neliön sisällä ja
jos ufoa ei ole kosketettu aikaisemmin alustetaan ufoMoved-ja
touchEvent-boolean todeksi


if(gManager.GetUfoRect().contains(posX, posY )&& touchEvent==false  ){

                              touchEvent=true;
                              ufoMoved=true;
                    }
                    }

//Jos pelataan tykillä suoritetaan tämän ehdon sisältämät toiminnot
                     if(gInfo.getPlayer()==2 ){

                    gManager.explosionEvent(posX, posY);

//Jos peliä pelataan oikeilla laitteilla ja Bluetooth-yhteydellä
//lähetetään räjähdystapahtuman koordinaatit gInfo-luokalle joka
lähettää ne eteenpäin toiselle laitteelle
                    if(gInfo.getHardData()==2){
                    gInfo.setXplCoord(posX, posY);
                              }
                    }
if(gInfo.getPlayer()==2 && gManager.GetUfoRect().contains(posX, posY)
&& ufoMoved==true && gInfo.getHardData()==2 ){

                              gManager.UfoHit();

if(gManager.isUfoDestroyed()==true && painter.IsVictoryOne()==false){
                              painter.setVictoryTwo(true);
                              }
                              }
if(gInfo.getPlayer()==2 && gManager.GetUfoRect().contains(posX, posY)
&& gInfo.getHardData()==1 ){

                              gManager.UfoHit();

if(gManager.isUfoDestroyed()==true && painter.IsVictoryOne()==false){
                              painter.setVictoryTwo(true);
                              }
                              }


                    break;
                    }
```

Liite 7:6(7)

```
                    case MotionEvent.ACTION_UP: {
                        if(touchEvent==true){
                       x=(int)ev.getX();
                       y=(int)ev.getY();
                        touchEvent=false;

     Log.e("Panel","On ACTION_UP touchevent,setUfoX starting");
                        gInfo.setUfoCoord(x, y);


                        }
                            break;
                    }

 //ACTION_MOVE kertoo koska sormea liutetaan  näytöllä
                    case MotionEvent.ACTION_MOVE: {

          if(touchEvent==true && gInfo.getPlayer()==1){
if( (int) ev.getY()<mCanvas.getHeight()-gManager.GetUfoRect().width())
{

                            y=(int)ev.getY();

           Log.e("Panel","On ACTION_MOVE touchevent,setUfoY starting");



if((int) ev.getX()>0 && (int) ev.getX()<mCanvas.getWidth()-gManager.-
GetUfoRect().width()){

                            x=(int)ev.getX();
           Log.e("Panel","On ACTION_MOVE touchevent,setUfoX starting");


                                    }

                                gInfo.setUfoCoord(x,y);

                                    }
                            }

                            break;
                    }

//Jos havaitaan toinen kosketustapahtuma ja jollei valeufoa ole jo
piirretty käsketään GameObjectManagerin piirtää se nyt
          case MotionEvent.ACTION_POINTER_1_DOWN:{
                    if(mirrorUfo==false){

                    gManager.CreateMirrorUfo();
                      mirrorUfo=true;
                            }
                            }
                    break;
                    }
                    return true;
                    }

}
```

CanvasThread Liite 7:7(7)

```
//Tämä luokka hoitaa Panel-luokan onDraw()-metodin kutsumisen
public class CanvasThread extends Thread {
            private SurfaceHolder _surfaceHolder;
            private Panel _panel;
            private boolean _run = false;

    public CanvasThread(SurfaceHolder surfaceHolder, Panel panel) {
        _surfaceHolder = surfaceHolder;
        _panel = panel;
                    }

    public void setRunning(boolean run) {
        _run = run;
    }

    @Override
    public void run() {
          //Luodaan uusi canvas-olio c
        Canvas c;
        //Suoritetaan niin kauan kuin _run on tosi
        while (_run) {
          //Alustetaan canvas-olio null:ksi
            c = null;
            try {

//Alustetaan canvas-olio lockCanvas-metodin palauttamalla
//vain tälle säikeelle tarkoitetulla canvaksella
                c = _surfaceHolder.lockCanvas(null);

                synchronized (_surfaceHolder) {
//Annetaan canvas panel-luokan onDraw()-metodille piirtoa varten
                    _panel.getCanvas(c);

          //Käsketään Panel-luokan piirtää canvakselle
                    _panel.onDraw(c);
                }
            } finally {

//Jotta virheen sattuessa ylläolevassa koodissa Surface ei jäisi
epämääräiseen tilaan suoritetaan unlockCanvasAndPost()-metodi, joka
vapauttaa kanvaksen tämän säikeen käytöltä, vasta täällä

                if (c != null) {
                    _surfaceHolder.unlockCanvasAndPost(c);
                }
            }
        }
    }
}
```

Liite 8:Soundtrack-luokka

```
//Tämä luokka huolehtii pelin taustamusiikin toistosta
public class Soundtrack  {

          private Context c;
          private static MediaPlayer mp;

          public  Soundtrack(Context cntxt){
          c=cntxt;
          new MediaPlayer();
          //Luodaan uusi MediaPlayer
          mp=MediaPlayer.create(c, R.raw.music);
          }


          public void initMusic(){

//Annetaan soittimelle applikaation konteksti ja polku toistettavaan
tiedostoon
          //        mp.create(c, R.raw.music);

          }

          //Lopettaa musiikin toiston
          public void endMusic(){
                    mp.pause();
          }
          //Aloittaa musiikin toiston
          public void startMusic(){
                    mp.start();
          }




}
```

Liite 10: GameObjectManager-luokka Liite 10:1(2)

```
//Luokka joka hoitaa kaikkien peli objektien luonnit ja päivitykset
public class GameObjectManager {

          private SurfaceHolder sHolder;
          private Context context;
          private Ufo ufo,ufoMirror;
          private Panel panel;
          private Rect ufoRect;
          private ExplosionManager eManager;
          private GameInfo gInfo;
          private boolean mirrorUfo;
          private int x,y;

public GameObjectManager(Context c,SurfaceHolder s, Panel p,
GameInfo gI){

                    context=c;
                    sHolder=s;
                    panel=p;
                    mirrorUfo=false;
                    gInfo=gI;
          }

//Tätä metodia kutsutaan heti kun Panel-luokka on ladannut pelinäkymän
          public void panelReady(){

          //Luodaan uusi ufo ja räjähdysmanageri
          ufo=new Ufo(context,panel,sHolder,this,gInfo);
          eManager=new ExplosionManager(context,sHolder,gInfo);

          }

//Tämä metodi luo valeufon näytölle kun Panel-luokka havaitsee toisen
//yhtäaikaisen kosketustapahtuman näytöllä
          public void CreateMirrorUfo(){

//Luodaan uusi ufo samaan kohtaan jossa alkuperäinen ufo oli
                    ufoMirror=new
                    Ufo(context,panel,sHolder,this,gInfo);
                    ufoMirror.newRect(ufo.getXPos(),ufo.getYPos());
                    mirrorUfo=true;
          }

          //Uudet ufon-paikkakoordinaatit
          public void ufoMoved(int xPos, int yPos){
                    x=xPos;
                    y=yPos;
          }
          public void explosionEvent(int x, int y){
//Uuden räjähdys tapahtuman sattuessa pyydetään explosionManageria
luomaan uusi räjähdys x- ja y-koordinaattien määräämään paikkaan
                    eManager.newExplosionEvent(x, y);
                    }
```

(Jatkuu)

Liite 10:2(2)

```
		//Tätä metodia kutsutaan kun ufoon on osuttu
		public void UfoHit(){

			ufo.Destroyed();
		}

		//Tämä metodi palauttaa tiedon onko ufo tuhottu
		public boolean isUfoDestroyed(){
			if(ufo.isUfoDestroyed()==true){
				return true;
			}
			else
				return false;
		}
		//Metodi joka hoitaa peliobjektien päivityksen
		public void update(Canvas c){

			Log.d("GameObjectManager","Updating things");

//Päivitetään ensin ufoRect uusilla koordinaateilla
//ja sen jälkeen pyydetään ufo-luokkaa piirtämään ufo näiden koordin-
aattien osoittamaan paikkaan
			int[] ufoC=new int[2];
			ufoC=gInfo.getUfoCoord();
			ufo.newRect(ufoC[0],ufoC[1]);
			ufo.draw(c);

		//Jos valeufo on luotu
			if(mirrorUfo==true){

			   ufoMirror.draw(c);
			}
		//Päivitetään explosionManageria
			eManager.update(c);
			}


		//Palauttaa ufon sijainnin
		public Rect GetUfoRect(){
			ufoRect=ufo.getRect();
			return ufoRect;
		}

		//Siinä tapauksessa että ufo on tuhottu kutsutaan
		//tätä metodia
		public void justDrawCrashedUfo(Canvas c,int x, int y){
if(panel.IsUfoMoved()==true || gInfo.getHardData()==1 && gInfo.get-
Player()==2 ){
			ufo.newRect(x,y);
			ufo.draw(c);
			}
		if( gInfo.getPlayer()==1){
			ufo.newRect(x,y);
			ufo.draw(c);
				}
		}
}
```

Liite 11:Ufo- ja UfoSoundEffects-luokat Liite 11:1(4)

```
//Tämä luokka edustaa ufoa

public class Ufo  {

          private Drawable mDrawable;
          private boolean ufoNoMore,ufoRectCreated;
          private SurfaceHolder sHolder;
          private int width,height,xPos,yPos;
          private long hitPoints;
          private Rect ufoRect;
          private Context context;
          private UfoSoundEffects uS;
          private GameObjectManager gManager;
          private GameInfo gInfo;

public Ufo(Context cntxt,Panel p,SurfaceHolder s,
GameObjectManager g,GameInfo gi) {

//Handlerit surfaceholderiin ja aktiviteetin contextiin
                    sHolder=s;
                    context=cntxt;
                    gInfo=gi;
                    ufoRectCreated=false;

//Haetaan näytön leveys ja korkeus contextin getResource():n avulla
width= cntxt.getResources().getDisplayMetrics().widthPixels/2;

height= cntxt.getResources().getDisplayMetrics().heightPixels/2;

                    ufoNoMore=false;

//Sijoitetaan resurssien drawable hakemistosta löytyvä
//app_lunar_lander niminen png mDrawable muuttujaan.
mDrawable= cntxt.getResources().getDrawable(R.draw-
able.app_lunar_lander);

//Sijoitetaan ufoC-taulukkoon etälaitteelta tulleet ufo-koordinaatit
                    int[] ufoC=new int[2];
                    ufoC=gInfo.getUfoCoord();
    ufoRect=new Rect(ufoC[0]-50,ufoC[1]-50, ufoC[0]+100,ufoC[1]+100);


                  //Ufolla on viisi terveyspistettä
                     hitPoints=5;
                    gManager=g;

//Jos äänentoisto on päällä niin luodaan uusi UfoSoundEffects-olio
          if(gInfo.getMuteData()==1){
                    if(gInfo.getMuteData()==1){

                    uS=new UfoSoundEffects(context);
                                        }
                              }
          }
```

(Jatkuu)

Liite 11:2(4)

```
        //Palauttaa ufon sijainnin x-koordinaatin
        public int getXPos(){

    return xPos;

        }

        //Palauttaa ufon sijainnin y-koordinaatin
        public int getYPos(){

            return yPos;

                }

//Tämä metodi suoritetaan kun ufo on saanut osuman
        public void Destroyed(){

     //Jos hitPoints muuttuja on nolla niin ufo tuhotaan
                if(hitPoints==0){

//Ufon tuhoutuessa mDrawable muuttuja alustetaan lander_crashed
png:llä
mDrawable= context.getResources().getDrawable(R.draw-
able.lander_crashed);

//Jos äänentoisto on päällä toistetaan asiaan kuuluva efekti

            if(gInfo.getMuteData()==1){
                    uS.Destroyed();
            }

                  ufoNoMore=true;
                }

//Jos hitPoints-muuttuja ei ole nolla niin vähennetään muuttujan arvoa
yhdellä.
                else{
                hitPoints--;

//Jos äänentoisto on päällä toistetaan asiaankuuluva efekti.
        if(gInfo.getMuteData()==1){
                uS.Hit();
                        }
                                }
        }


        public void draw(Canvas canvas)
                {

                synchronized(sHolder){
if(ufoRectCreated==true && gInfo.getPlayer()==1 &&
gInfo.getHardData()==2){

//mDrawable piirretään ufoRect-neliön määrittämien rajojen sis-
äpuolelle
                mDrawable.setBounds(ufoRect);
        //Piirretään ufo canvakselle
                mDrawable.draw(canvas);
```

Liite 11:3(4)

```
//Luodaan uusi Paint-olio ja määritellään sen väriksi musta.
                    Paint paint=new Paint();
                    paint.setColor(Color.BLACK);

//Piirretään canvakselle edellä määritellyn Paintin avulla ufon ter-
veyspisteet
canvas.drawText("Hit Points: " + hitPoints ,(float) width/2+300,10,
paint);
                                }

          else if(gInfo.getPlayer()==1){

                    mDrawable.setBounds(ufoRect);
                    mDrawable.draw(canvas);

                    Paint paint=new Paint();
                    paint.setColor(Color.BLACK);

canvas.drawText("Hit Points: " + hitPoints ,(float) width/2+300,10,
paint);
                                }
          else if(gInfo.getPlayer()==2 && gInfo.getHardData()==1){

                    mDrawable.setBounds(ufoRect);
                    mDrawable.draw(canvas);

                    Paint paint=new Paint();
                    paint.setColor(Color.BLACK);
canvas.drawText("Hit Points: " + hitPoints ,(float) width/2+300,10,
paint);
                                          }

                    }
          }

//Ufon liikkuessa ruudulla päivitetään ufoRectiä asianmukaisesti
GameObjectManagerilta tulevilla koordinaateilla
          public void newRect(int x , int y){

                      ufoRect.set(x-50,y-50,x+100,y+100);

                      xPos=x;
                      yPos=y;
          }

          //Palauttaa ufoRectin
          public Rect getRect(){
                    return ufoRect;
          }

          //Palauttaa booleanin joka kertoo onko ufo tuhottu vai ei
          public boolean isUfoDestroyed(){
                    return ufoNoMore;
          }



}
```

UfoSoundEffects Liite 11:4(4)

```
//Tämä luokkaa toimii vastaavasti kuin MenuSound- ja Explosion-
Sound-luokat joten tätä ei sen enempää kommentoida
public class UfoSoundEffects
{
          private SoundPool sP,sP2;
          private int explosion,bang;
          private HashMap<Integer,Integer> soundsMap;
          private Context c;

         public UfoSoundEffects(Context context){

                   c=context;
         sP=new SoundPool(4, AudioManager.STREAM_MUSIC,100);
         sP2=new SoundPool(4, AudioManager.STREAM_MUSIC,100);

          soundsMap=new HashMap<Integer,Integer>();
          soundsMap.put(explosion, sP.load(c,R.raw.expl05,1));
          soundsMap.put(bang, sP2.load(c,R.raw.bang,2));

         }


         public void run(){

        int volumeLeft=50;
        int volumeRight=50;
        int loopPriority=1;
        int loop=0;
        float rate=1;
        Hit();
        Destroyed();

         }

         public void Hit(){

       int volumeLeft=50;
       int volumeRight=50;
       int loopPriority=1;
       int loop=0;
       float rate=1;
sP.play(soundsMap.get(explosion), volumeLeft, volumeRight, loopPrior-
ity, loop, rate);
         }

          public void Destroyed(){
       int volumeLeft=50;
       int volumeRight=50;
       int loopPriority=1;
       int loop=0;
       float rate=1;
sP2.play(soundsMap.get(bang), volumeLeft, volumeRight, loopPriority,
loop, rate);
         }
}
```

Liite 12: ExplosionManager-, Explosion- ja ExplosionSound-luokat Liite 12:1(5)

```
//Tämä luokka hoitaa kaikkien räjähdysten päivitys- ja luontitapahtu-
mat
public class ExplosionManager  {

          private SurfaceHolder sHolder;
          private Context context;
          //Olio luokasta jossa varsinainen ääniefektin toisto tapahtuu
          private ExplosionSound eSound;
          //Tähän listaan tullaan säilömään kaikki räjähdystapahtumat
          private ArrayList<Explosion> explList;
          //Int taulukko joka sisältää räjähdystapahtumien koordinaatit
          private int[] xplCoord;
          private int i,x,y;
          //Tähän tullaan sijoittamaan handleri GameInfo-luokkaan
          private GameInfo gInfo;

public ExplosionManager(Context cntxt,  SurfaceHolder s,GameInfo gI){

                    context=cntxt;
//Taulukko joka tulee sisältämään räjähdysten paikkakoordinaatit
                    xplCoord=new int[2];
                    sHolder=s;
                    i=0;
                    gInfo=gI;
                    explList=new ArrayList<Explosion>();

//Jos äänentoisto on päällä luodaan uusi explosionSound-olio hoitamaan
//räjähdykseen sisältyvää ääniefektiä
                    if(gInfo.getMuteData()==1){
                    eSound=new ExplosionSound(context);
                    eSound.initCannon();
                    }
          }


          public void newExplosionEvent(int posX, int posY){

                    //Lisätään uusi räjähdys-olio explList:iin
                    explList.add(new Explosion(context,sHolder,gInfo));
                    //Räjähdystapahtuman x- ja y-koordinaatit
                    x=posX;
                    y=posY;

//Haetaan GameInfo-luokalta toisella laitteella tapahtuneet räjähdys-
tapahtumien koordinaatit

               if(gInfo.getPlayer()==1){
                    xplCoord=gInfo.getXplCoord();
                    explList.get(i).setPos(xplCoord[0],xplCoord[1]);
               }
   //Annetaan x- ja y-koordinaatit uudelle räjähdykselle
               else if(gInfo.getPlayer()==2){
                    explList.get(i).setPos(x,y);
               }
```

(Jatkuu)

Liite 12:2(5)

```
  //Jollei äänentoistoa ole poistettu käytöstä toistetaan ääniefekti
                    if(gInfo.getMuteData()==1){
                    eSound.run();
                    }

                    i++;


          }


          public void update(Canvas c){



//Käydään jokainen explList:ssä olevista räjähdys-olioista läpi ja
kutsutaan niiden draw()-metodia
          for(int k=0;k<explList.size();k++){
          if(explList.get(k)!=null){

          explList.get(k).draw(c);


                              }
          }
}

          }
```

Explosion-luokka Liite 12:3(5)

```
//Tämä luokka hoitaa räjähdysten piirron ja animoinnin
public class Explosion extends Drawable {

//Olio joka tulee pitämään sisällään räjähdykseen kuuluvat eri
//kuvat
          private Drawable explosion;

//Taulukko joka säilöö räjähdyksen eri vaiheisiin kuuluvat Drawablet
private static int[] exArray={R.drawable.explosion1,R.drawable.explo-
sion2,R.drawable.explosion3,R.drawable.explosion4,R.drawable.explo-
sion5,R.drawable.explosion6,R.drawable.explosion7};

          private int posX,posY,index,frameIndex;
          private Context c;
          private SurfaceHolder sHolder;
          private GameInfo gInfo;
          private boolean done,device;

public Explosion(Context context,SurfaceHolder s,GameInfo gI){

                    c=context;
                    sHolder=s;
                    done=false;
                    index=0;
                    gInfo=gI;
                    frameIndex=0;
          //Katsotaan pelataanko peliä oikeilla laitteilla vai ei
          if(gInfo.getHardData()==2){
                    device=true;
                              }

          else if(gInfo.getHardData()==1 ||gInfo.getHardData()==0){

          device=false;

          }


          }

//Metodi jonka avulla saadaan asetettua räjähdystapahtuman koordinaat-
it
          public void setPos(int x, int y){

                    posX=x;
                    posY=y;

          }
//Metodi joka palauttaa done-booleanin joka kertoo jos animaatio on
suoritettu loppuun
          public boolean isAnimationDone(){
                    return done;
          }
```

Liite 12:4(5)

```
//Tämä metodi huolehtii Drawablen kankaalle piirrosta
          @Override
          public void draw(Canvas canvas) {

       synchronized(sHolder){
//Jos index on pienempi kuin exArray-taulukon koko
//tarkoittaa se ettei animaatiota ole vielä suoritettu loppuun
                    if(index<exArray.length){
//Sijoitetaan index:in osoittama Drawable taulukosta explosion-draw-
ableen
                                          explosion=c.getResources().get-
Drawable(exArray[index]);

//Määritetään Drawable piirrettäväksi setBounds-metodin kertomien ra-
jojen sisälle
           explosion.setBounds(posX-50, posY-50, posX+50, posY+50);

           //Piirretään Drawable
           explosion.draw(canvas);
//Kasvatetaan frameIndexiä yhdellä joka kerta kun draw-metodia kut-
sutaan
           frameIndex++;
//FrameIndex kertoo kuinka usein animaation yksittäinen ruutu piir-
retään kankaalle, arvon ollessa viisi jos peliä pelataan oikeilla
laitteilla koska laitteiden ruudunpäivitys on emulaattoria nopeampi
            if(frameIndex==5 && device==true){
           index++;
           frameIndex=0;

            }

//Jos peliä pelataan emulaattorilla frameIndexia ei käytetä ollenkaan
            else if(device==false){index++;}
                     }
                     else {done=true;}
            }
          }

          @Override
          public int getOpacity() {
                    // TODO Auto-generated method stub
                    return 0;
          }

          @Override
          public void setAlpha(int alpha) {
                    // TODO Auto-generated method stub

          }

          @Override
          public void setColorFilter(ColorFilter cf) {
                    // TODO Auto-generated method stub

          }

}
```

ExplosionSound Liite 12:5(5)

```
//Luokka on toteutettu erillisenä Threadina jottei UI thread pysähty-
isi tämän suorituksen ajaksi
public class ExplosionSound extends Thread {


          private SoundPool sP;
          private int cannon;
//HashMap on Map:in alaluokka. Map on erilaista dataa sisältävä
//luokka jossa on sisällä eri arvoja ja näitä vastaavia avaimia.
//Tässä tapauksessa HashMappiin annetaan  kaksi integeria jotka vast-
aavat soitettavaa ääntä ja sitä edustavaa ID:tä projektin resurs-
seissa.
          private HashMap<Integer,Integer> soundsMap;
          private Context c;

          public ExplosionSound(Context cntxt){

                    c=cntxt;
//Alustetaan uusi SoundPool-olio. Ensimmäinen parametri määrittää mon-
tako yhtäaikaista streamia käytetään äänen toistoon
//toinen minkä tyyppistä ääntä toistetaan ja kolmas äänenvoimakkuuden
          sP=new SoundPool(4, AudioManager.STREAM_MUSIC,100);

          soundsMap=new HashMap<Integer,Integer>();

          }

          public void initCannon(){

 //Laitetaan cannon-efekti soundsMappiin cannon-avaimen kohdalle
              soundsMap.put(cannon,
              sP.load(c,R.raw.cannon,1));

                    }



                    //Metodi joka suorittaa ääniefektin toiston
                    @Override
          public void run(){

                     //Vasemman ja oikean kanavan äänenvoimakkuus
                   int volumeLeft=50;
                   int volumeRight=50;
                    //Loop-prioriteetti
                   int loopPriority=1;
                   //Toistetaanko ääni uudestaan 0=ei, 1=kyllä
                   int loop=0;
                    //Äänentoiston taso
                   float rate=1;
                  //Toistetaan efekti
sP.play(soundsMap.get(cannon), volumeLeft, volumeRight, loopPriority,
loop, rate);
                    }
}
```

Liite 13: GameTimer- ja Painter-luokat Liite 13:1(3)

```
//Tämä luokka hoitaa peliajan vähentämisen
public class GameTimer{

                private GameInfo gInfo;
                private long time;
                private long gameTime;
                private CountDownTimer timer;

            public GameTimer(GameInfo g ){
            //Handleri GameInfo-luokkaan
                        gInfo=g;
                        time=60000;

//Haetaan GameInfo-luokasta, alkuvalikossa määritelty peliaika
                        gameTime=gInfo.getTimerData()*1000;

//Luodaan uusi CountdownTimer joka alkaa laskea gameTimen määrit-
tämästä ajasta alaspäin joka 1000 millisekunnin välein
                      timer= new CountDownTimer(gameTime, 1000){

//Joka päivityskerta suoritetaan tämä metodi
                            @Override
public void onTick(long millisUntilFinished) {

if(millisUntilFinished<gameTime || millisUntilFinished==gameTime){

            //Sijoitetaan millisekuntit time muuttujaan
                                  time=millisUntilFinished;

             }

                            }
//Ajastuksen loppuessa time=0. Tämä pitää tehdä käsin koska muuten
time jäisi 1:ksi
                            @Override
                                 public void onFinish() {
                                time=0;
                            }
                        };

            }


            //Aloitetaan ajanlasku
            public void Start(){
                        timer.start();
            }

            //Palautetaan nykyinen aika

            public long getTime(){
                        return time;
            }
}
```

(Jatkuu)

Painter Liite 13:2(3)

```
//Tämä luokka huolehtii pelin ajastimen voittoilmoitusten ja piirrosta
public class Painter extends Thread {
private Paint paint,paint2;

private SurfaceHolder sHolder;
private int x,y;
private boolean firstTick,victoryOne,victoryTwo;
         public Painter(Context c ,SurfaceHolder s){

//Määritellään kaksi eri Paintia ajastimen ja voittoilmoitusten
ruudulle piirtämiseen
                   paint=new Paint();
                   paint2=new Paint();

//Sijoitetaan molemmat tekstit keskelle ruutua ja määritellään laskur-
in väriksi musta.
                   paint.setTextAlign(Align.CENTER);
                    paint.setColor(Color.BLACK);
                   paint2.setTextAlign(Align.CENTER);

                   //Haetaan handleri näyttöön
Display display = ((WindowManager) c.getSystemService(Contex-
t.WINDOW_SERVICE)).getDefaultDisplay();

//Sijoitetaan näytön leveys ja korkeus muuttujiin x ja y.
                    x = display.getWidth();
                    y = display.getHeight();

//Tämä boolean kertoo onko ensimmäinen ajastimen päivitys suoritettu
                   firstTick=false;
                   victoryOne=false;
                   victoryTwo=false;
                   sHolder=s;
         }


public void drawTimer(Canvas c,long time){

                   synchronized(sHolder){
          if(time==0 && firstTick==false){time =60000;}

                   Log.v("drawing timer","drawtimer");

//Jos aika on loppunut ja kello on laskenut ensimmäisen sekuntinsa
                   //voittaa ensimmäinen pelaaja
          if(time==0 && firstTick==true && victoryTwo==false){

                    victoryOne=true;
                    Victory(c);
                    }

          Log.v("drawing timer","done with the if");
```

Liite 13:3(3)

```
//Jollei voittoehdot ole täyttyneet päivitetään näytöllä näkyvä
ajastin
            if(victoryOne==false && victoryTwo==false ){
                    Log.v("c.drawtext","c.drawtext");

                    if(time<60000 || time==60000){

//Piirretään annettu teksti canvakselle c, x koordinaatin ollessa puo-
let ruudun leveydestä ja y:n 10.
                    c.drawText("Seconds remaining: "+time/1000,x/2,10,
paint);
             //Boolean joka kertoo koska ajanlasku on alkanut
                    firstTick=true;
                    }
             }
                    }

          }

//Palauttaa booleanin joka kertoo onko ensimmäinen pelaaja voittanut
public boolean IsVictoryOne(){

          return victoryOne;
}

//Palauttaa booleanin joka kertoo onko toinen pelaaja voittanut
public boolean IsVictoryTwo(){

          return victoryTwo;
}

public void setVictoryTwo(boolean v){

          victoryTwo=v;
}

public void Victory(Canvas c){
          //Määritellään tekstin koko
          paint2.setTextSize(20);

          //Suoritetaan tämä jos ufo on voittanut
          if(victoryOne==true && victoryTwo==false){
                    c.drawText("Player One", x/2, y/2-100, paint2);
          }
          //Suoritetaan tämä jos tykki on voittanut
          if(victoryTwo==true && victoryOne==false){
                    c.drawText("Player Two", x/2, y/2-100, paint2);
          }
                    paint2.setTextSize(100);



           //Voittoilmoitus piirretään punaisella
           paint2.setColor(Color.RED);
           if(victoryOne==true || victoryTwo==true){
          c.drawText("   WINS   ", x/2, y/2, paint2);
           }
}}
```

Liite 14 : Esimerkki Ohjelman XML-tiedostot Liite 14:1(6)

mainmenu.xml

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/an-
droid"
    android:id="@+id/mainView"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical" >
   <Button
        android:id="@+id/serverbutton3"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
         android:layout_centerHorizontal="true"
       android:layout_centerVertical="true"
        android:text="@string/server" />
    <Button
        android:id="@+id/clientbutton3"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/serverbutton3"
        android:layout_centerHorizontal="true"


        android:text="@string/client" />



</RelativeLayout>
```

(Jatkuu)

seekbar.xml Liite 14:2(6)

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/an-
droid"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >


  <TextView android:id="@+id/text"

       android:layout_width="wrap_content"

        android:layout_height="wrap_content" />

    <SeekBar
        android:id="@+id/timerbar"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:max="100"
        android:progress="50" />

    <Button
        android:id="@+id/backbutton"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/back" />

</LinearLayout>
```

btsetup.xml Liite 14:3(6)

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/an-
droid"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >

    <TextView
       android:id="@+id/progressview"
         android:layout_width="wrap_content"
    android:layout_height="wrap_content"
        />
    <TextView
       android:id="@+id/progressview2"
         android:layout_width="wrap_content"
    android:layout_height="wrap_content"
        />
    <TextView
       android:id="@+id/progressview3"
         android:layout_width="wrap_content"
    android:layout_height="wrap_content"
        />
    <TextView
       android:id="@+id/progressview4"
         android:layout_width="wrap_content"
    android:layout_height="wrap_content"
        />
     <TextView
       android:id="@+id/progressview5"
         android:layout_width="wrap_content"
    android:layout_height="wrap_content"
        />
      <TextView
       android:id="@+id/progressview6"
         android:layout_width="wrap_content"
    android:layout_height="wrap_content"
        />
       <TextView
       android:id="@+id/progressview7"
         android:layout_width="wrap_content"
    android:layout_height="wrap_content"
        />

</LinearLayout>
```

game.xml Liite 14:4(6)

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/an-
droid"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
<com.test.UfoTest.Panel android:id="@+id/SurfaceView01" android:lay-
out_width="wrap_content"

android:layout_height="wrap_content" android:maxHeight="40dip">



</com.test.UfoTest.Panel>
<TextView
  android:id="@+id/textview"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent"
  />

</FrameLayout>
```

string.xml Liite 14:5(6)

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="hello">Hello World, UfoTest!</string>
    <string name="app_name">WatchTheSkies</string>
    <string name="find_devices">Find devices</string>
    <string name="quit">Quit</string>
      <string name="server">Server</string>
    <string name="client">Client</string>
     <string name="tutorial">Tutorial</string>
      <string name="tutorials">Tutorial for Ufo</string>
       <string name="tutorialc">Tutorial for Cannon</string>
      <string name="device">Device</string>
       <string name="emulator">Emulator</string>
        <string name="shortgame">Set game time</string>
         <string name="longgame">Play a 60 second game</string>
         <string name="mute">Mute</string>
         <string name="ok">Ok</string>
          <string name="back">Back</string>
           <string name="btdemo">Bt Demo</string>
</resources>
```

AndroidManifest.xml Liite 14:6(6)

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
      package="com.test.UfoTest"
      android:versionCode="1"
      android:versionName="1.0">
    <uses-sdk android:minSdkVersion="7" />
    <uses-permission android:name="android.permission.BLUETOOTH_ADMIN"
/>
    <uses-permission android:name="android.permission.BLUETOOTH" />
    <application android:icon="@drawable/icon" android:la-
bel="@string/app_name" android:debuggable="true">
        <activity android:name=".MainMenu"
                  android:label="@string/app_name"
                  android:screenOrientation="landscape">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.cat-
egory.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity android:name=".Game"
                  android:theme="@android:style/Theme.NoTitleBar"
                  android:screenOrientation="landscape">
        </activity>
        <activity android:name="BtDeviceSetup"
                  android:theme="@android:style/Theme.NoTitleBar"
                  android:screenOrientation="landscape">

        </activity>

    </application>
</manifest>
```