Lappeenrannan teknillinen yliopisto
Tietotekniikan osasto

Kontekstitietoisen mobiilipelin suunnittelu ja toteutus

Diplomityön aihe on hyväksytty 13.9.2006 Tietotekniikan osaston osastoneuvostossa.

Työn tarkastajina toimivat Jari Porras ja Kari Heikkinen. Työn ohjaajana toimii Kari Heikkinen.

Lappeenrannassa 19. lokakuuta 2006

Harri Perälä
Ruotsalaisenraitti 3 C 27
53850 Lappeenranta
(05) 412 0728
harri.perala@iki.fi

Tiivistelmä

Lappeenrannan teknillinen yliopisto
Tietotekniikan osasto

Harri Perälä

Kontekstitietoisen mobiilipelin suunnittelu ja toteutus

Diplomityö

2006

iii + 55 sivua, 19 kuvaa, 3 taulukkoa ja 1 liite

Tarkastajat: Professori Jari Porras
Yliassistentti Kari Heikkinen

Hakusanat: Mobiilipelit, pelisuunnittelu, kontekstitietoisuus
Keywords: Mobile games, game design, context awareness

Kontekstitietoisuuden katsotaan voivan parantaa sovellusten ja palvelujen käytettävyyttä matkapuhelimissa. Kontekstitietoisuuden tekniikoita voidaan käyttää myös peleissä, joko siksi, että ne mahdollistavat uudenlaisia pelejä, tai siksi, että peleillä voidaan havainnollistaa ja testata eri tekniikoiden toimintaa.

Diplomityössä esitellään prototyyppi monen pelaajan kontekstitietoisesta mobiilipelistä, jossa pelivälineinä käytetään kamerapuhelimella luettavia tavallisia viivakoodeja. Viivakoodit on yhdistetty palvelimella sijaitsevan pelimaailman kohteisiin, joiden omistuksesta pelaajat kilpailevat. Peliä on tarkoitus arvioida myöhemmin pelattavuuden ja idean kiinnostavuuden kannalta. Prototyypin toinen tehtävä on havainnollistaa Multi-User Publishing Environment (MUPE) -sovellusalustan tukea kontekstitietoisuudelle.

Työ kuvaa pelin suunnittelun, toteutuksen ja arvioinnin alkaen varhaisimmista ideoista ja päättyen osittaiseen prototyyppiin. Prototyypissä on toteutettu osa pelilogiikasta ja käyttöliittymästä, mutta sitä ei ole integroitu kontekstitietoa keräämiin sensoreihin. Pelin suunnittelussa käytettiin apuna heuristista arviointia ja kahta fokusryhmähaastattelua.

Abstract

Lappeenranta University of Technology
Department of Information Technology

Harri Perälä

Design and implementation of a context-aware mobile game

Thesis for the Degree of Master of Science in Technology

2006

iii + 55 pages, 19 figures, 3 tables and 1 appendix

Examiners: Professor Jari Porras
Senior Assistant Kari Heikkinen

Keywords: Mobile games, game design, context awareness

It is believed that context awareness can improve the usability of applications and services in mobile phones. Context-aware technologies can also be used in games. There are two motivations for this: the technologies can enable new types of games, and on the other hand, games can be used to demonstrate and test the technologies.

This thesis presents a prototype of a multi-player context-aware mobile game. Ordinary barcodes, read with a camera phone, are the main game elements. The barcodes are mapped to objects in a persistent game world, and it is the players' goal to compete for the ownership of these objects. The potential of the idea and the playability of the game are going to be evaluated in the future. Another purpose of the prototype is to demonstrate how the Multi-User Publishing Environment (MUPE) application platform supports context awareness.

The thesis describes how the game has been designed, implemented and evaluated, beginning with the earliest ideas and continuing up to a partial prototype. The prototype includes a part of the game logic and user interface, but it is not yet integrated with sensors that gather context information. Evaluation methods used to support the design process included a heuristic evaluation and two focus groups.

Sisältö

Lyhenteet

2D	Two-dimensional, kaksiulotteinen
GPS	Global Positioning System
HIIT	Helsinki Institute for Information Technology
HTTP	Hypertext Transfer Protocol
ITEA	Information Technology for European Advancement
J2ME	Java 2 Micro Edition
J2SE	Java 2 Standard Edition
JAR	Java Archive
LOC	Lines Of Code
LTY	Lappeenrannan teknillinen yliopisto
MIDP	Mobile Information Device Profile
MIT	Massachusetts Institute of Technology
MoMUPE	Mobile Context-Aware Applications and Games
MUPE	Multi-User Publishing Environment
PDA	Personal Digital Assistant
PSP	PlayStation Portable
RFID	Radio Frequency Identification
SNAP	Scalable Network Application Package
UI	User Interface
VTT	Valtion teknillinen tutkimuskeskus
XML	Extensible Markup Language

1 Johdanto

MoMUPE (pidemmältä nimeltään Mobile Context-Aware Applications and Games) [8] on Nokian tutkimuskeskuksen koordinoima projekti, joka pyrkii edistämään kontekstitietoisten sovellusten yleistymistä matkapuhelimissa. Projektissa kehitetään Multi-User Publishing Environment (MUPE) -sovellusalustaa, jonka tarkoitus on nopeuttaa kontekstitietoisten mobiilisovellusten prototyypitystä, sekä luodaan alustan avulla uusia sovelluksia. MoMUPEssa ovat Nokian lisäksi mukana Tampereen ja Lappeenrannan teknilliset yliopistot, Valtion teknillinen tutkimuskeskus VTT sekä Teknillisen korkeakoulun ja Helsingin yliopiston yhteinen tietotekniikan tutkimuslaitos HIIT. Projekti toteutetaan vuosina 2006–2007.

Tässä työssä suunniteltiin ja toteutettiin projektia varten yksi sovellus. MoMUPEssa sovellusten tehtävä on toimia esimerkkeinä kontekstitietoisuuden mahdollisuuksista ja havainnollistaa MUPE-alustan ominaisuuksia. Sovelluksia testataan todellisilla käyttäjillä palautteen keräämiseksi. Sovellukset ovat enimmäkseen pelejä, ja myös tässä työssä toteutettiin peli. Suomelan [19, s. 21, 72] mukaan pelit sopivat hyvin kontekstitietoisuuden tutkimuksessa käytettäviksi prototyypeiksi, koska ne ovat usein vähemmän herkkiä puutteelliselle tai epätarkalle kontekstitiedolle kuin muut sovellukset. Siinä missä hyötysovellus saattaa muuttua käyttökelvottomaksi, jos kontekstitietoa keräävät järjestelmät eivät ole luotettavia, pelissä tämä saattaa vain luoda hieman toisenlaisen pelitilanteen.

1.1 Tärkeimmät käsitteet

Kontekstitietoisuus voidaan lyhyesti selittää esimerkiksi muuttuviin käyttötilanteisiin reagoimiseksi [21]. Tarkempia määritelmiä kontekstille ja kontekstitietoisuudelle sekä luokitteluja eri kontekstin lajeille on olemassa lukuisia. Laajasti käytetty on Anind Deyn kontekstin määritelmä, johon myös MoMUPEen ja MUPE-alustaan liittyvässä tutkimuksessa on usein viitattu (esim. [15][21]). Määritelmässä kontekstia ei rajoiteta juuri muuten kuin vaatimalla, että se liittyy käyttäjän ja sovelluksen vuorovaikutukseen [3]:

> ”Context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is

considered relevant to the interaction between a user and an application, including the user and application themselves."

Dey esittää lisäksi kontekstitietoisuudelle erillisen määritelmän, joko korostaa käyttäjän tehtävään mukautumista [3]:

"A system is context-aware if it uses context to provide relevant information and/or services to the user, where relevancy depends on the user's task."

Sanaa *mobiilipeli* käytetään tässä työssä suppeasti tarkoittamaan matkapuhelimella pelattavaa peliä. Mobile Entertainment Forumin ym. [9] mukaan *mobiiliviihde* tarkoittaa viihdetuotteita, jotka toimivat langattomia verkkoja käyttävillä, mukana kulkevilla henkilökohtaisilla laitteilla. Mobiiliviihteeseen sisältyvät myös pelit, joten tämän määritelmän mukaan mobiilipelien alustoiksi voitaisiin laskea erilaisia PDA-laitteita ja verkkoyhteydellisiä pelilaitteita kuten Sony PSP, tosin määritelmän esittävässä dokumentissa on ilmeisesti ajateltu etupäässä matkapuhelimia. Vieläkin laajemman määritelmän mukaan mobiilipelejä pelataan kaikilla mukana kuljetettavilla laitteilla [2], jolloin mukaan lasketaan esimerkiksi Nintendo Game Boy Advance.

1.2 Tavoitteet ja rajaukset

Diplomityössä kuvataan yhden MoMUPEn pelin ideointi, suunnittelu ja kehittäminen ensimmäiseksi osittain pelattavaksi prototyypiksi. Lopullisen pelin toteuttaminen kaikkine ominaisuuksineen ja sen käyttäjätestaus jäävät työn ulkopuolelle. Diplomityössä tehtyyn käytännön työhön sisältyi kaksi tehtävää: pelisuunnitteluun osallistuminen sekä pelin toteutus. Pelille tehtyjä arviointeja käsitellään työssä sen kannalta, miten niiden tuloksia käytettiin pelisuunnittelussa.

Projektissa kehitettävien pelien tulee täyttää kaksi vaatimusta. Ensinnäkin niiden tulee käyttää yhtä tai useampaa alustan tukemaa kontekstitietoa mielekkäänä osana peliä. Tavoitteena on tuottaa innovatiivisia kontekstitietoisia sovelluksia [8], mihin myös pelien tulee pyrkiä. Toiseksi pelien tulee olla riittävän toimivia ja pelattavia, jotta niille voidaan tehdä tavallisten käyttäjien suorittama pelitestaus.

Tutkimushankkeessa sovellusten ei tarvitse olla yhtä viimeisteltyjä tai esimerkiksi yhtä laajalla laitevalikoimalla testattuja kuin kaupallisissa projekteissa, mutta koska tarkoitus on tehdä lopullinen arviointi todellisilla käyttäjillä, pelin toiminnallisuuden on oltava riittävä arvioinnin mahdollistamiseksi.

Työssä pyrittiin saavuttamaan seuraavat tavoitteet:

1. Työssä suunnitellaan edellä mainittuihin vaatimuksiin sopiva monen pelaajan mobiilipeli. Suunnitelmasta kirjoitetaan alustava pelisuunnitteludokumentti.

2. Pelistä toteutetaan MUPE-alustalla versio, joka sisältää osan suunnitelluista ominaisuuksista. Tämän version pohjalta kehitetään myöhemmin lopullinen peli. Tuloksena on itse ohjelmisto sekä toteutusratkaisujen dokumentaatio.

Tässä kirjallisessa selostuksessa ei käsitellä yleisemmin mobiilipelejä, digitaalisten pelien uusia suuntauksia tai kontekstitietoisuuden tutkimusta.

Loppuosa työstä on jäsennetty seuraavasti. Kaksi seuraavaa lukua esittelee käytettyjä menetelmiä ja työkaluja: luvussa 2 esitellään pelisuunnittelua ja projektissa käytettyjä arviointimenetelmiä, ja luvussa 3 kuvaillaan MUPE-sovellusalusta. Tämän jälkeen seuraa kaksi lukua, joissa kuvataan pelin kehitys ja lopputulokset. Luvussa 4 kuvataan pelisuunnitelman kehitys sekä lopputuloksena syntynyt pelisuunnitteludokumentti sekä esitellään pelin käyttöliittymä. Luvussa 5 kuvataan pelin tekninen toteutus sekä esitetään joukko mittaustuloksia. Luku 6 esittää yhteenvedon ja johtopäätökset.

2 MoMUPE-projektin toimintamuodot

MoMUPE-projektissa pelien suunnittelun tukena käytettiin eri vaiheissa sekä asiantuntijoiden tekemiä arviointeja että käyttäjäpalautetta. Tässä luvussa esitellään eräitä pelisuunnitteluun ja arviointiin liittyviä toimintatapoja. Tekniseen toteutukseen käytetty alusta esitellään luvussa 3.

2.1 Pelisuunnittelu

Pelisuunnittelussa suunnittelija määrittelee pelin ydintoiminnallisuuden, josta käytetään englanniksi nimitystä *gameplay*. Esimerkiksi Björk ja Holopainen määrittelevät termin tarkemmin seuraavasti [1]:

> "we define gameplay simply as the structures of player interaction with the game system and with the other players in the game. Thus, gameplay includes the possibilities, results, and the reason for the players to interact within the game."

Näiden interaktioiden dokumentointiin voidaan käyttää pelisuunnitteludokumenttia. Jos kyseessä on digitaalinen peli, pelisuunnitteludokumentti vastaa osittain tavanomaisen ohjelmiston toiminnallista määrittelyä.

Rouse [16] esittää seuraavan esimerkin pelisuunnitteludokumentin mahdollisesta sisällöstä kaupallisessa projektissa:

1. Introduction/Overview
2. Game Mechanics

   Tämä on dokumentin tärkein luku, jossa kuvataan, mitä pelaaja pelissä tekee ja miten, alkaen perusasioista ja edeten edistyneempiin toimintoihin. Myös pelin käyttöliittymä kuvataan tässä, mutta pelimaailman objektit, hahmot ja vastaavat selostetaan vasta myöhemmin Game Elements -luvussa.
3. Artificial Intelligence

4. Game Elements

   Kuvaus pelin hahmoista, esineistä ja pelimaailman objekteista tai mekanismeista.

5. Story Overview

6. Game Progression

   Tässä luvussa voidaan kuvata esimerkiksi tasoihin jaetun pelin eteneminen taso tasolta. Kaikissa peleissä tälle osuudelle ei välttämättä ole tarvetta.

7. System Menus

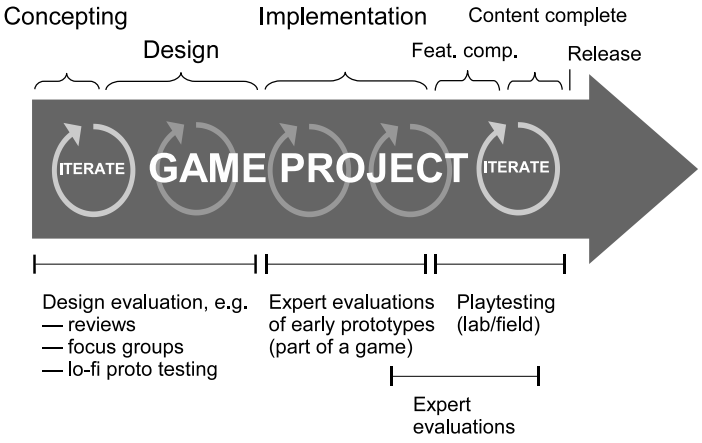
   Kuvaus varsinaisten pelinäkymien ulkopuolisista valikoista.

MoMUPEssa käytössä oli valmis pelisuunnitteludokumentin pohja, joka toimi osaltaan myös ohjeena suunnittelulle. Pohja sisälsi eräissä kohdissa yksityiskohtaisia kysymyssarjoja, joihin dokumentin kirjoittaja saattoi yksinkertaisesti täydentää vastaukset. Rousen esimerkistä pohja erosi mm. siten, että siinä oli varattu omat lukunsa projektin kannalta tärkeille aiheille kuten kontekstitietoisuudelle ja moninpelitoiminnallisuudelle. Tekoälyä ja valikoita, jotka harvoin ovat tärkeitä yksinkertaisissa peleissä, ei mainittu erikseen. Tässä työssä kirjoitettu pelisuunnitteludokumentti esitetään kohdassa 4.5 sivulta 29 alkaen.

2.2 Arviointi

Pelisuunnittelun apuna voidaan käyttää erilaisia arviointimenetelmiä, joista yleisin on epäilemättä tavallinen pelitestaus. Kuvassa 1 on esitetty eräs malli peliprojektista ja sen aikana käytettävistä arvioinneista. Projekti on jaettu konseptointi- (*Concepting*), suunnittelu- (*Design*) ja toteutusvaiheeseen (*Implementation*) sekä julkaisua edeltäviin vaiheisiin, joissa valmiina on toiminnallisuus, mutta ei vielä kaikkea sisältöä (*feature complete*, kuvassa *Feat. comp.*), ja vaiheeseen, jossa myös sisältö on valmis (*Content complete*).

Heuristinen arviointi on yleisesti käytetty menetelmä hyötysovellusten käytettävyyden arvioimiseksi. Arvioinnissa sovelluksen käyttöliittymästä etsitään ongelmia vertaamalla sitä listaan suunnitteluperiaatteita. Menetelmän etuja ovat edullisuus, helppous ja soveltuvuus kehitysprosessin alkuvaiheisiin [13]. Menetelmää



Kuva 1: Arviointimenetelmät peliprojektin eri vaiheissa. Piiretty uudelleen lähteen [6] pohjalta.

on sovellettu myös peleihin. Koivisto ja Korhonen [6] esittävät erityisesti mobiilipelien pelattavuuden arviointiin suunnitellut heuristiset säännöt, joita on käytetty Nokian tutkimuskeskuksessa. Pelien yhteydessä heuristisesta arvioinnista käytetään myös nimitystä *asiantuntija-arviointi* [6, s. 23].

MoMUPE-projektissa peleille on toistaiseksi tehty suunnitteluvaiheen aikana yksi asiantuntija-arviointi Nokian säännöstön pohjalta. Arvioinnilla pyrittiin löytämään ongelmia ja toisaalta myös hyviä puolia pelisuunnitelmista. Arvioinnin kohteena oli senhetkinen pelisuunnitteludokumentin versio ja mahdollinen prototyyppi, joka käytännössä tarkoitti yleensä paperille luonnosteltua käyttöliittymää. Kuvassa 1 tällainen arviointi sijoittuu otsikon *Design evaluation* alle.

Focus group- eli *fokusryhmähaastattelu* [4] tarkoittaa ryhmähaastattelua, jossa tavallisesti 5–10 henkilöä keskustelee annetun aiheen ja tilaisuuden ohjaajan esittämien kysymysten pohjalta. Menetelmällä pyritään luomaan yksittäishaastatteluja luontevampaa keskustelua, joka saattaa tuottaa enemmän tietoja ja myös uusia ideoita. Tuotekehityksessä fokusryhmähaastatteluja voidaan käyttää eri vaiheissa keräämään kohderyhmän mielipiteitä. Aluksi voidaan selvittää, mikä saisi heidät käyttämään suunniteltua tuotetta, myöhemmin taas voidaan testata tuotteen prototyyppejä ja lopulta itse tuotetta.

MoMUPEssa fokusryhmähaastatteluilla on kerätty sekä mielipiteitä sovelluksista

että parannusehdotuksia ja uusia ideoita. Haastatteluja on järjestetty ensimmäisen kerran varhaisten konseptien pohjalta sekä myöhemmin hieman pitemmälle, yleensä varhaisen prototyypin asteelle ehtineille peleille. Kuvasta 1 poiketen fokusryhmien käyttö on siis saattanut sijoittua myös pelin toteutusvaiheeseen. Yksittäisten sovellusten parantamisen lisäksi haastatteluilla pyrittiin selvittämään mahdollisten käyttäjien yleisiä mielipiteitä mm. mobiilipelaamisesta.

Pelitestaus alkaa peliprojektin loppupuolella, kun pelistä on olemassa pelattava versio, jossa ainakin suurin osa toiminnallisuudesta on toteutettu. Ohjelmiston tulisi tässä vaiheessa olla mahdollisimman virheetön, sillä pelitestauksen tarkoitus ei ole virheiden etsintä, ja huonosti toimiva ohjelmisto vaikeuttaa itse pelin arviointia. Koiviston ja Korhosen mallissa [6] pelitestaus tarkoittaa tavallisten, ei-ammattimaisten pelitestaajien suorittamaa testausta, joka yleensä aloitetaan, kun käytettävissä on versio, jossa kaikki ominaisuudet on toteutettu. Rousen [16, s. 483–499] mukaan peliteollisuudessa "perinteinen pelitestaaja" on resurssien salliessa usein palkattu työntekijä, joka testaa täysipäiväisesti. MoMUPEssa pelitestausta vastaa valmiille sovelluksille tehtävä käyttäjäarviointi, joka ei sisälly tähän työhön.

3 MUPE-sovellusalusta

Työssä kehitetty monen pelaajan peli toteutettiin kokonaisuudessaan Java-pohjaisella MUPE-alustalla. Alusta sisältää valmiin tuen sovelluspalvelimen toteuttamiseen, matkapuhelimissa toimivien käyttöliittymien luomiseen, viestinvälitykseen palvelimen ja asiakasohjelmien välillä ja käyttäjien autentikointiin. Lisäksi alustalle on toteutettu valmiita komponentteja, joiden avulla sovellus voi käyttää eräitä kontekstitiedon tyyppejä. Tässä luvussa esitellään sellaisia MUPE-alustan ominaisuuksia, joihin viitataan myöhemmin kuvattaessa pelin toteutusta. Koska käytettävä alusta oli ennalta valittu, muita tapoja toteuttaa mobiilisovelluksia ei käsitellä tarkemmin.

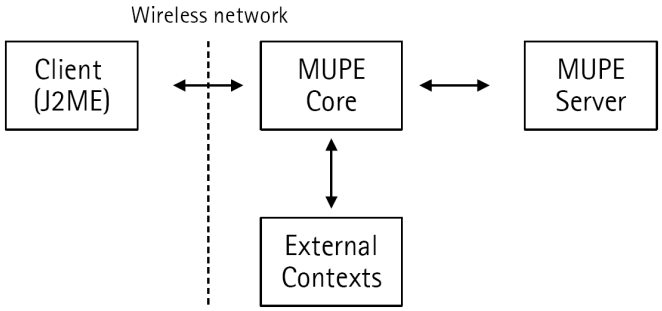
3.1 Yleistä

Vuonna 2003 julkaistu MUPE (Multi-User Publishing Environment) [11] on asiakas-palvelinjärjestelmä, jolla voidaan luoda sovelluksia matkapuhelimille ja muille J2ME-alustan MIDP-profiilia tukeville päätelaitteille. MUPE on kehitetty Nokian tutkimuskeskuksessa yhteistyökumppanien avustuksella. Alustan lähdekoodi on saatavilla Nokia Open Source License (NOKOS) -lisenssin alaisena.

Alustan käyttökohteeksi ilmoitetaan monen käyttäjän kontekstitietoisten sovellusten luominen matkapuhelimille [21]. Verrattuna esimerkiksi kaupallisille mobiiliverkkopeleille tarkoitettuun SNAP Mobile -alustaan [18] MUPE on kevyt ja suunnattu nopeaan prototyypitykseen tutkimuskäytössä. Alustaa ovat toistaiseksi käyttäneet ainakin Nokian tutkimuskeskus, eräät suomalaiset yliopistot, MIT Media Lab, eurooppalaiseen ITEA-ohjelmaan kuulunut Nomadic Media -projekti ja Aucklandin yliopisto Uudessa-Seelannissa [10].

Kuvassa 2 on esitetty yleisnäkymä MUPE-alustasta. MUPE pyrkii helpottamaan mobiilisovellusten kehitystä tarjoamalla oman komentojonokielen (*MUPE script language*), jolla luodaan sovellusten käyttöliittymät ja niiden palvelimesta riippumaton toiminnallisuus. Käyttöliittymä ladataan asiakasohjelmaan, joka on päätelaitteessa toimiva MIDP-sovellus. Samalla asiakasohjelmalla käytetään kaikkia MUPE-pohjaisia palveluja. Palvelun logiikka toteutetaan MUPE-palvelimeen, joka on periaatteeltaan persistentti virtuaalimaailma. MUPE Core on välitason ohjelmisto, joka huolehtii asiakasohjelmien yhteyksistä palvelimeen. Palvelimesta ja

MUPE Coresta koostuva palvelinpuolen ohjelmisto toimii J2SE-alustalla. MUPE-palvelin voi vastaanottaa kontekstitietoa sekä asiakasohjelmilta että erillisiltä kontekstintuottajiksi kutsutuilta palvelimilta, jotka tukevat tiettyä protokollaa.



Kuva 2: MUPEn rakenne [20].

Nykyinen asiakasohjelma käyttää MIDP:n versiota 2.0. Olennainen uusi ominaisuus MIDP 2.0:ssa on tuki matalan tason socket-yhteyksille. Versiossa 1.0 ainoa yhteystyyppi oli HTTP-protokolla, mistä seurasi MUPEn kannalta, että jatkuvaa näkymän päivitystä vaativien sovellusten oli tehtävä ajastimen avulla säännöllisin väliajoin kyselyjä palvelimelle (pull). Nykyinen asiakasohjelma käyttää koko ajan auki olevaa socket-yhteyttä, jonka ansiosta palvelin voi lähettää tarvittavat päivitykset kaikille yhteydessä oleville asiakasohjelmille heti muutosten tapahtuessa (push).

Vuoden 2006 kesällä MUPEn viralliselta sivustolta [11] oli saatavissa kuusi sovellusta: yleiskäyttöinen ryhmätyösovellus MUPEGui sekä pelit FirstStrike, Assassin, Ancient Runes, Constellations ja MUPEDungeon. Näiden lisäksi ei ollut tiedossa yhtään MUPE-sovellusta, jonka lähdekoodi olisi julkaistu. Kun jatkossa jotakin toteutusratkaisua kuvataan tyypilliseksi, arvio perustuu näihin esimerkkeihin.

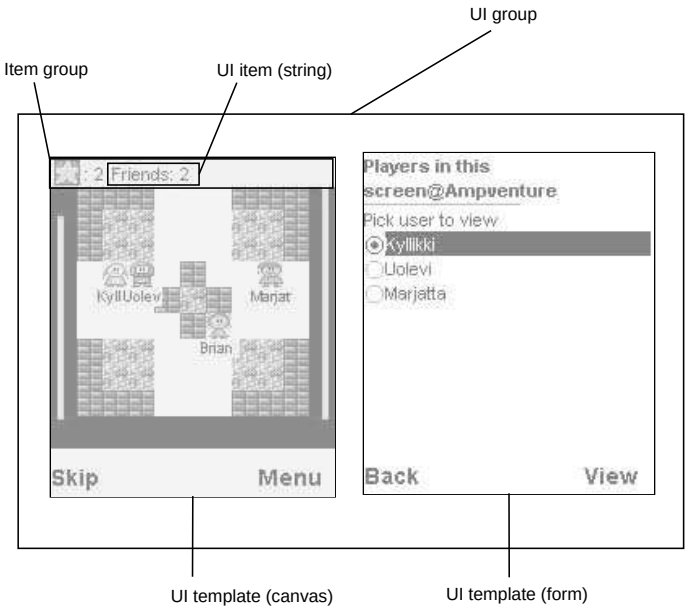
3.2 Käyttöliittymän toteuttaminen

MUPEn XML-pohjaisen komentojonokielen [12] kautta voidaan käyttää päätelaitteen MIDP-rajapintoja mm. käyttöliittymien luomiseen sekä määritellä yksinkertaista toiminnallisuutta, joka vähentää asiakasohjelman riippuvuutta palvelimesta. Kieli perustuu viesteihin, jotka kohdistetaan tietylle asiakasohjelman muistissa

olevalle käyttöliittymäelementille. Palvelin lähettää viestejä asiakasohjelmalle joko push-toiminnon avulla tai vastauksena asiakasohjelman tekemään pyyntöön. Viesti voi mm. luoda tai tuhota käyttöliittymäelementin, muokata elementtiä tai määritellä tapahtumankäsittelijän, joka aktivoituu myöhemmin tietyssä tilanteessa.

Kuvassa 3 on havainnollistettu MUPE-sovelluksen käyttöliittymän koostumista eri tason rakenteista. Aikaisempia suomennoksia rakenteiden nimille ei ollut tiedossa, joten tässä työssä käytetyt käännökset eivät perustu mihinkään lähteeseen. Käyttöliittymän peruselementti on käyttöliittymäpohja (*UI template*), joka edustaa yhtä näkymää. Käyttöliittymäpohja on joko form- tai canvas-tyyppinen. Form-käyttöliittymätyyli käyttää MIDP:n käyttöliittymän korkean tason ohjelmointirajapintaa (mm. luokka *Form*), joka tarjoaa valmiita komponentteja kuten tekstikenttiä ja valintalistoja. Canvas-tyyli käyttää MIDP:n matalan tason ohjelmointirajapintaa (mm. luokka *Canvas*) ja mahdollistaa muun muassa tekstin ja grafiikan vapaan sijoittelun sekä animaation. Sen sijaan tekstisyötteen lukemiseen käyttäjältä ei ole valmista toiminnallisuutta. Näkymät voidaan jakaa ryhmiin (*UI group*), joista asiakasohjelman muistissa pidetään kerrallaan vain yksi. Canvas-käyttöliittymässä voidaan käyttää ikkunaa vastaavia elementtiryhmiä (*item group*).

Kaikissa MUPE-sivuston sovelluksissa on käytetty ensisijaisesti canvas-käyttöliittymiä. Varsinkin pelien kannalta canvas on yleensä välttämätön näyttävän ja pelimäisen käyttöliittymän luomiseksi. Valmiiden käyttöliittymäkomponenttien puute on kuitenkin ongelma, joka hidastaa kehitystyötä. Esimerkiksi vieritettävän listan luominen on työlästä ja vaatii komentojonokielen tarkempaa tuntemusta. Ongelmaa on pyritty korjaamaan julkaisemalla helposti kopioitavia ja muunneltavia esimerkkejä ja toukokuusta 2006 alkaen myös sisällyttämällä asiakasohjelmaan eräänlaisina makroina toteutettujen valmiiden komponenttien kirjasto.

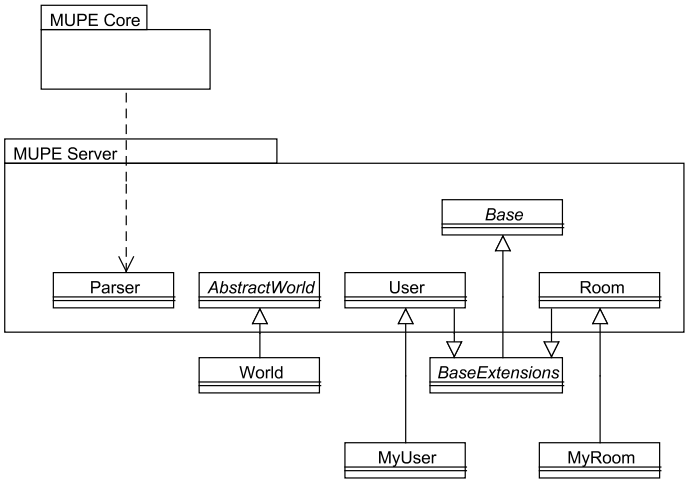


Kuva 3: Esimerkki käyttöliittymän koostumisesta MUPE-asiakasohjelmassa.

3.3 Sovelluksen logiikan toteuttaminen

MUPE-alustan palvelinpuolen ohjelmisto toimitetaan kahtena JAR-pakkauksena, joista toinen sisältää Core-osan ja toinen palvelimen. Uusi sovellus toteutetaan tavallisesti laajentamalla valmiin palvelimen luokkarakennetta (kuva 4). Uuden MUPE-sovelluksen pohjaksi tarvitaan JAR-pakkauksen lisäksi muutamia luokkia, joita kehittäjä muokkaa tarpeen mukaan sovelluskohtaisiksi. Kuvassa näitä luokkia ovat *World* ja *BaseExtensions*. Luokka *World* edustaa MUPE-palvelun luomaa virtuaalimaailmaa. *Base*-luokan jälkeläiset, joita kutsutaan sisältöluokiksi (*content classes*), edustavat maailman sisältöä. Esimerkkejä sisältöluokista kuvassa 4 ovat *User* ja *Room* ja niiden sovelluskohtaiset aliluokat *MyUser* ja *MyRoom*.

Viestintä asiakasohjelman ja palvelimen välillä perustuu push-toiminnon ohella asiakasohjelmien tekemiin metodikutsuihin. Kutsun tekee asiakasohjelman muistissa oleva komentojono, ja se kohdistuu joko palvelimen maailmaolion tai yksittäisen sisältöolion tiettyyn metodiin. Kutsu saapuu MUPE Corelta viestinä, jonka palvelimen *Parser*-luokka muuntaa Java-metodikutsuksi. Kutsun paluuar-



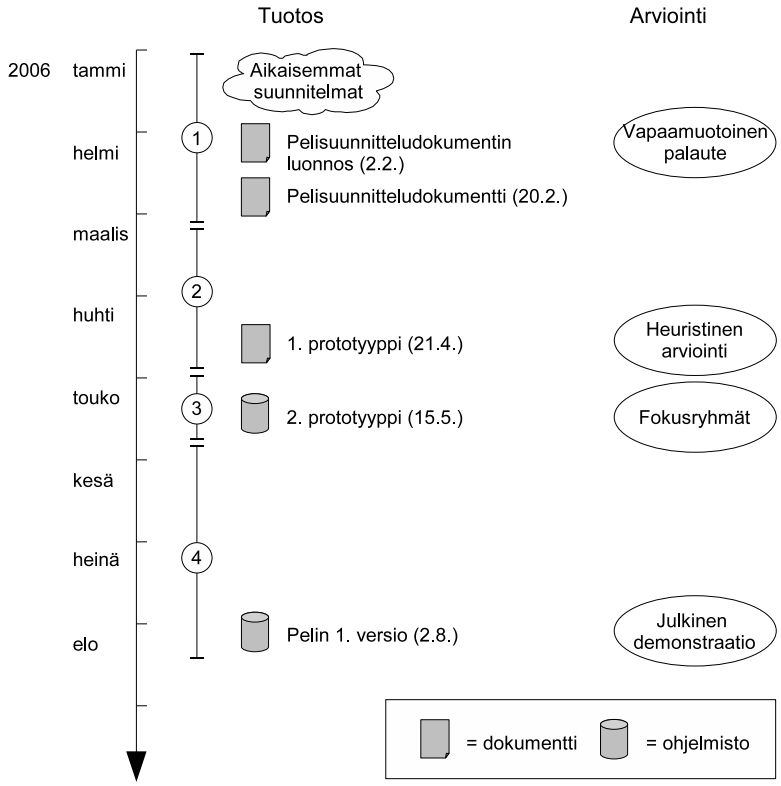
Kuva 4: Uuden MUPE-sovelluksen luominen palvelinta laajentamalla.

vona voidaan lähettää asiakasohjelmalle mitä tahansa viestejä. Kutsu edustaa usein käyttäjän sovellukselle antamaa käskyä, ja palvelimen palauttamat viestit päivittävät käyttöliittymän ulkoasun ja toiminnallisuuden vastaamaan käskyn seurauksia. Tyypillisessä MUPE-sovelluksessa sisältöluokat sekä edustavat sovelluksen käsitteitä että vastaavat itseensä liittyvien käyttöliittymien päivittämisestä.

4 Pelisuunnittelu

Työssä kehitettiin viivakoodien käyttöön perustuva peli, joka sai nimen Collect Me. Pelin kehitys voidaan jakaa neljään vaiheeseen: 1) alkuperäisen pelisuunnitelman laatimiseen, 2) ensimmäisen prototyypin kehittämiseen, 3) toisen, MUPE-pohjaisen prototyypin kehittämiseen ja 4) ensimmäisen osittain pelattavan version toteuttamiseen. Tässä luvussa kuvataan pelisuunnittelun vaiheet ja esitetään lopuksi pelisuunnitelma sellaisena kuin se oli työn lopussa.

Vaiheet ja niissä tuotetut dokumentit ja prototyypit on esitetty kuvassa 5. Tuotosten oikealle puolelle on merkitty menetelmä, jolla peliä on kyseisessä vaiheessa arvioitu. Päiväykset ovat projektisuunnitelman mukaiset takarajat kunkin tuotoksen valmistumiselle.



Kuva 5: Collect Me -pelin kehitys neljään vaiheeseen jaettuna.

4.1 Alkuperäinen pelisuunnitelma

MoMUPE-projektissa pelikonseptien kehittäminen aloitettiin tammikuussa 2006 pidetyllä suunnittelutyöpajalla, johon osallistuivat kaikki projektin osapuolet. Työpajassa synnytettiin eri menetelmin peli- ja muita sovellusideoita ja kirjoitettiin niistä lyhyet kuvaukset. Tätä ennen eri organisaatioissa oli jo kehitetty muutamia konsepteja, joiden joukossa oli myös Lappeenrannan teknillisessä yliopistossa syntynyt Collect Me. Lähes kaikki projektin toistaiseksi tuottamat sovelluskonseptit ovat saaneet alkunsa joko tammikuun suunnittelutyöpajassa tai jo tätä ennen. Kun konseptien jalostaminen oli alkanut, uusia ideoita ei enää näyttänyt syntyvän.

Collect Me -pelin alkuperäinen idea oli Skannerzin [17] innoittama. Radica Gamesin 2001 julkaisemaan alkuperäiseen Skannerz-laitteeseen on yhdistetty näyttö, pelin ohjaimet ja viivakoodinlukija. Pelaaja kerää itselleen hirviöitä lukemalla viivakoodeja. Hirviöitä on kolmea eri heimoa, joista jokaiselle on oma lukijansa. Pelaaja voi kerätä oman heimonsa hirviöitä ja taistella muiden heimojen hirviöitä vastaan. Valmistajan WWW-sivuilta sittemmin poistuneen dokumentin mukaan tuote on suunnattu 7–12-vuotiaille pojille.

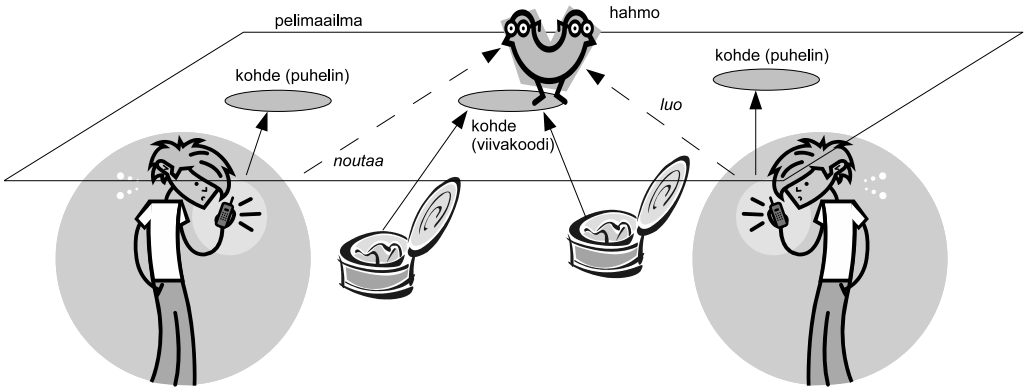
Skannerz edustaa erästä ratkaisua siihen ongelmaan, että pelaajien päivittäisessä ympäristössä ei visioista huolimatta ole vielä juurikaan sensoreita tai muuta tietotekniikkaa, jota voisi käyttää pelissä hyväksi. Viivakoodit ovat koneellisesti luettavia tunnuksia, joiden avulla voidaan muuttaa tavallisia esineitä pelivälineiksi samoin kuin kehittyneemmillä tekniikoilla (esim. RFID-tunnisteet). Ennen Skannerzia viivakoodipelejä on ilmestynyt ainakin 90-luvun alussa sekä pelikonsolien lisälaitteina että itsenäisinä laitteina [5].

Collect Men perusajatus oli hyödyntää MUPEn tarjoamaa virtuaalimaailmaa antamalla pelaajille mahdollisuus (Skannerzista poiketen) luoda omia hahmojaan ja sijoittaa niitä viivakoodeihin. Kun toinen pelaaja missä tahansa lukisi kamerapuhelimellaan saman viivakoodin, hän näkisi siihen sijoitetun hahmon. Pelaajat kuuluisivat ryhmittymiin, ja tämä ratkaisisi, mitä tapahtuu pelaajan löytäessä uuden hahmon. Jos löytäjä olisi samaa ryhmittymää kuin hahmon luoja, hän saisi hahmon itselleen, muussa tapauksessa seuraisi taistelu. LTY:ssa pidettiin tammikuussa ideointikokous, jossa alkuperäistä ideaa kehiteltiin yleisempään suuntaan ja kirjattiin mahdollisia lisäominaisuuksia.

Ensimmäinen pelisuunnitelman luonnos kirjoitettiin ideointikokouksen pohjalta. Luonnoksessa olivat mukana seuraavat ominaisuudet:

Tavoite:	valloittaa ”osoiteavaruus” omalle ryhmittymälle (pelaajan henkilökohtainen tavoite oli jätetty auki)
Pelaajien yhteistyö:	pelaaja kuuluu yhteen kolmesta ryhmittymästä (hyvä, paha, neutraali)
Hahmot:	pelaajat luovat hahmoja, sijoittavat niitä osoitteisiin ja keräilevät muiden oman ryhmittymänsä pelaajien luomia hahmoja
Kontekstitieto:	osoitteet (osoitteen tyyppi oli jätetty auki, mutta viivakoodien mainittiin sopivan peliin parhaiten)

Kuvassa 6 on esitetty pelisuunnitelman tärkeimmät elementit peli- ja reaalimaailmassa. Pelimaailma on esitetty reaalimaailman päällä olevana kerroksena, joka sisältää virtuaalisia objekteja. Samankaltaista vertausta käyttää Suomela [19, s. 29–40] kuvatessaan paikannukseen perustuvia palveluita. Kun käyttäjän sijainti tiedetään, hänelle voidaan tarjota kyseiseen paikaan liittyviä tietoja ja palveluja, jolloin syntyy eräänlainen virtuaalimaailma, jossa tietty digitaalinen sisältö yhdistyy tiettyyn todelliseen paikkaan.



Kuva 6: Pelin elementit varhaisessa suunnitelmassa.

Collect Me -konseptiin ei kuulunut pelaajan paikannusta. Sen sijaan todellisia esineitä yhdistettiin virtuaalisiin objekteihin käyttämällä hyväksi niissä olevia tunnisteita tai osoitteita, joita pelaaja voi lukea päätelaitteellaan. Osoitteiksi on ku-

vassa 6 valittu viivakoodit, jolloin esineet voivat olla tavallisia tuotteita. Pelin hahmot sijaitsivat aina jossakin pelimaailman kohteessa, jota reaalimaailmassa vastasi joko pelaajan puhelin tai jokin viivakoodi. Kun pelaaja lukee viivakoodin puhelimellaan, peli tietää hänen olevan kosketusetäisyydellä kyseisestä tuotteesta, ja hänelle annetaan mahdollisuus käsitellä vastaavaa pelimaailman kohdetta. Pelaaja voi sijoittaa kohteeseen luomansa uuden hahmon tai siirtää kohteessa jo olevan hahmon omaan puhelimeensa. Viivakoodien erikoisuus on, että yleisimmät tuotteet ovat saatavissa mistä tahansa paikkakunnasta tai jopa maasta riippumatta, ja kaikki saman tuotteen kappaleet vastaavat samaa virtuaalista objektia.

Vaikka peruskäsitteet olivat selvät, itse pelin suunnittelu ideoitujen ominaisuuksien ympärille osoittautui haastavaksi. Ilman kokemusta pelisuunnittelusta ja vailla mainittavaa pelien tuntemusta konkreettisen lähtökohdan löytäminen oli vaikeaa. Ongelmana oli erityisesti pelimaailman luonne, joka ei muistuttanut lainkaan tuttuja kaksi- tai kolmiulotteisia pelimaailmoja. Kohteiden välillä ei ollut lainkaan yhteyksiä, joten monista peleistä tuttu alueiden valloittaminen liikuttamalla yksiköitä kartalla ei tullut kysymykseen. Ensimmäisessä luonnoksessa pelaajan tavoite pelissä ja mahdollinen pistejärjestelmä jäivätkin enimmäkseen avoimiksi.

Helmikuun alkupuolella Nokian edustaja antoi palautetta pelisuunnitelmien luonnoksista. Collect Men kommenteissa ehdotettiin, että pelissä käytettäisiin laajemmin eri kontekstitiedon lähteitä. Lisäksi peliin suositeltiin monipuolisempaa resurssien hallintaa, jonka oli tarkoitus mahdollistaa pelaajien erikoistuminen ja taktikointi. Kommenttien perusteella Collect Men pelisuunnitelmasta muokattiin uusia versio. Uudessa suunnitelmassa yksiköt sijaitsivat aina jossakin viivakoodissa ja tuottivat uusia yksiköitä ja muita hyödykkeitä käyttäen koodista saatavia rajallisia resursseja. Muunlaiset osoitteet kuin viivakoodit mainittiin edelleen suunnitelmassa vaihtoehtona, koska käytettävissä olevista tekniikoista ei ollut varmuutta. Tuotantoon vaikutti resurssien lisäksi joukko reaalimaailmasta peräisin olevia arvoja, joita voitaisiin kutsua esimerkiksi kontekstimuuttujiksi. Kontekstimuuttujiksi harkittiin tässä vaiheessa mm. pelaajan kotiseudun luonnonilmiöitä (esim. alueen seisminen toiminta) ja pelaajan päivän aikana vastaanottamien puhelujen määrää. Periaate oli, että eri yksiköt ovat sopeutuneet erilaisiin olosuhteisiin. Pelin tarkka päämäärä ja voittoehdot tai pisteytys olivat tässä suunnitelman versiossa edelleen epäselviä.

Tämä suunnitelma jäi kuitenkin tilapäiseksi välivaiheeksi, kun helmikuun loppu-

puolella Nokian edustaja esitti uuden idean. Uudessa konseptissa pelaaja pyrkii omistamaan tuotteita valtaamalla niitä vastaavia viivakoodeja. Tuotteen omistamisesta saa pisteitä, ja pisteiden määrä riippuu siitä, kuinka usein tuotetta on yritetty vallata. Tällä tavoin tunnetuimmat tuotteet muuttuvat automaattisesti arvokkaimmiksi pelimaailmassa. Pelaajan tavoitteena on pyrkiä jatkuvasti päivittyvän pistelistan kärkeen valtaamalla omistamiensa yksiköiden avulla itselleen mahdollisimman arvokkaan joukon tuotteita. Tässä suunnitelmassa oli aikaisempaan nähden useita etuja, joista ehkä tärkein oli aikaisempaa selkeämpi päämäärä. Myös tuotteiden itsestään syntyvä arvojärjestys ja mahdollisuus kilpailla suosittujen tuotteiden omistuksesta katsottiin kiinnostaviksi, joten uusi idea otettiin jatkokehityksen pohjaksi. Samoihin aikoihin myös päätettiin, että Collect Me tultaisiin toteuttamaan projektin aikana, joten seuraavana tehtävänä oli laatia ensimmäinen prototyyppi tarkempia arviointeja varten.

4.2 Ensimmäinen prototyyppi ja heuristinen arviointi

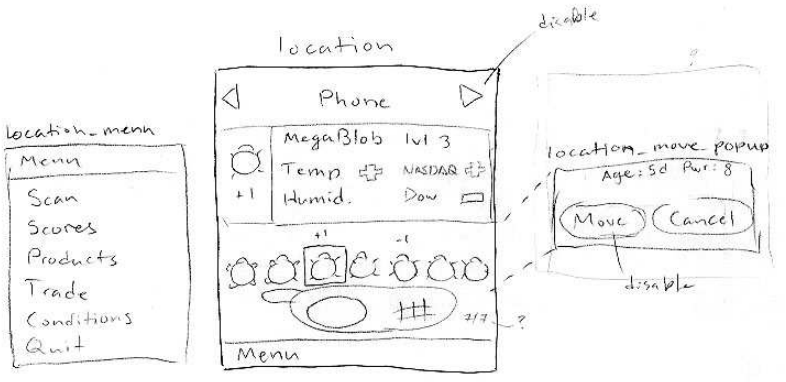
Ensimmäinen prototyyppi toteutettiin maaliskuussa. Pelisuunnitelman tila oli tässä vaiheessa seuraava:

Tavoite:	päästä tuotteita valtaamalla pistelistan kärkeen ja pysyä siellä
Pelaajien yhteistyö:	ei erityistä tukea
Yksiköt:	sijoitetaan viivakoodiin sen valtaamiseksi; uusia yksiköitä keräillään eri tekniikoin
Kontekstitieto:	viivakoodit (vallattavat kohteet); säätila ja osakemarkkinoihin liittyvät indeksit (satunnaisvaihtelu); Bluetooth-laitteiden osoitteet ym. (yksiköiden keräily)

Prototyyppi toteutettiin paperille hahmoteltuina kuvina käyttöliittymästä. Tässä vaiheessa käytettävissä ei ollut graafikkoa tai käyttöliittymäsuunnittelijaa, mikä oli huomattava este näyttävän ja pelattavan pelin suunnittelulle. Kenties paras ratkaisu Collect Men käyttöliittymän suunnitteluun olisi ollut aikaisempien pelien jäljittely, mutta se olisi vaatinut tarkempaa perehtymistä eri mobiilipelien käyttöliittymiin. Lisäksi useimpien kaupallisten pelien käyttöliittymä perustuu runsaaseen ja ammattimaisesti tehtyyn grafiikkaan, jota ilman niiden käyttöliittymäratkaisut eivät ehkä olisi mielekkäitä. Käyttöliittymä suunniteltiin siksi ilman

erityisiä esikuvia.

Kuvassa 7 on luonnos pelin päänäkymästä. Näkymässä pelaaja näkee yhden omistamansa kohteen, joka voi olla joko tuote tai pelaajan oma puhelin kuten kuvassa. Näkymän alaosassa on esitetty kaikki kyseiseen kohteeseen sijoitetut yksiköt, joita voi olla enintään seitsemän. Liikuttamalla kohdistimen (neliö) yksikön kohdalle pelaaja näkee näytön yläosassa yksikön ominaisuudet sekä kontekstimuuttujien senhetkiset positiiviset ja negatiiviset vaikutukset yksikköön. Yksikön valitseminen avaa näkymän päälle ikkunan, jossa pelaaja voi valita *Move* siirtääkseen yksikön toiseen omistamaansa kohteeseen. Pelaajan matkapuhelimessa oletettiin olevan kaksi soft key -näppäintä. Päänäkymässä vasen näppäin avaa valikon, josta pelaaja pääsee mm. pelin pistelistaan. Pelaaja navigoi omistamiensa kohteiden välillä näkymän yläreunan nuolipainikkeiden avulla. Kuvan 7 lisäksi prototyyppiin sisältyi viitisentoista vastaavaa luonnosta.



Kuva 7: Pelin päänäkymä ensimmäisessä prototyypissä.

Pelille tehtiin huhtikuussa asiantuntija-arviointi käyttäen Koiviston ja Korhosen ohjeistoa [6]. Peliä edusti arvioinnissa ensisijaisesti prototyyppi. Käytettyyn ohjeistoon kuuluu kolme ydinmodulia, jotka ovat Usability, Mobility ja Gameplay. Ohjeistoa voidaan laajentaa tiettyä pelityyppiä koskevilla moduleilla, joista itse ohjeiston mukana on julkaistu moduli Multi-player. Usability-moduli (ohjeet GU1–12) käsittelee yleisiä käyttöliittymään liittyviä kysymyksiä, Mobility (MO1–3) mobiililaitteen erikoispiirteitä, Gameplay (GP1–14) varsinaista pelattavuutta ja Multi-player (MP1–6) erilaisia moninpeleihin liittyviä kysymyksiä. Lisäksi arvioinnissa käytettiin vielä kehitteillä ollutta kontekstitietoisuusmodulia [15].

Koska peli oli vielä melko varhaisessa vaiheessa, vain osa ohjeista oli sovellettavissa. Vertaaminen moniin pelin etenemistä ja pelikokemusta koskeviin ohjeisiin (esim. GP5, "Challenge, strategy, and pace are in balance") olisi vaatinut, että pelistä olisi ollut huomattavasti tarkempi suunnitelma. Käyttöliittymää oli jo mahdollista arvioida tarkemmin, tosin esimerkiksi käytettävyysohje GU6, "The player understands the terminology" ei ollut varsinaisesti sovellettavissa, koska taustatarina ja pelissä käytettävä sanasto olivat päättämättä. Taulukoon 1 on koottu osa arvioinnissa paljastuneista puutteista.

Taulukko 1: Heuristisessa arvioinnissa esiin nostettuja ongelmia.

Ohje	**Ongelma**
GU6 Navigation is consistent, logical, and minimalist	Päänäkymän nuolipainikkeita ei käytetä muualla; päänäkymässä merkitykseltään epäselvä ikoni (puhelimen kuva ilmoittamassa, että yksiköt sijaitsevat pelaajan päätelaitteessa); pitkiä navigointipolkuja; ym.
GU8 Game controls are convenient and flexible	Ei mahdollista mukauttaa käyttöliittymää (mm. päänäkymässä näytettävien tietojen valinta voisi olla hyödyllinen), ei oikoteitä kokeneille pelaajille
GP1 The game provides clear goals or supports player-created goals	Ei muuta päämäärää kuin olla tällä hetkellä pistelistan kärjessä
GP8 There are no repetitive or boring tasks	Suuri osa pelistä muodostuu viivakoodien lukemisesta, jossa ei juuri vaihtelua

Osa käyttöliittymän yksityiskohtiin liittyneistä ongelmista korjattiin seuraavissa prototyypeissä. Muun muassa päänäkymän nuolipainikkeet, jotka poikkesivat pelin muusta navigoinnista, jäivät myöhemmin pois, ja merkitykseltään hämärä puhelimen kuva poistettiin, joskin samantapaista ikonia käytettiin myöhemmin toisaalla. Suurimpaan osaan heuristisen arvioinnin antamasta palautteesta ei kuitenkaan reagoitu työn aikana. Koivisto ja Korhonen [6] toteavat ohjeista poikkeamisen olevan täysin hyväksyttävää perustelluista syistä, mutta tässä tapauksessa korjauksia ei sen enempää hyväksytty kuin hylättykään. Monet parannusehdotukset vaikuttivat hyödyllisiltä, mutta niiden toteuttaminen olisi vaatinut käyttöliit-

tymään tai itse peliin huomattavia uudistuksia, joiden suunnittelu olisi vaatinut enemmän asiantuntemusta.

4.3 Toinen prototyyppi ja fokusryhmähaastattelut

Toukokuussa pelistä kehitettiin uusi prototyyppi MUPE-alustalle. Navigointiin ja ulkoasuun tehtiin heuristisen arvioinnin seurauksena tai teknisistä syistä joitain muutoksia, mutta enimmäkseen toisessa prototyypissä vain toteutettiin aiemmin hahmotellut näkymät MUPEn avulla. Yksiköiden ominaisuuksia tai pelin teemaa ei oltu tarkemmin suunniteltu, mutta prototyyppiä varten yksiköt nimettiin marsilaisiksi. Pelisuunnitelmaan syntyi tässä vaiheessa muutama uusi idea. Keskusteluissa projektin sisällä tultiin siihen tulokseen, että pelaajan pitäisi ehkä saada uusia yksiköitä suoraan viivakoodeista sen sijaan, että yksiköiden keräilyyn käytettäisiin jotain erillistä tekniikkaa (esim. Bluetooth-osoitteet), kuten aiemmin oli suunniteltu. Muutos yksinkertaistaisi peliä ja keskittäisi sen entistä selkeämmin viivakoodien ympärille. Lisäksi keksittiin, että pelaajien välinen yhteistyö voisi toimia paikallisesti siten, että fyysisesti samassa tilassa olevat pelaajat voisivat vallata tuotteen yhteisvoimin. Tämä voitaisiin toteuttaa havaitsemalla muut läsnä olevat pelaajat MUPE-asiakasohjelman Bluetooth-toiminnallisuuden avulla.

Lappeenrannassa järjestettiin touko–kesäkuun vaihteessa kaksi fokusryhmähaastattelua, joissa käsiteltiin Collect Me -peliä ja kahta muuta projektissa kehitettyä pelikonseptia. Collect Men osalta haastatteluissa ei käsitelty pelin prototyyppiä vaan peli-ideaa yleisesti, joten toiselle prototyypille itselleen ei tehty työn aikana arviointia. Haastateltaville esitetty näkemys pelistä ei sisältänyt kaikkein uusimpia ideoita, joten ensimmäisen prototyypin yhteydessä esitetty pelisuunnitelman yhteenveto kuvaa myös haastatteluissa käsiteltyä versiota. Uusi ajatus yksiköiden keräilystä suoraan viivakoodeista esitettiin kuitenkin haastattelun aikana vaihtoehtoisena ratkaisuna.

Ensimmäinen ryhmä koostui neljästä miespuolisesta tietotekniikan opiskelijasta, toinen neljästä kauppatieteen opiskelijasta, joista kaksi oli miehiä ja kaksi naisia. Osallistujille jaettiin taustatietona pelistä lyhyt kuvaus ja neljä ruudunkaappausta toisesta prototyypistä. Tämä materiaali on esitetty liitteessä 1 muutamin ymmärrettävyyden vuoksi tehdyin korjauksin. Tarkempia tietoja pelistä annettiin lisäksi keskustelun edetessä tarpeen mukaan. Haastattelussa keskusteltiin yleises-

ti peli-ideasta ja ideoitiin mahdollisia muutoksia ja uusia ominaisuuksia. Osallistuin haastattelujen järjestämiseen valmistamalla jaettavan esittelymateriaalin sekä laatimalla kysymyksiä, jotka osallistujilta kysyttiin haastattelun juontajan omien kysymysten lisäksi. Ensimmäisessä haastattelussa olin myös henkilökohtaisesti paikalla esittelemässä pelin ja vastaamassa kysymyksiin.

Kummassakin ryhmässä pelin perusajatus sai melko myönteisen vastaanoton. Tarkemmat tulokset jaoteltiin kuuden otsikon alle: 1) hyvää, 2) huonoa, 3) uudet ideat, 4) mahdolliset ongelmat, 5) vastaukset ennalta valmistamiini kysymyksiin ja 6) sekalaiset huomiot. Seuraavassa on esitetty (osittain tiivistetyssä muodossa) suurin osa kirjatusta palautteesta. Aineisto perustuu haastattelujen aikana tai niiden jälkeen tehtyihin muistiinpanoihin, jotka eivät ole kovin yksityiskohtaisia, joten kommentteihin sisältyy jonkin verran tulkintaa. Nämä muistiinpanot osoittautuivat kuitenkin käytännössä riittäviksi. Ensimmäinen haastattelu myös nauhoitettiin, mutta nauhan purkaminen tarkemmiksi muistiinpanoiksi ei vaikuttanut tarpeelliselta.

Hyvää:

— Pelin perusajatus oli selitettävissä muutamalla lauseella.
— Kaupalliset mahdollisuudet mainittiin useita kertoja, mm. erilaisia sponsorointimahdollisuuksia pohdittiin.

Huonoa:

— Sään ja talouselämän indeksien käyttämistä satunnaisvaihtelun lähteenä pidettiin tarpeettomana tai häiritsevänä pelin kannalta.
— Pelin marsilaisteema sai toisessa ryhmässä kielteistä palautetta. Tilalle toivottiin jotain, mikä liittyisi paremmin yksiköiden keräilyyn pelissä. Ensimmäisessä ryhmässä asiaan ei kiinnitetty erityistä huomiota.
— Uusien yksiköiden keräily jäi epäselväksi. Tämä johtui ainakin osittain siitä, että esiteltävänä ei ollut selvää suunnitelmaa, vaan useita keskeneräisiä ideoita.

Uusia ideoita:

— Solupaikannuksen käyttö yksiköiden keräilyssä taustamateriaalissa ehdotetun GPS:n sijaan.
— Pelaajien välistä kilpailua koskevia ideoita (erilaisia pisteytysjärjestelmiä, pelaajien jako divisiooniin taitotason mukaan).

— Pelaajan nähtävissä voisi olla esim. viimeksi luettu tuote, viimeksi vallattu tuote ja valittujen pelaajien vertailu.
— Uusi kontekstitieto: kellonaika. Osa yksiköistä voi soveltua päivällä, osa yöllä käytettäviksi.
— Tiedustelijayksiköt, jotka voivat esim. selvittää helposti vallattavia tuotteita tai harhauttaa muita pelaajia antamalla väärän kuvan oman tuotteen puolustuksesta.

Mahdollisia ongelmia:

— Erilaiset huijausmahdollisuudet.
— Aloittelevien pelaajien pärjääminen.
— Bluetoothin käytettävyys on usein huono, joten sen käyttö yksiköiden keräilyyn saattaa olla liian vaikeaselkoista tai hankalaa pelaajalle.
— Pelin säännöistä nostettiin esille seuraava tilanne: jos pelaaja käyttää kaikki yksikkönsä yhden ainoan suositun tuotteen valtaamiseen, hänen on pidettävä kaikki yksikkönsä puolustamassa sitä, jolloin pelissä ei ole enää mitään tehtävää. Ratkaisuksi ehdotettiin, että pelaajalla saisi olla rajaton määrä yksiköitä (esitellyssä pelin versiossa rajoituksena oli seitsemän yksikköä pelaajaa kohti), jolloin peliä voisi jatkaa keräilemällä lisää yksiköitä ja valloittamalla mahdollisesti uusia tuotteita. Pelin tasapaino säilytettäisiin muunlaisilla rajoituksilla kuten rajoittamalla yksiköiden määrää yhtä tuotetta kohden.

Ennalta määritellyt kysymykset (kokonaisuudessaan liitteessä 1):

— Kysymys 1 koski pelin ymmärrettävyyttä ensimmäisellä pelikerralla. Tämä kysymys ei ollut alun perin tarkoitettu suoraan osallistujille, mutta se oli lopulta mukana esittelymateriaalissa ja siitä myös keskusteltiin. Ainakin yhden annetun vastauksen perusteella pelin perusajatus on riittävän helppo ymmärtää myös ilman ohjeita.
— Kysymys 2 pyysi mielipidettä säätilan ja talouslukujen käytöstä satunnaisvaihtelun lähteenä. Kuten edellä kielteisen palautteen kohdalla todettiin, tämä ominaisuus vaikutti osallistujista melko irralliselta lisältä. Lisäksi huomautettiin, että talousluvut eivät juuri kiinnostaisi nuoria pelaajia, jotka olivat osallistujien mielestä paras kohderyhmä pelille.
— Kysymyksessä 3 pyydettiin vertaamaan kahta tapaa, joilla pelaaja voisi saada uusia yksiköitä: erillisten tekniikoiden kuten Bluetooth-osoitteisiin tai paikan-

nukseen perustuvaa keräilyä ja pelkkiin viivakoodeihin perustuvaa keräilyä. Vahvoja mielipiteitä ei esitetty, mutta GPS-paikannukseen perustuvan keräilyn katsottiin olevan kiinnostava, koska se mahdollistaa tiettyyn paikkaan sidotun toiminnallisuuden (kuten yksiköiden etsimisen tietystä liikkeestä).

Sekalaisia huomioita:

— Erään osallistujan mielestä tuotteiden omistuksesta kilpailun tulisi olla pelin perusosa, joka olisi riittävän nopeaa ja helppoa myös satunnaisille pelaajille. Keräily voisi olla keino, jolla innokkaimmat pelaajat saisivat itselleen lisäetua. Toinen osallistuja oli sen sijaan sitä mieltä, että juuri yksiköiden kerääminen saattaisi olla pelin kiinnostavin osa.

Palautteella oli kaksi suoraa vaikutusta. Koska marsilaisista ei pidetty, päätettiin, että ainakaan avaruusolentoja ei tultaisi käyttämään pelin teemana. Yksiköiden ulkomuodon ja ominaisuuksien suunnittelu jätettiin kuitenkin odottamaan, kunnes pelin parissa työskentelemään saataisiin graafikko. Merkittävämpi muutos oli, että sään ja talouslukujen käytöstä luovuttiin palautteen perustella. Muuhun kontekstitiedon käyttöön haastattelut eivät antaneet yhtä selvää suositusta, mutta tässä vaiheessa pelikonseptia haluttiin yksinkertaistaa, joten yksiköiden keräily päätettiin toteuttaa viivakoodien avulla ja muista tekniikoista luovuttiin.

4.4 Pelin ensimmäinen versio ja julkinen demonstraatio

Toisen prototyypin pohjalta kehitettiin ensimmäinen osittain pelattavissa oleva versio. Pelisuunnitelman tila oli tämän version valmistuessa seuraava:

Tavoite:	päästä tuotteita valtaamalla pistelistan kärkeen ja pysyä siellä
Pelaajien yhteistyö:	tuotteiden valtaus yhteisvoimin ja jaettu omistajuus
Yksiköt:	sijoitetaan viivakoodiin sen valtaamiseksi; uusia yksiköitä löytyy koskemattomista viivakoodeista
Kontekstitieto:	viivakoodit (vallattavat kohteet, uudet yksiköt); muiden paikalla olevien pelaajien tunnistaminen Bluetoothin avulla (yhteistyö tuotteiden valtaamisessa).

Peliin oli toteutettu tuotteiden valtaaminen yksinkertaisessa muodossa. MUPEn

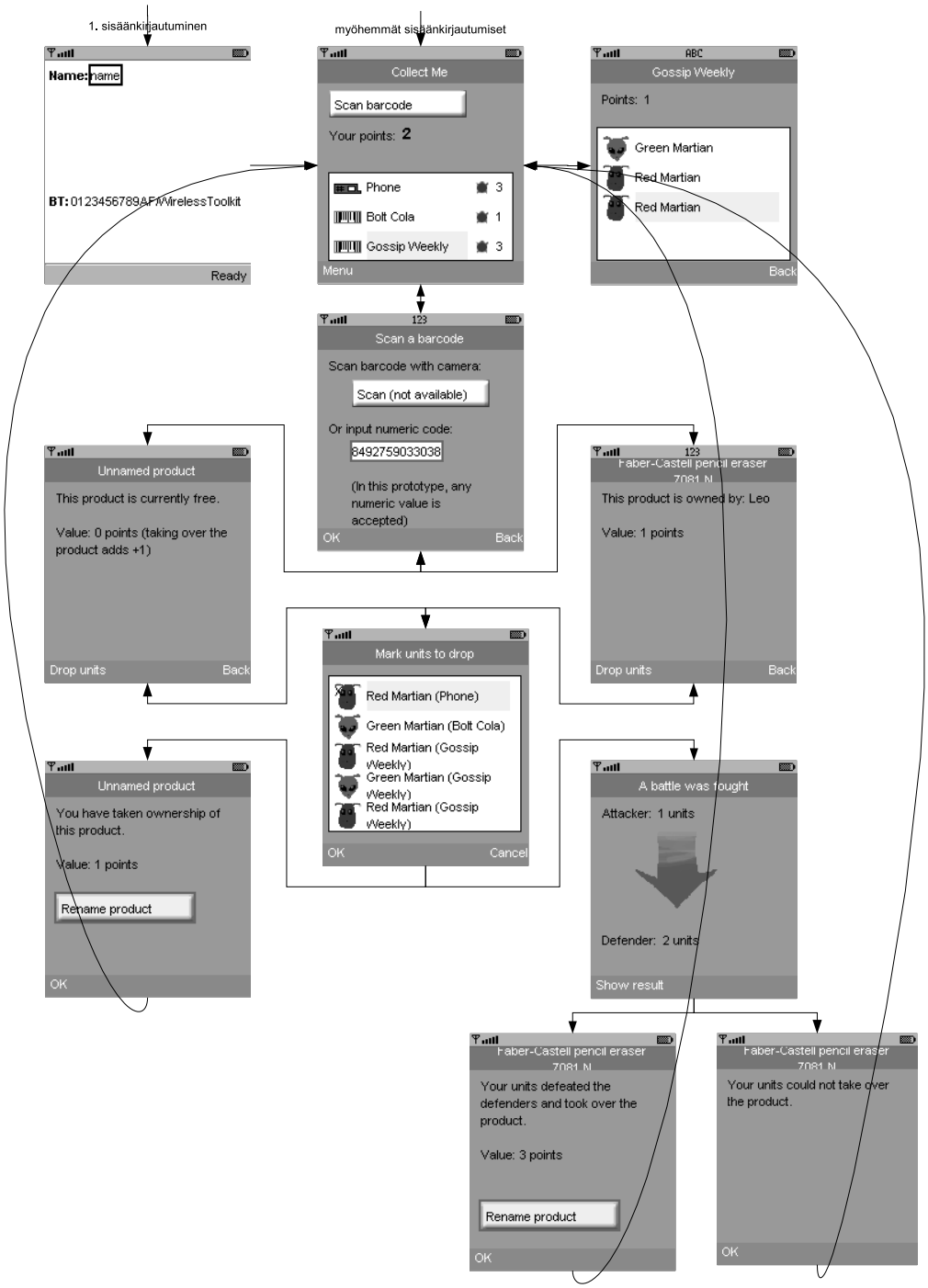
asiakasohjelmaan integroitua viivakoodinlukijaa ei ollut saatavilla, joten koodin lukeminen oli mahdollista ainoastaan syöttämällä numerosarja käsin. Itse pelistä puuttuivat vielä mm. yksiköiden keräily, yhteishyökkäykset ja tietoja esittävät näytöt kuten tuloslista.

Toteutetun pelin kaikki näytöt on koottu kuvaan 8. Ruudunkaappaukset ovat J2ME Wireless Toolkit 2.2:n emulaattorista. Emulaattorissa on käytetty ulkoasua MediaControlSkin, jonka näyttö on kokoa 180 × 208 pikseliä, siis käytännössä sama kuin S60-älypuhelinalustan alkuperäinen näytön koko (176 × 208). Muun muassa listakomponenttien koko on määritelty suhteellisesti, joten ne hyödyntävät ainakin osittain ylimääräisen tilan suuremmissa näytöissä, mutta käyttöliittymää ei ole suunniteltu toimimaan juurikaan kokoa 180 × 208 pienemmillä näytöillä.

Koska pelin teemaa ja yksikkötyyppejä ei ehditty suunnitella ensimmäiseen versioon, aiemmassa prototyypissä esiintyneet marsilaiset on säilytetty. Grafiikka on painikkeita lukuun ottamatta omaa tilapäisgrafiikkaani. Päänäkymään on sijoitettu pelaajan pistemäärä sekä lista hänen omistamistaan kohteista, joihin on laskettu myös pelaajan oma puhelin. Valitsemalla kohteen listasta pelaaja voi siirtyä tarkempaan näkymään, jossa näkyvät kohteeseen sijoitetut yksiköt. Päänäkymän valikossa, joka avataan vasemmalla soft keyllä, on tässä versiossa ainoastaan kaksi toimintoa: koko käyttöliittymän lataaminen uudelleen ja sovelluksesta poistuminen.

Painikkeella ”Scan barcode” pelaaja pääsee näkymään, jossa voi syöttää viivakoodin. Kun pelaaja on syöttänyt koodin ja hyväksyy lukemisen, näytetään koodin sisältö. Kuvassa 8 on esitetty erikseen tapaus, jossa luettu tuote on tyhjä (kuvan vasen laita) ja tapaus, jossa tuote on toisen pelaajan omistuksessa (oikea laita). Kummassakin tapauksessa valinta ”Drop units” vie näyttöön, jossa esitetään kaikki pelaajan yksiköt nykyisine sijainteineen. Pelaaja merkitsee haluamansa yksiköt ja hyväksyy pudotuksen. Onnistuneen tuotteen valtauksen jälkeen on mahdollista nimetä tuote uudelleen (”Rename product” -painike). Nimen syöttäminen tapahtuu tavallisella MIDP-toteutuksen mukaisella lomakkeella kuten muutkin syötteiden lukemiset sovelluksessa.

Tämä Collect Men versio ja muita MoMUPEn sovelluksia oli nähtävillä Lappeenrannan teknillisen yliopiston tietoliikennetekniikan laitoksen järjestämän kesäkoulun yhteydessä 3. elokuuta. Kesäkoulun ohjelmaan sisältyi lyhyt esittelytuokio,



Kuva 8: Toteutettu osa käyttöliittymästä.

jossa läsnäolijoilla oli mahdollisuus kokeilla sovelluksia tavallisissa matkapuhelimissa. Collect Men esittelyyn oli käytettävissä yksi Nokian 6600-malli. Esittelyn aikana muutama ihminen tutustui peliin pikaisesti. Esittelytilanne ei ollut erityisen sopiva palautteen keräämiseen, mutta tilanne antoi silti mielenkiintoista tietoa, sillä kyseessä oli ensimmäinen kerta, kun joku projektin ulkopuolelta todella käytti pelin käyttöliittymää.

Eräät itsestään selviksi katsotut toiminnot osoittautuivat vaikeiksi. Ainakin päänäkymän valikon käyttö, koodinumeron syöttäminen ja tuotteen valtaaminen yksiköitä "pudottamalla" olivat epäselviä yhdelle tai useammalle kokeilijalle. Lisäksi päänäkymän listassa näkyvän kohdan "Phone" (pelaajan oma puhelin) merkitys oli epäselvä eräälle projektin jäsenelle, vaikka hän tunsi peli-idean suhteellisen hyvin. Osa käyttöliittymän ongelmista johtuu luultavasti siitä, että valikoiden, painikkeiden ym. toiminta ja ulkoasu eivät noudata tarkasti minkään tutun ympäristön käytäntöjä, joten käyttäjä ei pysty varmuudella päättelemään, miten niiden on tarkoitus toimia. Peliä kokeilleet pystyivät kuitenkin enimmäkseen navigoimaan näytöissä ilman opastusta.

Seuraavaksi esitetään pelisuunnitteludokumentti, joka vastaa käsitystä pelistä elokuun julkisen esittelyn aikaan. Alkuperäisen englanninkielisen dokumentin rakenne on enimmäkseen säilytetty, mutta pois on jätetty alussa ollut tiivistelmä sekä kohtia, jotka olivat vielä käytännössä tyhjiä.

4.5 Päivitetty pelisuunnitelma

Collect Me

Alkuperäinen idea: Jari Porras
Pelisuunnittelu: Harri Perälä, Riku Suomela ym.

4.5.1 Yhteenveto

Visio pelistä. Pelaaja voi kilpailla minkä tahansa tavallisen viivakoodin omistuksesta. Pelin kulku on yksinkertainen ja intuitiivinen: pelaaja valtaa tuotteen, nimeää sen halutessaan uudelleen ja tämän jälkeen puolustaa sitä.

Pelin elementit. Viivakoodit edustavat tuotteita. Pelaaja saa tuotteen omistukseensa sijoittamalla siihen yksiköitä. Tuotteen omistamisesta saa vaikutusvaltapisteitä. Osasta tuotteita löytyy uusia yksiköitä.

Lyhyt kuvaus. Kun pelaaja lukee tuotteen viivakoodin, peli joko näyttää niiden pelaajien nimet, jotka omistavat tuotteen, tai ilmoittaa tuotteen olevan vapaa. Koodissa voi olla yksikkö, jota kukaan pelaaja ei omista. Pelaaja voi kerätä näitä yksiköitä, mutta kerrallaan pelaaja voi omistaa vain seitsemän yksikköä.

Saadakseen tuotteen omistukseensa pelaaja voi lähettää sitä valtaamaan yhden tai useampia yksiköitä. Jos koodi oli varattu, seuraa taistelu, jonka voittaja saa koodin omistajuuden. Koodin vaikutusvaltapisteiden määrä on sama kuin niiden pelaajien lukumäärä, jotka ovat toistaiseksi yrittäneet vallata sen. Hyökkäykseen osallistuneet yksiköt eivät pysty liikkumaan uudestaan saman vuorokauden aikana (todellista aikaa), joten pelaaja ei voi siirtää osaa yksiköistä heti takaisin puolustamaan muita koodeja.

Fyysisesti samassa tilassa olevat pelaajat voivat myös tehdä yhteishyökkäyksen samaan koodiin. Jos hyökkäys onnistuu, pelaajat jakavat tuotteen omistajuuden. Tuotteen vaikutusvaltapisteet jaetaan tasan pelaajien kesken riippumatta siitä, kuinka paljon kukin osallistui hyökkäykseen.

Kohdeyleisö. Ei selvää käsitystä. Idea saattaa vedota eniten Skannerzille ilmoitettuun kohderyhmään eli 7–12-vuotiaisiin poikiin. Toisaalta peliin on tulossa yksinkertaisia viivakoodipelejä monipuolisempaa taktikointia.

Taustatarina lyhyesti. Ei kirjoitettu.

Genre. Monen pelaajan kevyt strategia?

Erikoispiirteet (*unique selling points*). Pelaaja voi omistaa kaikki saman tuotteen kappaleet maailmassa.

4.5.2 Peruskysymykset

Miten peliä pelataan? Pelaaja kerää itselleen hyvän yhdistelmän yksiköitä ja ottaa haltuunsa yhden tai useampia viivakoodeja. On mahdollista valita, pitääkö hallussaan yhtä hyvin arvokasta tuotetta vai jakaako voimansa moneen vähäisempään kohteeseen.

Miksi peli kannattaa toteuttaa? Varhaisista viivakoodipeleistä poiketen peli hyödyntää persistenttiä monen pelaajan pelimaailmaa ja tukee paikallista yhteistyötä pelaajien välillä.

Missä peliä pelataan? Missä tahansa paikassa, jossa saatavilla on viivakoodeja.

Mitä pelaaja ohjaa? Omia yksiköitään.

Montako hahmoa pelaaja ohjaa? Enintään seitsemää.

Mikä pelissä on erikoista? Pelin sijainnit ovat tavallisia esineitä.

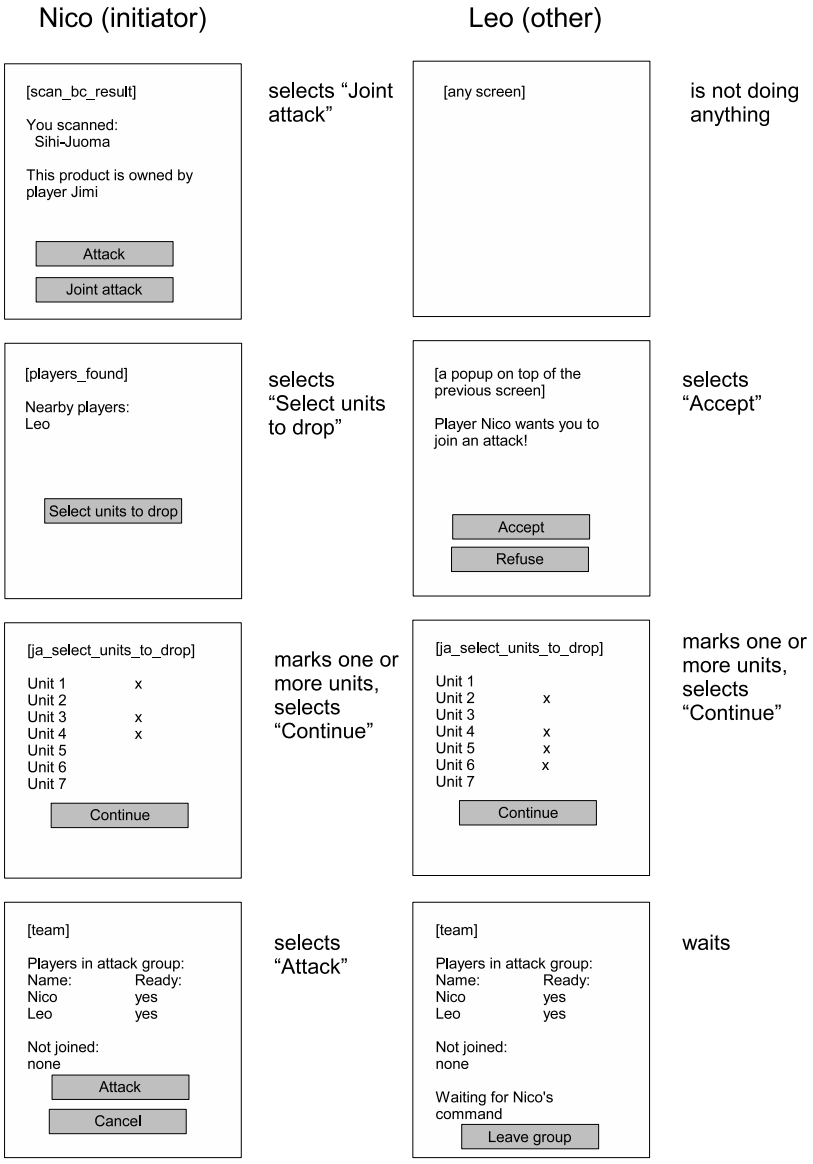
Miten paljon aikaa pelaaminen vaatii? Ei arvioitu.

4.5.3 Päivä pelaajan elämässä

Leo ja Nico pelaavat kamerapuhelimillaan Collect Me -peliä tamperelaisessa supermarketissa. He näkevät pelin suosituimpien tuotteiden listasta, että eräs virvoitusjuomapullo on parhaillaan arvokas kohde pelissä. Pojat aikovat vallata sen yhdessä, mutta ensin he lukevat muita viivakoodeja löytääkseen uusia yksiköitä. Leo löytää hyvän yksikön tonnikalapurkista ja korvaa sillä yhden omista yksiköistään.

Seuraavaksi Nico lukee virvoitusjuomapullon koodin. Peli kertoo hänelle, että tuotteen omistaa Rovaniemellä asuva pelaaja nimeltä Jimi. Nico valitsee toiminnon "Yhteishyökkäys". Peli havaitsee, että myös Leo on paikalla, ja näyttää hänen ruudullaan kutsun liittyä hyökkäykseen. Leo hyväksyy kutsun, ja molemmat pojat ryhtyvät valitsemaan hyökkäyksessä käytettäviä yksiköitä. He vertailevat puhelintensa näyttöjä ja neuvottelevat. Päätetään, että Nico käyttää kolmea yksikköä ja Leo neljää. Tapahtuman kulku on esitetty kuvassa 9.

Kun molemmat ovat valmiita, Nico käynnistää hyökkäyksen ja tulokset ilmestyvät molemmille näytöille. Nicon ja Leon yhdistetyt voimat riittivät ajamaan Jimin puolustavat yksiköt pois koodista. Kumpikin saa tästä lähtien puolet tuotteen vaikutusvaltapisteistä.



Kuva 9: Esimerkki kahden pelaajan yhteishyökkäyksestä.

4.5.4 Pelimekaniikka

Pelin aloittaminen. Kun pelaaja rekisteröityy peliin ottaessaan ensimmäistä kertaa yhteyttä palvelimeen, hän syöttää nimensä. Yksityiskohtaisempaa pelihahmon luontia peli ei välttämättä tarvitse. Jos pelaajan puhelimessa on Bluetooth ja se on päällä, laitteen osoite haetaan ja tallennetaan automaattisesti.

Pelin alussa pelaajalla on puhelimessaan muutama yksikkö. Puhelin toimii pelaajan tukikohtana, jota toiset pelaajat eivät voi vallata.

Viivakoodin lukeminen. Kun pelaaja lukee viivakoodin tai syöttää sen numerosarjana, peli esittää näkymän, jonka sisältö esittää yhtä kolmesta vaihtoehdosta:

— tuote on vapaa
— tuote on yhden tai useamman muun pelaajan omistuksessa eli sisältää näiden pelaajien yksiköitä
— pelaaja omistaa itse kyseisen tuotteen, mahdollisesti yhdessä yhden tai useamman muun pelaajan kanssa.

Lisäksi tuote voi sisältää yhden vapaan yksikön, jonka pelaaja voi ottaa omistukseensa.

Yksiköiden liikuttaminen. Yksiköiden liikuttamiseen on kaksi erillistä operaatiota: siirto ja pudotus. Siirto tarkoittaa yksikön siirtämistä yhdestä pelaajan omistamasta kohteesta toiseen. Yksiköitä voi siirrellä vapaasti milloin tahansa. Lähtöpisteenä tai kohteena olevaa viivakoodia ei tarvitse lukea. Pudotus tarkoittaa yritystä sijoittaa yksiköitä tuotteeseen, jota pelaaja ei valmiiksi omista. Jos tuote on vapaa, pudotus onnistuu aina ja pelaaja saa tuotteen itselleen. Jos tuote ei ole vapaa, käydään taistelu. Pelaaja voi valita pudotettaviksi mitkä tahansa yksiköt riippumatta niiden sijainneista. Pelaajan ei siten tarvitse esimerkiksi siirtää yksiköitä puhelimeensa ennen pudotusta. Jos siirto tai pudotus jättää jonkin tuotteen tyhjäksi, pelaaja menettää välittömästi sen omistajuuden.

Taistelu. Taistelussa on kaksi osapuolta, puolustaja ja hyökkääjä, joista kumpikin voi koostua yhden tai useamman pelaajan yksiköistä. Jos hyökkääjä voittaa, puolustajien yksiköt siirretään omistajiensa puhelimiin. Jos puolustaja voittaa, pudotetut yksiköt palautetaan alkuperäisiin kohteisiinsa. Tarkempi taistelujärjestelmä suunnitellaan myöhemmin.

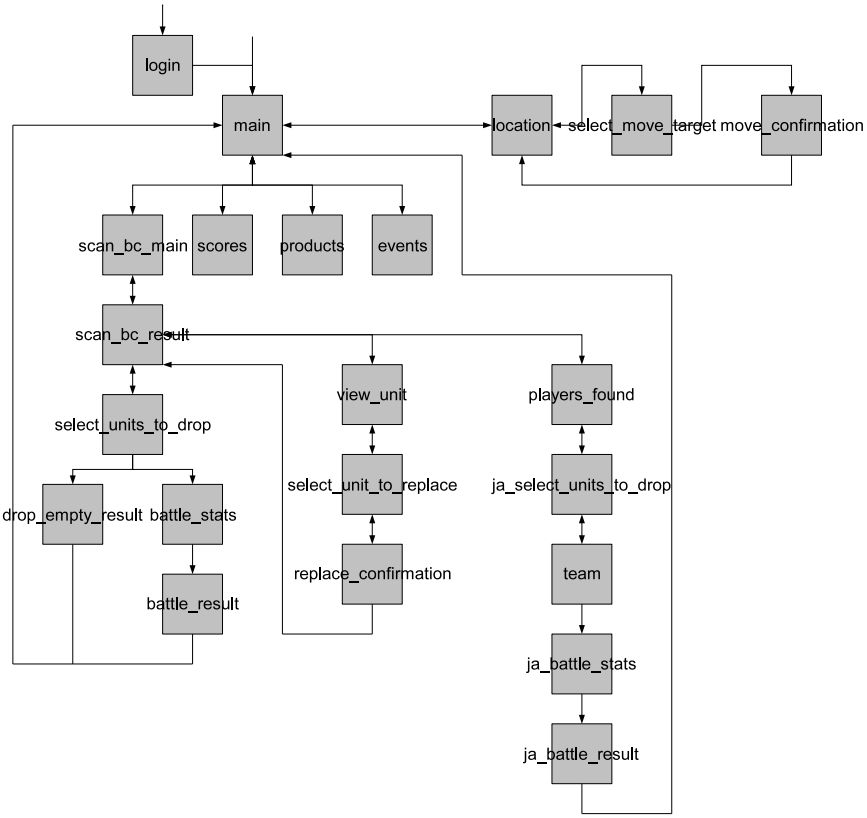
Yksiköt. Yksiköiden tyypit, ulkonäkö ja erityisominaisuudet suunnitellaan myöhemmin. Yksiköt vanhenevat ja poistuvat tietyn iän saavutettuaan pelistä, joten pelaajien on etsittävä jatkuvasti uusia yksiköitä. Vanheneminen on sidottu reaaliaikaan.

4.5.5 Interaktiokaaviot

Käyttöliittymän navigointikaavio on esitetty kuvassa 10. Kun pelaaja ottaa ensimmäistä kertaa yhteyden pelipalvelimeen, hänelle näytetään sisäänkirjautumisnäkymä (*login*). Myöhemmillä yhteydenottokerroilla pelaaja ohjataan suoraan päänäkymään (*main*), jossa pelaaja näkee omistamansa tuotteet ja mahdollisia muita keskeisiä tietoja. Tarkempia tietoja pelitilanteesta on kolmessa erillisessä näkymässä: *scores* on high score -lista eniten pisteitä keränneistä pelaajista, *products* esittää arvokkaimmat tuotteet ja *events* esittää lokin tuoreimmista pelitapahtumista (esim. viimeksi vallatut tuotteet). Kullakin pelaajan omistamalla tuotteella sekä pelaajan puhelimella on oma *location*-näkymä, josta pelaaja näkee kohteeseen sijoitetut yksiköt. Yksiköitä on mahdollista siirrellä pelaajan omien kohteiden välillä (*select_move_target*, *move_confirmation*).

Viivakoodin lukeminen tapahtuu näkymän *scan_bc_main* kautta. Näkymä *scan_bc_result* esittää luetun tuotteen sisällön. Koodista riippuen tästä näkymästä on tavallisesti pääsy joihinkin kolmesta toiminnosta: tavallinen yksiköiden pudottaminen (*select_units_to_drop* ja sitä seuraavat näkymät), tuotteesta löytyneen yksikön poimiminen (*view_unit* ja sitä seuraavat näkymät) sekä yksiköiden pudottaminen monen pelaajan yhteishyökkäyksessä (*players_found* ja sitä seuraavat näkymät).

Tavallisessa pudotuksessa pelaaja valitsee pudotettavat yksiköt (*select_units_to_drop*), minkä jälkeen hänelle esitetään joko vahvistus tyhjän kohteen valtauksesta (*drop_empty_result*) tai pudotuksesta seuranneen taistelun tiedot, jotka on jaettu kahteen näkymään (*battle_stats*, *battle_result*). Jos tuotteesta löytyi vapaa yksikkö, pelaaja voi katsella sen tarkempia tietoja ja halutessaan ottaa sen omakseen. Kaaviossa on esitetty ainoastaan tapaus, jossa yksiköiden maksimimäärä on jo täynnä, jolloin pelaajan on valittava, mikä nykyisistä yksiköistä korvataan uudella (*view_unit*, *select_unit_to_replace*, *replace_confirmation*). Viimeinen toiminto, monen pelaajan yhteishyökkäys, on hahmoteltu edellä kuvassa 9.



Kuva 10: Käyttöliittymän navigointikaavio.

5 Toteutus

Kuten edellisessä luvussa on kuvattu, pelin toisen prototyypin pohjalta kehitettiin pääasiassa kesä–heinäkuussa ensimmäinen osittain pelattava versio. Tässä luvussa kuvataan tämän version toteutusratkaisut. Luvun lopussa esitetään numeerista tietoa työmääristä ja itse ohjelmistosta.

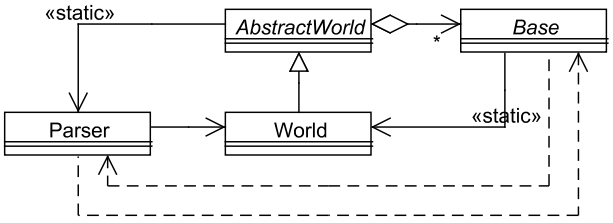
5.1 Arkkitehtuuri

Palvelin pyrittiin suunnittelemaan mahdollisimman suurelta osin yksikkötestattavaksi, mikä vaikutti sen yleiseen rakenteeseen. Yksikkötestauksella tarkoitetaan tässä erityisesti automaattisia testejä, jotka on kirjoitettu jonkin valmiin testauskehyksen, tässä tapauksessa JUnitin (`http://junit.org/`), avulla. Yksikkötestauksessa ongelmia aiheuttavat usein riippuvuudet valmiisiin kirjastoihin ja kehyksiin, jotka riippuvat edelleen esimerkiksi tiedostojärjestelmästä, tietokannasta tai verkkoyhteyksistä. Näin on erityisesti, jos kirjastoa tai kehystä ei ole suunniteltu tukemaan yksikkötestausta. Ongelmiin on useita ratkaisuja, joista osa on yleiskäyttöisempiä ja osa kehitetty tiettyä alustaa varten [7].

MUPE-palvelin ei juuri riipu ympäristöstään (palvelimessa ei esimerkiksi ole riippuvuuksia MUPE Coreen), mutta sitä ei toisaalta ole erityisesti suunniteltu yksikkötestattavaksi luokkatasolla. Tärkeimpien luokkien välillä on monia riippuvuuksia, kuten kuvassa 11 on esitetty. Suurin osa MUPE-sovelluksen toiminnallisuudesta sijaitsee tavallisesti sisältöluokissa eli abstraktin *Base*-luokan jälkeläisissä. *Base* taas käyttää luokan *Parser* palveluja käsitellessään komentojonokieltä sisältäviä tiedostoja. Lisäksi *Base* käyttää *World*-luokkaa mm. muodostimessaan lisätäkseen itsensä palvelimen kirjanpitoon. Käytännön seuraus näistä riippuvuuksista on, että yhdenkin sisältöolion luominen vaatii koko palvelimen luomista.

Ratkaisumalleja Collect Me -palvelimen yksikkötestaukseen voidaan hahmotella ainakin viisi. Samat vaihtoehdot ovat käytettävissä mille tahansa MUPE-sovellukselle.

1. *Parser*-, *World*- ym. tarvittavien olioiden luominen aina, kun sisältöluokkaa testataan.



Kuva 11: Tärkeimpien MUPE-palvelimen luokkien keskinäiset riippuvuudet.

Edut: Sisältöluokkiin sijoitettua toiminnallisuutta voidaan testata.

Haitat: Kaikkien palvelimen sisäisten riippuvuuksien vaikutusta testien kirjoittamiseen on vaikea arvioida etukäteen. Eräs esimerkki palvelimen sisäisen toteutuksen aiheuttamasta ongelmasta on, että *Base*-luokan stattinen viittaus palvelimen *World*-olioon ei päivity, vaikka ohjelmassa luotaisiin kokonaan uusi *World* ja uudet sisältöoliot. Jos tätä ei ole otettu huomioon, testit saattavat käyttäytyä oudosti, koska myöhemmissä testitapauksissa osa palvelimesta käyttää edelleen ensimmäisessä testitapauksessa luotua maailman tilaa.

2. Valmiin palvelimen muuttaminen siten, että yksittäistä sisältöluokkaa voidaan testata erillään muusta palvelimesta.

   Edut: Sisältöluokkiin sijoitettua toiminnallisuutta voidaan testata ottamatta huomioon valmiin palvelimen sisäistä rakennetta.

   Haitat: Kun MUPEn palvelimesta ilmestyy uusi versio, muutokset pitäisi integroida uudestaan. Lisäksi sisältöolioiden käsittely täysin erillisinä vaatisi ilmeisesti palvelimeen varsin suuria muutoksia. Tämä vaihtoehto ei ole mielekäs yksittäisessä sovellusprojektissa.

3. Riippuvuuksien korvaaminen erityisillä työkaluilla tai kääntämällä palvelimesta erityisiä testiversioita, joissa MUPE-luokkia on korvattu sopivilla tyngillä. Riippuvuuksien korvaamiseen voitaisiin ehkä käyttää esimerkiksi aspect-oriented programming -menetelmälle kehitettyjä työkaluja kuten Javan AspectJ:tä (`http://www.eclipse.org/aspectj/`).

   Edut: Testiympäristöä olisi luultavasti mahdollista kontrolloida varsin tarkasti.

   Haitat: AspectJ:n kaltaisista työkaluista ei ollut kokemusta. Työkaluihin tu-

tustuminen ja testitapausten toteuttaminen olisi saattanut vaatia enemmän aikaa kuin muissa ratkaisuissa.

4. Tärkeimpien, monimutkaista logiikkaa sisältävien toimintojen sijoittaminen luokkiin, jotka eivät ole *Basen* jälkeläisiä ja joita voidaan testata erillisinä. Nämä luokat saisivat tarvitsemansa tiedot joko metodien parametreina tai käyttäen rajapintoja, jotka eristävät ne sisältöluokista.

   Edut: Toiminnallisuutta voidaan testata kokonaan erillään valmiista palvelimesta.

   Haitat: Testien kattavuus saattaa jäädä pieneksi, tai testattavien luokkien eristäminen MUPEsta saattaa osoittautua monimutkaiseksi.

5. Kaiken pelilogiikan sijoittaminen erilliseen MUPEsta riippumattomaan sovellusmoottoriin, jota muu palvelin käyttää selkeästi määritellyn ohjelmointirajapinnan kautta.

   Edut: Sovelluslogiikan toteutus voidaan suunnitella vapaasti MUPEsta riippumatta. Teoriassa sovellusmoottoria voisi käyttää myös jokin muu käyttöliittymä, mutta tässä projektissa sellaista ei ollut suunnitelmissa.

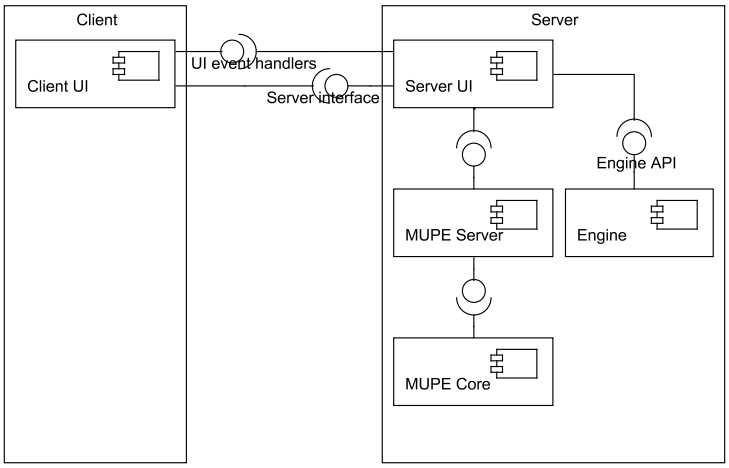
   Haitat: Collect Men tarvitsema pelilogiikka ei ole kovin mutkikasta, joten se erottaminen omaksi modulikseen ei välttämättä ole vaivan arvoista.

Kaikkein vaihtoehtojen perusteelliseen arviointiin ei ollut aikaa, mutta viimeinen ratkaisu vaikutti houkuttelevimmalta jo varhaisessa vaiheessa. Myös vaihtoehdot 1 ja 4 tai niiden yhdistelmä vaikuttivat mahdollisilta. Maaliskuusta alkaen pieniä osia pelin toiminnallisuudesta toteutettiin vaihtoehdon 5 mukaisesti, ja koska suuria ongelmia ratkaisulle ei näyttänyt ilmenevän, erillinen sovellusmoottori päätyi lopulliseksi ratkaisuksi.

Toinen yleiseen rakenteeseen vaikuttanut ratkaisu koski käyttöliittymän käsittelyä palvelimesta käsin. Kun jokin käyttöliittymän osa vaatii päivittämistä, palvelin voi muokata sitä suoraan lähettämällä tarvittavat viestit muutettaville käyttöliittymäelementeille. Esimerkiksi pelaajan pistemäärän muuttuessa palvelin voi muuttaa lukemaa esittävän *string*-elementin sisällön lähettämällä elementille itselleen viestin. Toinen vaihtoehto on määritellä käyttöliittymäpohjalle tai jollekin yksittäiselle elementille loogista päivitysoperaatiota edustava tapahtumankäsittelijä, joka huolehtii näkymän päivittämisen yksityiskohdista. Palvelin voi kutsua

tällaista itse määriteltyä tapahtumankäsittelijää lähettämällä viestin, jonka parametreina välitetään tarvittavat tiedot. Esimerkiksi pistemäärän muutoksesta huolehtiva tapahtumankäsittelijä voi ottaa parametrina uuden pistemäärän. Ainakin periaatteessa jälkimmäisen ratkaisun etuna on, että esitystavan yksityiskohtien muutokset eivät vaikuta palvelimeen. Jos pistemäärä halutaan esittää tekstin sijaan tai sen ohella ikonina, riittää muutoksen tekeminen tapahtumankäsittelijään. Palvelimen erottaminen ulkoasun yksityiskohdista on kuitenkin mahdollista saada aikaan myös muilla keinoin.

Tapahtumankäsittelijöihin perustuva malli vaikutti selkeimmältä, joten se valittiin ratkaisuksi. Kun käyttöliittymän päivitykset tapahtuvat ensisijaisesti tapahtumankäsittelijöitä kutsumalla, asiakasohjelmassa toimivan käyttöliittymän voidaan ajatella tarjoavan eräänlaisen rajapinnan palvelimelle. Koko sovelluksen rakenne voidaan näin ollen esittää kolmen rajapinnan ja niiden toteutusten avulla periaatekuvan 12 mukaisesti.



Kuva 12: Periaatekuva sovelluksen osista ja rajapinnoista.

Kuvan komponentit ovat seuraavat:

Client UI Asiakasohjelman muistissa oleva MUPEn komentojonokielellä määritelty käyttöliittymä. Huolehtii ruudulla näkyvän käyttöliittymän käsittelyn yksityiskohdista.

Server UI Palvelimen käyttöliittymälogiikka, joka on toteutettu MUPE Serverin

laajennuksena. Huolehtii käyttöliittymien koostamisesta, niiden lähettämisestä asiakasohjelmille ja niiden päivittämisestä korkealla tasolla.

MUPE Server MUPE-alustaan kuuluva palvelin.

Engine Sovelluksen MUPE-riippumaton osa, joka sisältää pelin tilan ja kaiken pelilogiikan.

MUPE Core MUPE-alustan välitason ohjelmisto.

Sovelluskohtaiset rajapinnat ovat:

UI event handlers Sovelluskohtaiset tapahtumankäsittelijät, joita Server UI kutsuu lähettämällä komentojonokielisiä viestejä asiakasohjelmalle.

Server interface Ne palvelimen sisältöolioiden Java-metodit, jotka on määritelty näkyviksi asiakasohjelmalle.

Engine API Java-luokat ja -rajapinnat, joiden kautta Server UI käyttää sovellusmoottoria.

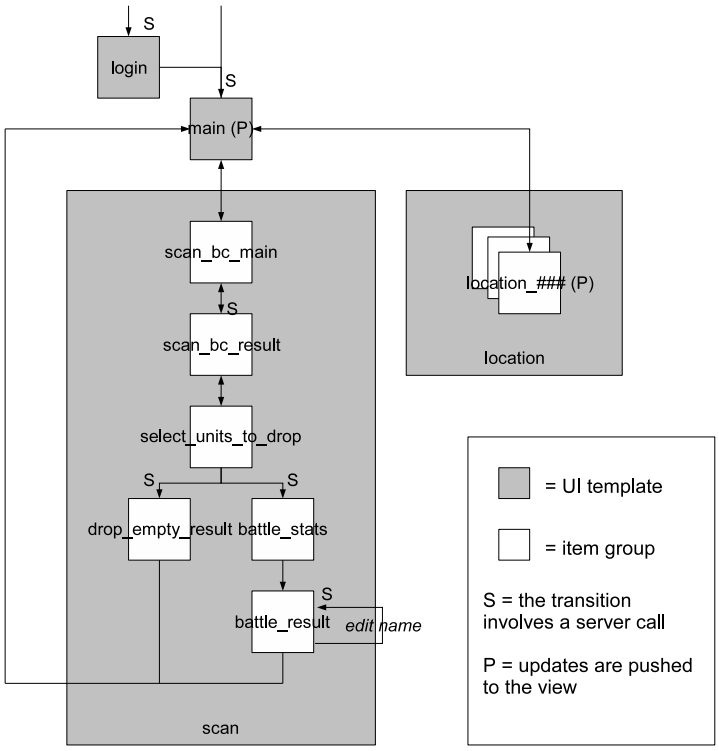
Seuraavaksi kuvataan kukin kolmesta rajapinnasta toteutuksineen alkaen asiakasohjelmassa toimivasta käyttöliittymästä. Kuvauksista on jätetty pois kaikki ohjelmiston sellaiset osat, jotka olivat ensimmäisessä pelattavassa versiossa vielä kesken eivätkä vaikuttaneet pelaajalle näkyvään toiminnallisuuteen.

5.2 Käyttöliittymä asiakasohjelmassa

Käyttöliittymän toteuttamisessa merkittävin käytännön rajoitus oli käytettävissä olleiden käyttöliittymäpohjien määrä. Tavallisesti rajana on ainoastaan muistin riittävyys, mutta testilaitteena toimineen Nokia 6600 -puhelimen MIDP-toteutuksessa on virhe, jonka kiertämiseksi MUPEn asiakasohjelma joutuu asettamaan käyttöliittymäpohjien määrälle kiinteän ylärajan. Toteutukseen käytettiin alustan versiota 2.30, jossa on käytettävissä kymmenen canvas-tyyppistä käyttöliittymäpohjaa. Koska peliin oli suunniteltu 22 eri näkymää (kuva 10), näkymät oli toteutettava jotenkin muuten kuin tekemällä jokaiselle oma käyttöliittymäpohja. Vaikka kohdealusta olisi ollut jokin muu, suuri määrä käyttöliittymäpohjia olisi

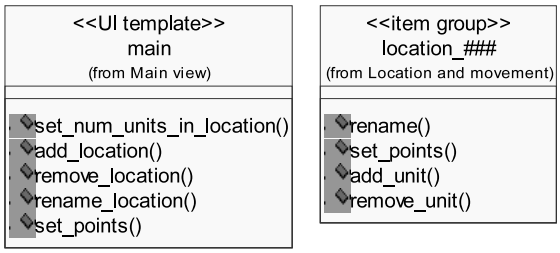
ilmeisesti ollut joka tapauksessa ongelma muistinkulutuksen kannalta [14]. Ratkaisuna useimmat näkymät toteutettiin koko ruudun kokoisina elementtiryhminä (*item group*), jotka sisälsivät kyseisen näkymän käyttöliittymäelementit. Loogisesti yhteen kuuluvat ryhmät koottiin saman käyttöliittymäpohjan sisälle.

Näkymien toteutuksen lisäksi oli suunniteltava käyttöliittymän dynaamisten osien päivitysstrategia. Syntynyt suunnitelma on esitetty toteutettujen näkymien osalta kuvassa 13. Kuvaan on merkitty kaikki palvelimelle tehtävät kutsut (S) sekä näkymät, joille palvelin lähettää push-päivityksiä (P). Päänäkymän ja kohdetta (puhelinta tai tuotetta) edustavan näkymän tapahtumankäsittelijät, joiden kautta näkymiä luomisen jälkeen päivitetään, on esitetty kuvassa 14.



Kuva 13: Käyttöliittymän navigointikaavio, johon on lisätty toteutukseen liittyvät tiedot.

Näkymien toteuttaminen elementtiryhmien avulla tuotti jonkin verran enemmän työtä kuin käyttöliittymäpohjien käyttö, joka oli ollut aikaisemmassa prototyypis-



Kuva 14: Käyttöliittymän tapahtumankäsittelijät, joita kutsutaan palvelimesta.

sä käytetty ratkaisu. Näkymän vaihtamisessa tarvitaan pitempiä komentojonoja, jotta halutut tapahtumankäsittelijät yhdistetään oikeisiin näppäimenpainalluksiin ja oikeat käyttöliittymäelementit tulevat näkyviin ja etualalle. Alustaversiossa 2.30 elementtiryhmien käsittelyssä oli lisäksi fokusoidun elementin käsittelyyn liittyvä virhe, joka esti joissain tapauksissa käyttöliittymän käytön. Ongelma pystyttiin kuitenkin kiertämään. Selvittelyä vaati myös kuvien lataaminen verkosta, mikä käytetyssä testilaitteessa toimi ainoastaan joissain tapauksissa. Tämäkin ongelma oli mahdollista ohittaa, joten käyttöliittymä saatiin lopulta toimimaan halutulla tavalla myös laitteella.

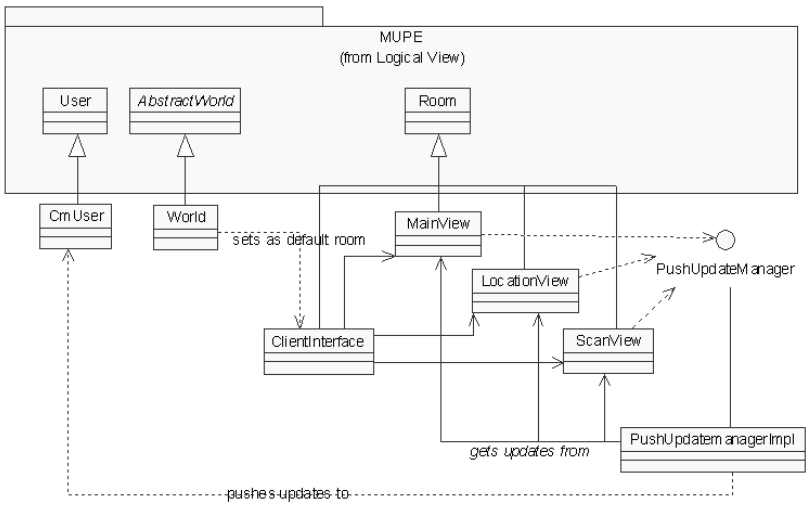
5.3 Käyttöliittymälogiikka palvelimessa

Server UI -modulin luokkarakenne on esitetty kuvassa 15. Luokkien keskinäisiä riippuvuuksia on jätetty esittämättä selkeyden vuoksi. Luokat ovat seuraavat:

CmUser Edustaa pelaajaa palvelimen MUPE-osassa. Luokkaan ei ole sijoitettu muuta tietoa kuin pelaajan nimi ja tunnistenumero.

World Luo muut palvelimen oliot. Ei merkittäviä laajennuksia valmiiseen *World*-luokkaan.

ClientInterface Kaikki asiakasohjelmille näkyvät metodit on koottu tähän luokkaan. Luokan ainoa instanssi asetetaan kutsumisen helpottamiseksi palvelimen ns. oletushuoneeksi, jolloin kutsumisessa käytettävä tunnistenumero on aina tiedossa. *ClientInterfacen* ainoa tehtävä on delegoida asiakasohjelmien pyynnöt oikealle näkymäluokalle.



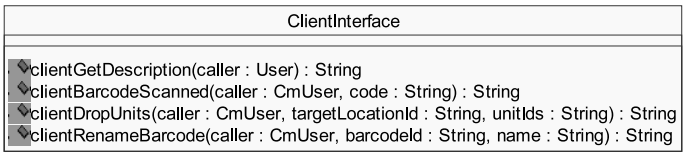
Kuva 15: Server UI -modulin luokkarakenne.

MainView, LocationView, ScanView Nämä kolme näkymäluokkaa vastaavat käyttöliittymäpohjia *main*, *location* ja *scan*. Kukin luokka koostaa käyttöliittymäpohjaan sisältyvät näkymät ja niiden päivittämiseksi lähetettävät viestit. Näkymäluokkien toiminnallisuus voitaisiin toteuttaa pelkkinä staattisina metodeina, mutta käytännön syistä kustakin luokasta luodaan yksi instanssi.

PushUpdateManagerImpl Näkymäluokat palauttavat omaa vastuualuettaan koskevat päivitykset suoraan kutsuvalle asiakasohjelmalle. Jos on tarpeen päivittää myös jonkin toisen näkymäluokan alueeseen kuuluvia näkymiä tai muiden asiakasohjelmien käyttöliittymiä, näkymäluokka kutsuu sopivaa metodia rajapinnassa *PushUpdateManager*. *PushUpdateManagerImpl* päättelee, mitä päivityksiä kullekin asiakasohjelmalle on lähetettävä. Tarvittavat viestit pyydetään näkymäluokilta itseltään ja lähetetään *User*-luokan push-toiminnon avulla asiakasohjelmille.

Aiemmassa navigaatiokaaviossa (kuva 13) esiintyneet S-symbolit vastaavat neljää eri palvelimen metodia luokassa *ClientInterface*. Metodit on esitetty kuvassa 16.

Metodien tehtävät ovat seuraavat:



Kuva 16: Palvelimen asiakasohjelmalle tarjoama rajapinta.

clientGetDescription Palauttaa pelin koko käyttöliittymän. Metodia kutsutaan, kun asiakasohjelma ottaa yhteyden palvelimeen. Kun uusi käyttäjä ottaa yhteyttä ensimmäistä kertaa, asiakasohjelmalle lähetetään ensin sisäänkirjautumisnäkymä, mutta tästä alusta huolehtii automaattisesti.

clientBarcodeScanned Palauttaa näkymän, joka kertoo luettua viivakoodia vastaavan tuotteen sisällön ja omistajan pelissä.

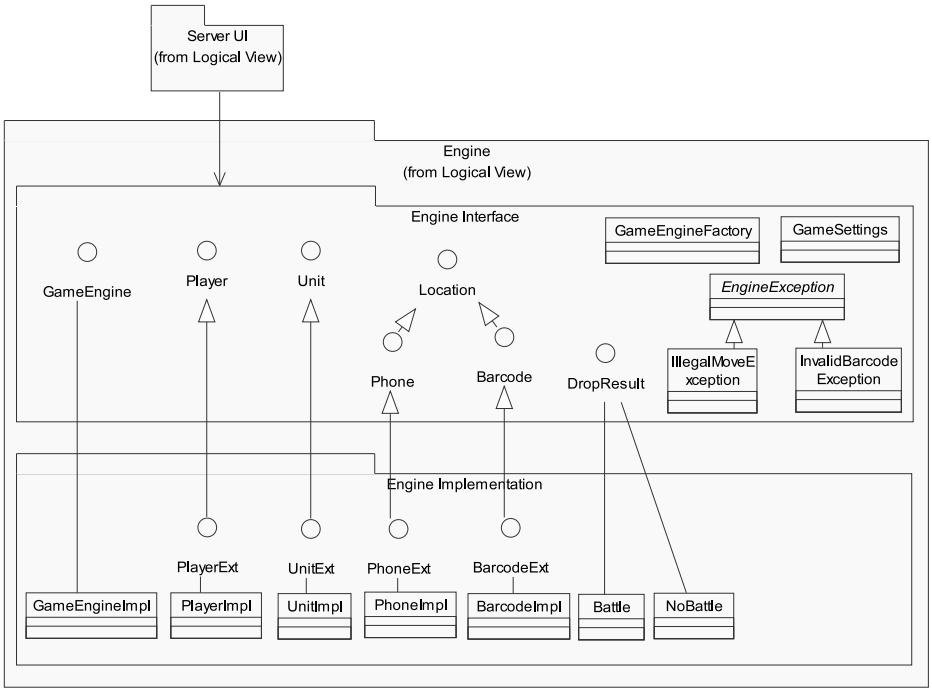
clientDropUnits Suorittaa yksiköiden pudottamisen tiettyyn tuotteeseen. Palvelin ratkaisee mahdollisen taistelun ja tuotteen omistajanvaihdokseen ja päivittää niiden pelaajien käyttöliittymät, joihin muutos vaikuttaa. Pudotuksen tehneelle pelaajalle näytetään lopputuloksesta kertova näkymä.

clientRenameBarcode Vaihtaa tuotteen nimen palvelimella.

5.4 Pelimoottori

Pelimoottorin ohjelmointirajapinta on esitetty kuvassa 17. Kun palvelin käynnistetään, pelimoottori luodaan kutsumalla luokan *GameEngineFactory* staattista metodia *create*, joka palauttaa toteutuksen *GameEngine*-rajapinnasta. Tämän rajapinnan kautta käyttöliittymälogiikka pyytää viittauksia muihin pelimoottorin olioihin tai pyytää pelimoottoria luomaan uusia olioita. Pelin käsitteitä edustavat luokat sijaitsevat pelimoottorissa. Ainoa poikkeus ovat itse pelaajat, joista on pidettävä kirjaa myös palvelimen MUPE-osassa (luokka *CmUser*). *Player*, *Unit*, *Location*, *Phone* ja *Barcode* sisältävät toiminnot pelin pelaamiseen ja käyttöliittymässä näytettävien tietojen hakemiseen. *DropResult* esittää tiedot lopputuloksesta, kun yksiköitä pudotetaan tuotteeseen. Jotta luokkia olisi mahdollista testata erillisinä ilman erikoisia työkaluja tai toimenpiteitä, pelimoottorin sisäiset

luokat riippuvat enimmäkseen rajapinnoista muiden konkreettisten luokkien sijaan. Tämän vuoksi vain palvelimen sisällä käytetyt toiminnot on sijoitettu omiin *Ext*-päätteisiin rajapintoihinsa.

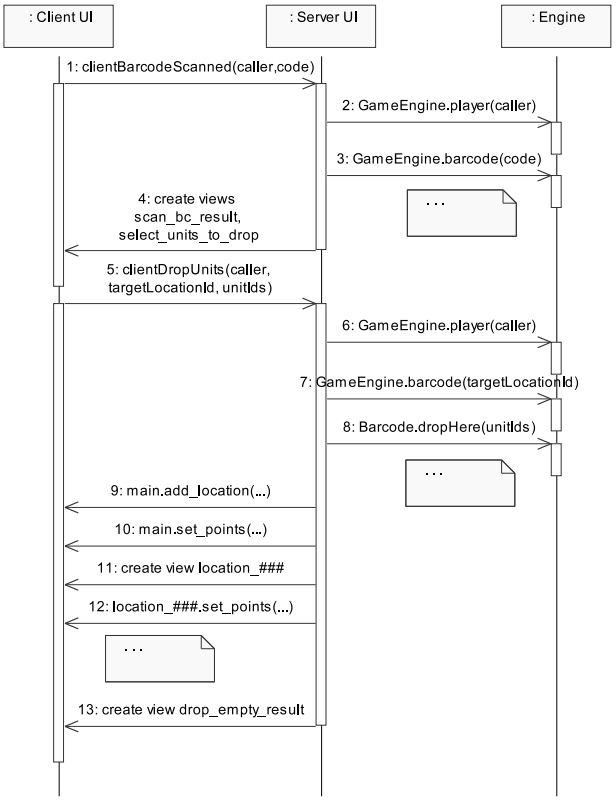


Kuva 17: Pelimoottorin Server UI -modulille tarjoama ohjelmointirajapinta ja osa toteutuksesta.

Pelimaailman olioihin on pystyttävä viittaamaan käyttöliittymän palvelimelle tekemissä kutsuissa. Tavanomaisemmassa MUPE-pohjaisessa sovelluksessa voidaan käyttää sisältöolioiden tunnisteita, jotka MUPE-palvelin luo automaattisesti. Collect Me -palvelimessa pelimoottorin on pidettävä itse kirjaa omista olioistaan ja varattava niille ainutkertaiset tunnisteet, joita voidaan välittää merkkijonomuotoisina palvelimen ja asiakasohjelman välillä. Pelimoottorissa on toisin sanoen toteutettava toiminnallisuutta, joka on jo valmiiksi toteutettu MUPE-alustassa. Tämä on valitun toteutuksen suurimpia ongelmia.

5.5 Dynaaminen toiminnallisuus

Kuvassa 18 on esitetty sekvenssikaaviona sovelluksen osien yhteistyö seuraavassa skenaariossa: Pelaaja on viivakoodinlukunäkymässä (*scan_bc_main*). Hän syöttää koodin ja valitsee ”OK”. Peli esittää näkymän *scan_bc_result*, jossa kerrotaan, että tuote on vapaa. Pelaaja valitsee ”Drop units”. Peli esittää näkymän *select_units_to_drop*, jossa pelaaja valitsee yhden tai useampia yksiköitä ja valitsee ”OK”. Peli esittää näkymän *drop_empty_result*, joka ilmoittaa pelaajan saaneen tuotteen haltuunsa. Pelaaja valitsee ”OK”. Peli esittää päänäkymän (*main*), jossa pelaaja näkee uuden tuotteensa muiden joukossa.



Kuva 18: Esimerkki sovelluksen osien yhteistyöstä. Pelaaja valtaa vapaan tuotteen syöttämällä viivakoodin ja pudottamalla yksiköitä.

Kuten aikaisemmasta kuvasta 13 voidaan päätellä, skenaarion aikana on tehtävä kaksi kutsua palvelimelle. Kutsuttavat metodit ovat *clientBarcodeScanned* ja *clientDropUnits*. Server UI -modulin tekemistä kutsuista pelimoottoriin ja käyttöliittymän tapahtumankäsittelijöihin on kaaviossa esitetty vain osa. Ensimmäisellä kutsulla (sekvenssikaavion viesti 1) käyttöliittymä ilmoittaa palvelimelle, minkä viivakoodin käyttäjä on syöttänyt. Server UI -moduli hakee pelaajaa ja viivakoodia vastaavat oliot pelimoottorista (viestit 2–3) ja hakee näiden kautta edelleen mm. pelaajan omistamat yksiköt. Jos viivakoodia ei ole aikaisemmin luettu, pelimoottori luo sitä edustavan olion tässä vaiheessa. Pelimoottorilta pyydettyjen tietojen avulla Server UI täydentää dynaamiset kohdat näkymien *scan_bc_result* ja *select_units_to_drop* komentojonokielisiin kuvauksiin ja palauttaa nämä näkymät kutsun paluuarvona (4).

Seuraava palvelinkutsu tapahtuu, kun pelaaja on valinnut yksiköt ja hyväksyy niiden pudotuksen koodiin. Palvelin ei säilytä tietoa siitä, mitä kukin käyttäjä on tekemässä, vaan tila on tallennettu käyttöliittymiin. Tässä tapauksessa tilaa edustavat luettu viivakoodi sekä pelaajan käytettävissä olevien yksiköiden tunnisteet. Nämä tunnisteet sisältyivät palvelimen palauttamaan näkymään *select_units_to_drop*. Kun pelaaja hyväksyy pudotuksen, viivakoodi ja pudotettaviksi valittujen yksiköiden tunnisteet lähetetään takaisin palvelimelle (5). Server UI noutaa jälleen pelimoottorilta tarvittavat tiedot ja käyttää luettua koodia edustavan *Barcode*-olion metodia *dropHere* tehdäkseen pudotuksen (6–8). Paluuarvo on *DropResult*-tyyppinen olio, jolta käyttöliittymälogiikka voi pyytää tarkemmat tiedot pudotuksen tuloksesta. Tämän jälkeen Server UI käyttää push-toimintoa päivittääkseen uuden tuotteen tiedot päänäkymään ja luodakseen tuotteelle uuden näkymän *location_###* (9–12). Näkymän nimessä *###* edustaa tuotteen tunnistetta, jona toimii viivakoodin koodinumero itse. Samalla tavoin päivitetään yksiköiden sijainnit pää- ja tuotenäkymiin. Lopuksi koostetaan ja palautetaan siirron tulokset ilmoittava näkymä (13).

Pelin lopullisessa toteutuksessa pelaajien käyttöliittymien päivittäminen olisi monimutkaisempaa, sillä huomioon olisi otettava tuotteen valtaamisen mahdolliset vaikutukset arvokkaimpien tuotteiden listaan sekä parhaiden pelaajien listaan. Jos tuotteesta käydään taistelu, on nykyisessäkin versiossa huomioitava toisen osapuolen yksiköiden mahdolliset siirtymät ja hänen pisteidensä muutos. Tämän päivityslogiikan suunnittelu, toteutus ja testaus tulevat todennäköisesti olemaan

suurimpia ongelmia pelin lopullisessa toteutuksessa.

5.6 Tulosten tarkastelua

Taulukossa 2 on verrattu Collect Men ohjelmiston kokoa MUPEn palvelinohjelmistoon ja kolmeen muuhun MUPE-pohjaiseen peliin. Koko on ilmoitettu koodiriveinä (LOC), jotka eivät ole tyhjiä eivätkä kommentteja. Luvut on mitattu Eclipse Metrics -työkalulla (`http://metrics.sourceforge.net/`). Collect Men lähdekoodiin sisältyi koko sovellus elokuun alussa, myös keskeneräiset osat, joita edellä ei ole kuvattu. MUPE Serverin lähdekoodi on otettu julkaisusta Full-Source 2.30, muista peleistä taas käytettiin MUPE-sivustolta [11] toukokuussa 2006 ladattuja versioita. Missään neljästä pelistä ei ole muokattu alustaan kuuluvaa MUPE Serveriä, joten sovelluskohtaisen ja alustaan kuuluvan palvelinkoodin erottaminen ei ole ongelma. Ancient Runes sisältää yleiskäyttöisemmän keräilykorttipeleille tarkoitetun pelimoottorin, joka on Collect Mestä poiketen toteutettu MUPEn avulla.

Taulukko 2: Alustaan kuuluvan palvelinohjelmiston, Collect Me -pelin ja kolmen muun MUPE-sovelluksen koko koodiriveissä.

Ohjelmisto	LOC
MUPE Server	2214
Collect Me	2317
Ancient Runes	2834
Constellations	3505
FirstStrike	1193

Collect Men palvelin on vertailun perusteella suunnilleen samaa kokoluokkaa kuin muillakin yksinkertaisilla, mutta ei-triviaaleilla MUPE-pohjaisilla peleillä. Lisäksi voidaan mainita, että lähdekoodi jakautuu noin puoliksi Server UI- ja Engine-modulien kesken.

MUPEn komentojonokielellä toteutetun käyttöliittymän koosta saadaan karkea arvio laskemalla koodia sisältävien tiedostojen yhteiskoko. Lopputulokseen vaikuttaa tällöin mm. kommenttien määrä ja se, onko usein toistuvia osia siirretty omaan tiedostoonsa, joka sisällytetään ajonaikaisesti muihin tiedostoihin. Koska komentojonot ovat sisällöltään enimmäkseen kuvailevia, mutta saattavat sisältää

myös ehto- ja toistorakenteita, tiedostojen kokoon vaikuttavat sekä käyttöliittymän laajuus että siihen liittyvän logiikan monimutkaisuus. Taulukossa 3 on verrattu samoja pelejä komentojonojen koon perusteella.

Taulukko 3: Käyttöliittymän määrittelevien XML-tiedostojen yhteiskoko Collect Me -pelissä ja kolmessa muussa MUPE-sovelluksessa.

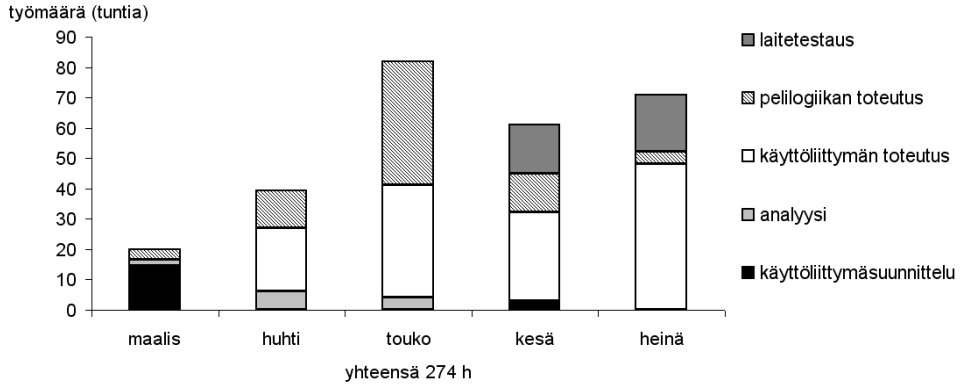
Ohjelmisto	kB
Collect Me	54
Ancient Runes	97
Constellations	199
FirstStrike	62

Collect Men käyttöliittymä on tämän perusteella huomattavan yksinkertainen, vaikka otetaankin huomioon, että vasta osa suunnitelluista näkymistä on toteutettu.

Koska suunnittelussa pyrittiin mahdollistamaan palvelimen yksikkötestaus, voidaan arvioida, missä määrin tässä onnistuttiin. Ensimmäisen pelattavan version valmistumisen aikaan djUnit -työkalu (`http://works.dgic.co.jp/djunit/`) ilmoitti pelimoottorin yksikkötestien rivikattavuudeksi 89 %. Erityisiä ongelmia testauksessa ei ilmennyt. Koska pelimoottori on kuitenkin vain noin puolet ohjelmistosta, eikä käyttöliittymälogiikalle ole kirjoitettu testejä, koko ohjelmiston testikattavuus jää alle viidenkymmenen prosentin. Valmiiksi testatun pelimoottorin olemassaolo vaikutti kuitenkin tekevän myös käyttöliittymälogiikan toteutuksesta helpompaa. Yksikkötestauksen tai erillisen pelimoottorin käytön kokonaisvaikutusta työmääriin tai ohjelmiston virheiden määrään ei kuitenkaan ole mahdollista arvioida.

Pelin toteutuksen työmäärästä on olemassa tehtävittäin ja kuukausittain jaoteltu kirjanpito, joka on esitetty kaaviona kuvassa 19. Työmäärään on laskettu pelin käyttöliittymän suunnitteluun ja varsinaiseen toteutustyöhön käytetty aika. Pelisuunnitteluun ja vastaaviin tehtäviin kuluneesta ajasta ei ole kirjanpitoa. Tehtävien merkitykset ovat seuraavat. 1) Käyttöliittymäsuunnitteluun on laskettu ensimmäisen prototyypin eli käyttöliittymäluonnosten tekeminen sekä muu vastaava suunnittelutyö. 2) Analyysiin on laskettu ohjelmiston korkean tason mallintaminen, johon käytettiin mm. käyttötapauskaavioita. 3) Käyttöliittymän toteu-

tukseen on laskettu sekä itse käyttöliittymän että palvelimen Server UI -modulin ohjelmistosuunnittelu ja toteutus. 4) Pelilogiikan toteutukseen on laskettu Engine-modulin suunnittelu ja toteutus. 5) Laitetestaukseen on laskettu kaikki työ, joka liittyi pelin kokeilemiseen matkapuhelimella. Suuri osa tästä ajasta kului todennäköisesti aiemmin mainittujen kuvanlatausongelmien selvittämiseen.



Kuva 19: Collect Men prototyyppien ja ensimmäisen pelattavan version toteutukseen käytetty aika eriteltynä kuukauden ja tehtävän mukaan.

Toukokuussa valmistuneessa toisessa prototyypissä suurin osa käyttöliittymästä oli jo toteutettu myöhempää versiota muistuttavalla tavalla (vertaa liitteen 1 esimerkkikuvia kuvaan 8 sivulla 28). Myös osa näkymien dynaamisia osia päivittävästä toiminnallisuudesta ja suuri osa pelilogiikasta oli toteutettu. Kaaviosta kuitenkin nähdään, että käyttöliittymän parissa kului vielä huomattavasti aikaa tämän jälkeen. Suuri osa ajasta kului ilmeisesti käyttöliittymän osien toteuttamiseen uudelleen eri tavalla. Tähän kuului mm. näkymien toteuttaminen elementtiryhmien avulla ja niihin liittyvien ongelmien kiertäminen sekä käyttöliittymien päivitysstrategian tarkempi suunnittelu.

6 Johtopäätökset

Diplomityössä pyrittiin kehittämään matkapuhelimilla pelattava peli, joka käyttäisi Multi-User Publishing Environment (MUPE) -ohjelmistoalustan kontekstitietoisuutta tukevia ominaisuuksia. Lähtökohtana oli idea, jossa viivakoodeja käytetään pelivälineinä.

Työ jakautui pelisuunnitteluun ja toteutukseen. Pelisuunnittelun tulos oli alustava suunnitelma peliksi, jossa käytetään kahta kontekstitietoisuuteen liittyvää tekniikkaa: viivakoodin lukemista puhelimen kameralla ja muiden Bluetoothilla varustettujen puhelinten havaitsemista. Jälkimmäinen ominaisuus on jo käytettävissä MUPE-alustassa, edellinen vasta suunnitteilla. Kumpikin toiminto sijoittuu käyttäjän päätelaitteeseen, josta tarvittavat tiedot siirretään sovelluslogiikan sisältävälle palvelimelle. Suunnitellussa pelissä valituilla tekniikoilla on mielekkäät tehtävät, eikä peli tiettävästi muistuta liian suoraan mitään aikaisempaa peliä, joten suunnitelma täyttää tärkeimmät sille asetetut vaatimukset. Peli-idean vahvuus on persistentin pelimaailman ja kaikkialla läsnä olevien, tavallisella kamerapuhelimella luettavissa olevien tunnisteiden yhdistäminen. Peli on pelattavissa missä vain, eikä pelaajamäärällä ole periaatteessa ylärajaa. Sen sijaan käytettyjen tekniikoiden kannalta peli ei tarjoa juuri uutta, sillä viivakoodinlukijoita ja Bluetoothia on aikaisemminkin käytetty peleissä. Yritykset hyödyntää pelissä yleisempää reaalimaailman tietoa eivät selvinneet ideointivaihetta pitemmälle.

Pelin suunnittelua on toistaiseksi tuettu yleisen avun ohella kahdella muodollisemmalla arvioinnilla. Arviointi heurististen suunnitteluohjeiden pohjalta antoi palautetta erityisesti käyttöliittymäsuunnittelun tueksi, mutta suuria käytännön parannuksia palautteen pohjalta ei pystytty tekemään. Työssä ei siten juurikaan toteutunut se ajatus, että kyseiset heuristiset ohjeet auttaisivat erityisesti opiskelijoita ja muita kokemattomia mobiilipelien kehittäjiä [15]. Kaksi fokusryhmähaastattelua antoi palautetta itse peli-ideasta ulkopuolisten silmin ja tuotti uusia ideoita muun muassa pelin mahdollisista ominaisuuksista. Haastatteluista saadut mielipiteet ovat toistaiseksi vaikuttaneet muun muassa päätökseen poistaa pelistä erilaisiin reaalimaailman tietoihin perustuva satunnaisvaihtelu. Koska haastattelut tuottivat melko suuren määrän aineistoa, oli olennaista, että pelistä oli selkeä näkemys, johon uudet ideat ja mielipiteet oli mahdollista suhteuttaa. Tässä Mo-MUPE-projektin tarjoama tuki osoittautui tärkeäksi.

Pelistä toteutettiin versio, jossa osa suunnitellusta toiminnallisuudesta on kokeiltavissa. MUPE-alustan kontekstitietoisuutta tukevia ominaisuuksia ei pelissä vielä käytetä. Tiedossa ei ole ylitsepääsemättömiä esteitä toteutetun version laajentamiselle lopulliseksi peliksi, joten myös toteutuksen tavoitteet saavutettiin. Perustoiminnallisuuden toteuttamiseen kului kuitenkin enemmän aikaa kuin toivottiin, ja lopullisen version kehittäminen viivästyy alkuperäisestä aikataulusta. Peliä ei ole vielä mahdollista pelata normaalisti, joten sen onnistumista pelinä ei ole mahdollista arvioida. Tyypillisestä MUPE-sovelluksesta poiketen pelin palvelin toteutettiin käyttöliittymäosana ja MUPEsta riippumattomana pelimoottorina. Ratkaisun toteutukselle ei ilmennyt esteitä, mutta sen hyödyllisyys tämänkokoisessa sovelluksessa on epävarmaa.

Pelin jatkokehityksessä on tarkoitus mahdollisuuksien mukaan korvata nykyinen ulkoasu ammattimaisesti suunnitellulla käyttöliittymällä ja grafiikalla. Pelisuunnittelua pitää tarkentaa muun muassa suunnittelemalla taistelujärjestelmä ja yksiköiden ominaisuudet. Pelattavuuden parantaminen voidaan aloittaa, kun riittävän suuri osa toiminnallisuudesta on valmiina.

Lähteet

[1] Björk, Staffan & Holopainen, Jussi. *Patterns in Game Design.* Hingham, Massachusetts: Charles River Media, cop. 2005. ISBN 1-58450-354-8.

[2] Davidsson, Ola & Peitz, Johan & Björk, Staffan. Game design patterns for mobile games [verkkodokumentti]. Project report to Nokia Research Center, Finland. 2004 [viitattu 20.7.2006]. Saatavissa: `http://procyon.lunarpages.com/~gamed3/docs/Game_Design_Patterns_for_Mobile_Games.pdf`.

[3] Dey, Anind K. *Providing Architectural Support for Building Context-Aware Applications.* Väitöskirja, Georgia Institute of Technology, College of Computing, 2000.

[4] Ellonen, Hanna-Kaisa & Cavén, Outi. Focus group -ryhmähaastattelu. Teoksessa: *E-demokratian ja elämysten arkea.* Lappeenranta: Telecom Business Research Center Lappeenranta, Lappeenranta University of Technology, 2003. s. 63–72. (TBRC Working Papers 18.) ISBN 951-764-848-0. ISSN 1456-9132. ISBN 951-764-859-9 (URL: `http://www.lut.fi/TBRC`).

[5] Hansen, Dale. Barcode Battler : console information [verkkodokumentti]. [Viitattu 11.8.2006.] Saatavissa: `http://www.consoledatabase.com/consoleinfo/barcodebattler/`.

[6] Koivisto, Elina M. I. & Korhonen, Hannu. Mobile game playability heuristics [verkkodokumentti]. Version 1.0; March 17, 2006 [viitattu 27.4.2006]. Saatavissa: `http://www.forum.nokia.com/`.

[7] Massol, Vincent & Husted, Ted. *JUnit in Action.* Greenwich, Connecticut: Manning, cop. 2004. ISBN 1-930110-99-5.

[8] Mobile Context-Aware Applications and Games [verkkodokumentti]. Projektin kuvaus Teknologian kehittämiskeskuksen verkkosivuilla. [Viitattu 14.8.2006.] Saatavissa: `http://www.tekes.fi/fenix/` (> Projektit > Mobile Context-Aware Applications and Games).

[9] Mobile Entertainment Forum & Booz Allen Hamilton & Mobile Entertainment Analyst. Future mobile entertaiment scenarios [verkkodokumentti]. March 2003 [viitattu 22.2.2006].

Saatavissa: http://mobileentertainmentforum.org/pdf/MEF-WP-on-Future-ME-Scenarios.pdf.

[10] MoMUPEn projektisuunnitelma. Projektin sisäistä dokumentaatiota.

[11] Multi-User Publishing Environment (MUPE) [WWW-sivuston etusivu]. [Viitattu 24.8.2006.] Saatavissa: http://www.mupe.net/.

[12] MUPE client script manual [verkkodokumentti]. Last modified 5 June 2006 [viitattu 14.6.2006]. Saatavissa: http://www.mupe.net/wiki/index.php/Scriptmanual.

[13] Nielsen, Jakob & Molich, Rolf. Heuristic evaluation of user interfaces. Teoksessa: *Proceedings of the SIGCHI conference on Human factors in computing systems: Empowering people.* [CHI '90, Seattle, Washington.] New York: ACM Press, 1990. s. 249–256. ISBN 0-201-50932-6.

[14] Perälä, Harri & Räsänen, Eero. UI limit for Nokia 6600 [viestiketju]. Julkaisussa: *MUPEdev Forum* [online-keskustelupalsta]. 22.6.2006 [viitattu 12.7.2006]. Saatavissa: http://mupe.nrln.net/mupedev/index.php?name=PNphpBB2&file=viewtopic&t=164.

[15] Porras, Jari & Heikkinen, Kari & Koskinen, Kimmo & Suomela, Riku. Context-awareness in mobile games. Teoksessa: *Proceedings of the Second Workshop on Context Awareness for Proactive Systems CAPS 2006 : Kassel, Germany, June 12–13, 2006.* Kassel, Saksa: Kassel University Press, 2006. s. 19–30. ISBN 3-89958-210-1.

[16] Rouse, Richard. *Game Design: Theory & Practice.* 2nd ed. Plano, Texas: Wordware Publishing, cop. 2005. ISBN 1-55622-912-7 (pbk.).

[17] Skannerz instruction manual [verkkodokumentti]. [Viitattu 6.3.2006.] Saatavissa: http://www.radicagames.com/user_manuals/skannerz-71051.pdf.

[18] SNAP Mobile [WWW-sivuston etusivu]. [Viitattu 24.8.2006.] Saatavissa: http://snapmobile.nokia.com/.

[19] Suomela, Riku. *Constructing and examining location-based applications and their user interfaces by applying rapid software development and structural*

analysis. Väitöskirja, Tampereen teknillinen yliopisto, Tietotekniikan osasto. Tampere: Tampereen teknillinen yliopisto, 2006. (Julkaisu 601.) ISBN 952-15-1600-3 (printed), 952-15-1832-4 (PDF). ISSN 1459-2045.

[20] Suomela, Riku & Räsänen, Eero & Koivisto, Ari & Mattila, Jouka. Open-source game development with the Multi-User Publishing Environment (MUPE) application platform. Teoksessa: *Entertainment Computing - ICEC 2004 : Third International Conference. Eindhoven, The Netherlands, September 1-3, 2004. Proceedings.* Berlin: Springer, 2004. s. 308–320. (Lecture Notes in Computer Science 3166.) ISBN 3-540-22947-7.

[21] Suomela, Riku & Koskinen, Kimmo & Heikkinen, Kari. Rapid prototyping of location-based games with the Multi-User Publishing Environment application platform. Teoksessa: *IEE International Workshop on Intelligent Environments.* [Colchester, Iso-Britannia 29.6.2005.] Stevenage, UK: IEE, 2005. s. 143–151. ISBN 0863415199.

Liite 1. Fokusryhmähaastatteluissa käytetty esittelymateriaali

Collect Me

Overview: Players collect Martians and use them to fight over the control of barcodes. The more popular the code is, the more influence the player gets from controlling it. Popularity is determined by how many players have tried to take over the code, which means that well known products will become the most valuable targets.

Goal of the game: To be the player with most influence at the moment.

What contexts are used?

- Barcode scanning with phone camera
- Technologies for collecting Martians:
  - Bluetooth addresses
  - (2D tags, GPS location ...)
- Variables that modify the performance of Martians:
  - temperature in player's home city
  - air pressure in player's home city
  - NASDAQ index
  - Dow Jones index

New Martians are randomly generated to identifiers or locations (depending on the technologies used). When the player finds a new Martian, it can be added to his or her herd. The conditions (e.g., weather, stock market indices) at the time when the Martian was collected are stored. In the future the Martian will receive a penalty or a bonus depending on how similar or different the current conditions are to the original values.

To get control of a product, the player scans the barcode using the phone camera and sends one or more Martians to take over the code. If the code was already occupied, a battle ensues, and the winner gets to control the code.

(jatkuu)

Kysymykset

1. Miten paljon pelistä on mahdollista ymmärtää nykyisen prototyypin pohjalta? Ts. jos pelaaja tietää vain, että peli liittyy jotenkin viivakoodeihin, ja ryhtyy käyttämään nykyistä protoa, jääkö koko idea hämärän peittoon? Mihin erityisesti tarvitaan selvennystä?

2. Nykyisessä suunnnitelmassa yksiköt saavat bonuksia tai miinuksia nykyisen säätilan ja talouslukujen mukaan. Pelaaja voi vaikuttaa bonuksiin keräämällä uusia yksiköitä, jotka ovat sopeutuneet erilaisiin olosuhteisiin kuin aikaisemmat. Vaikuttaako tällainen reaalimaailmaan perustuva satunnaisvaihtelu pelin kannalta mielenkiintoiselta, haitalliselta vai samantekevältä?

3. Peliä tullaan mahdollisesti muuttamaan tähän prototyyppiin nähden siten, että yksiköitä ei enää keräillä erillisillä tekniikoilla (esim. Bluetooth-osoitteet, GPS), vaan uusia yksiköitä löytyy viivakoodeista itsestään. Tarkempaa suunnitelmaa ei ole olemassa, mutta kumpi kuulostaa ajatuksena paremmalta, viivakoodien ympärille keskittyminen vai eri tekniikoiden yhdistely?

(jatkuu)

Esimerkkikuvat

main

location

scan_bc_result

products