TAMPEREEN AMMATTIKORKEAKOULU
Tietotekniikka
Ohjelmistotekniikka

Tutkintotyö

Mikko Nyrönen

HERO QUEST S60-LAITTEELLE

Työn-ohjaaja Lehtori Tony Torp
Työn teettäjä Tampereen ammattikorkeakoulu, valvoja lehtori Tony Torp
Tampere 2008

TAMPEREEN AMMATTIKORKEAKOULU
Tietotekniikka
Ohjelmistotekniikka
Nyrönen, Mikko
Hero Quest S60-laitteelle
Tutkintotyö 42 sivua + 2 liitesivua
Työn ohjaaja Lehtori Tony Torp
Työn teettäjä Tampereen ammattikorkeakoulu
Toukokuu 2008
Hakusanat EUnit, S60, Symbian, TDD, Yksikkötestaus

TIIVISTELMÄ

Mobiililaitteille on olemassa pelisovelluksia, mutta useat niistä ovat tehty JAVA-ohjelmointikielellä. Halusin tehdä pelisovelluksen Symbian-kiellä ja samalla syventää Symbian-osaamistani.

Opinnäytetyön tarkoituksena on luoda pelisovellus ja käyttää pelin kehittämisessä Test Driven Developmenttia (TDD) joihinkin sovelluksen osa-alueisiin. TDD:n ohessa syntyvät yksikkötestit ajetaan Digia Oyj:n EUnit-työkalulla.

Selvitän työssäni myös sovelluksen rakennetta ja käytän sen kuvaamiseen UML-kaavioita.

Työn tavoitteena on syventää Symbian-osaamistani ja opetella käyttämään EUnit-työkalua yksikkötestien ajamiseen. Tutkintotyön tulos on ajaa EUnit-työkalulla onnistuneesti TDD-menetelmällä tuotettuja yksikkötestejä.

TAMPERE UNIVERSITY OF APPLIED SCIENCES
Computer science
Software engineer
Nyrönen, Mikko
Hero Quest for S60 device.
Thesis 42 pages + 2-appendixes
Thesis-Supervisor Tony Torp (lecturer)
Commissioning Company Tampere University of Applied Sciences
May 2008
Keywords EUnit, S60, Symbian, TDD, Unit testing

ABSTRACT

Most of the games that have been made to mobile devices are programmed with JAVA programming language. I wanted to create a game program with Symbian language and learn more about Symbian.

The purpose of this thesis is to create a game program and use Test Driven Development (TDD) as development method to some parts of the game. I also used EUnit-tool by Digia Plc, to run unit tests that were result of TDD.

In this thesis I also explain applications structure and use UML-diagrams to clarify it.

The goal of this thesis is to learn more about the Symbian, secondly learn how to use EUnit-tool and run unit tests with it. Result of this thesis is to run unit test successfully that are outcome from TDD.

SISÄLLYSLUETTELO

LYHENTEET JA TERMIT

EUnit	Digia Oyj:n tekemä yksikkötestaustyökalu
OS	Operating System eli käyttöjärjestelmä
Portata	tarkoittaa ohjelmakoodin muuntamista
S60	Series 60 on Nokian kehittämä sovellusalusta.
Symbian	mobiililaitteille suunnattu sovellusalusta
TDD	Test Driven Development eli Testivetoinen ohjelmistokehitys
UML	Unified Modeling Language on tapa kuvata ohjelmiston rakennetta.

1 JOHDANTO

Työn tehtävänä oli tuottaa pelisovellus, joka muistuttaa Hero Quest-lautapeliä. S60 versioiden 2 ja 3 väliset erot ovat sen verran vaikuttavia, että version 2 sovellukset eivät toimi versiossa 3. Työssä selvitellään, mitä tarvitaan, että peliohjelma saadaan toimimaan nykyisen S60:n versiossa 3 ja sitä uudemmissa alustoissa. Työstä ilmenee myös sovelluksen arkkitehtuuri sekä pääsääntöisesti sen toiminta. Työhön liittyvän sovelluksen muuntamisen ohella kehitin sovellusta ja parantelin vanhoja toimintoja.

Pelisovelluksen pohja oli jo olemassa, mutta se oli tehty S60:n versio 2:lle, joten se tarvitsi muuntaa ensin S60:n versioon 3 sopivaksi. Merkittävin ero S60:n versioiden välillä on ominaisuus nimeltä Platform Security, jolla tuodaan lisää turvaa mobiililaitteisiin ja pyritään välttämään haittasovellusten kehittäminen eri mobiililaitteille.

Joillekin sovelluksen osakokonaisuuksista ajettiin myös erilaisia yksikkötestejä, jotka syntyivät, kun sovellusta kehitettiin Test Driven Developmentin avulla. Testejä ajettiin EUnit-työkalulla.

Työn tavoitteena on syventää tietämystä Symbianista. Tavoitteena oli myös käyttää Test Driven Developmentia eli Testivetoisen ohjelmistokehitystä osana sovelluksen kehitystä ja opetella käyttämään EUnit-työkalua. TDD:llä tuotettujen yksikkötestien testipetinä toimi Digia Oyj:n tekemä EUnit-työkalu S60-laitteelle.

2 SYMBIAN

Luvussa kerrotaan hieman Symbianin historiaa ja mitä Symbianille kuuluu tällä hetkellä.

2.1 Symbianin historiaa

Symbian perustettiin kesäkuussa vuonna 1998 ja sen omistivat Nokia, Ericsson, Motorola ja Psion. Matsushita (Panasonic) liittyi joukkoon lisensoijana ja osakkeenomistajana vuonna 1999, jolloin Symbianille myönnettiin myös nimike ”Potentiaalisin pitkällä tähtäimellä”.

Vuonna 2000 Sony ja Sanyo lisensoivat Symbian OS:n ja ensimmäinen Symbian-puhelin tuotiin markkinoille. Tämä puhelin oli Ericsonin R380.

Vuonna 2001 Symbian OS:n lisensoi Fujitsu, ja ensimmäinen 2,5G Symbian OS puhelin Nokian 7650 julkaistiin. Myös Nokian 9210 Communicator tuli markkinoille. Vuonna 2002 NTT DoCoMo lanseeraa 3G FOMA-palvelut japanissa ja Fujitsun F2051-puhelimen. Sendo lisensoi Symbian OS:n ja Siemens ryhtyy Symbianin osakkeenomistajaksi. Symbian OS v7.0 3G puhelimille julkaistaan, ja Sony Ericsson liittyy Symbianin osakkeenomistajaksi ja lisensoi Symbian OS:n.

Vuonna 2003 valmistettiin ensimmäinen Symbian OS v7.0s:ää käyttävä puhelin Nokian 6600. Symbian OS 7.0s julkaistiin. Useita Symbian puhelimia julkaistiin, muun muassa FOMA F2102v ja F900i, Motorola A920 ja A925, Nokia 7700, Sendo X, Siemens SX1, sekä Sony Ericsson P900. BenQ tiedotti P30:stä, joka käytti UIQ:ta ja Samsung liittyi Symbianin osakkaaksi.

Vuonna 2004 Sir Peter Gershon liittyi Symbianin hallitukseen riippumattomana puheenjohtajana. Symbianin osakkaat ostivat Psionin osuuden osakkeista ja investoivat vielä lisäksi 50 miljoonaa puntaa Symbianiin. Sharp, Lenovo, Arima ja LG lisensoivat Symbian OS:n. Symbian OS v8.0 julkaistiin.

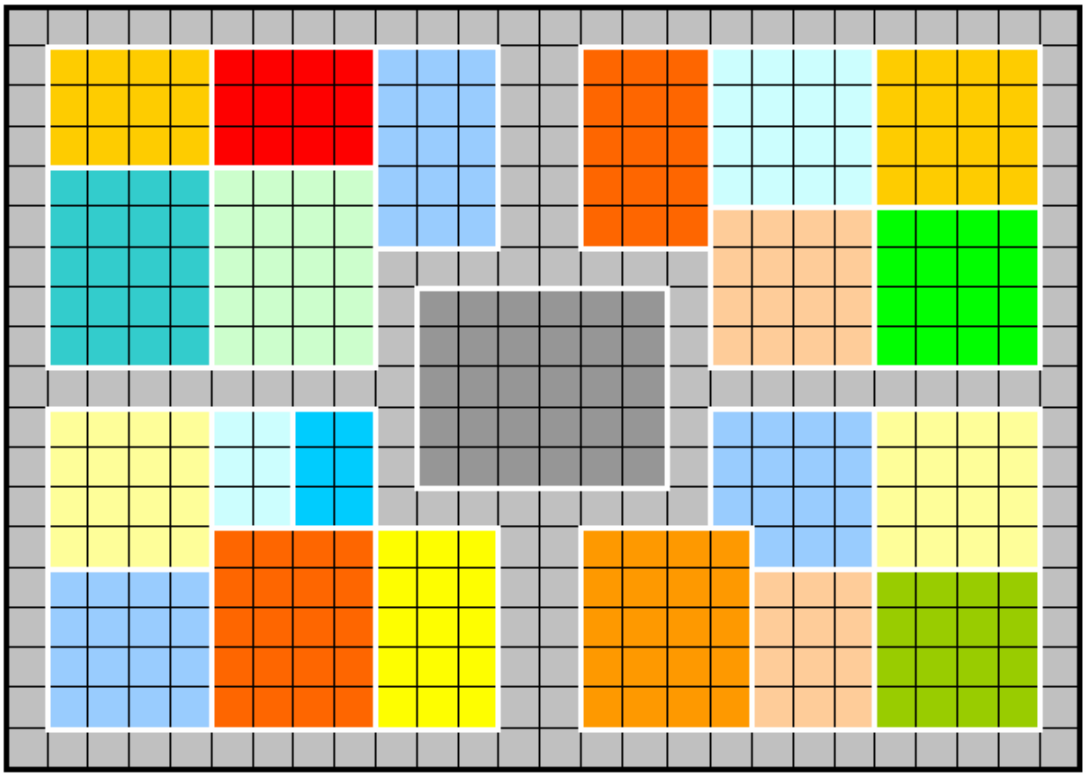
Vuonna 2005 Symbian OS v9.0 esiteltiin ja Symbian lisensoi Microsoft Exchange Server ActiveSyncin Symbian OS:ään. Lisäksi Nigel Clifford nimettiin Symbianin pääjohtajaksi. /1/

2.2 Symbian nykyisin

Symbian on tätä nykyä versiossa 9.5, mutta myynnissä olevissa puhelimissa versiot ovat pääsääntöisesti versiosta 8.1 eteenpäin. Suurin muutos Symbianin versiossa 9.1 ja sitä uudemmissa on ominaisuus nimeltä Platform Security, joka tekee puhelimesta huomattavasti turvallisemman. Platform Security on tapa suojata puhelin haittaohjelmilta ja viruksilta. Se pohjautuu ajattelutapaan, jossa ohjelmalle annetaan erilaisia kykyjä (capabilities), joilla rajoitetaan sen toimintaa. Tarkoitus siis on, että ohjelmalle annetaan pienin mahdollinen määrä näitä kykyjä, joilla se voi toimia oikein. Kykyjä ovat muun muassa ReadUserData, WriteUserData, LocalServices, Location ja monia muita. Kyvyt on jaettu kolmeen kastiin: Käyttäjätason kykyjä käyttäjä voi itse halutessaan hyväksyä asennuksen yhteydessä tai ohjelmiston tekijä voi hakea sertifikaatteja näille kyvyille. Seuraava on Luotettujen pavelimien taso, tämän tason kyvyille voi myös hake sertifikaatteja, mutta se vaatii huomattavasti suurempaa byrokratian pyöritystä. Hakijan tulee olla yritys, ja tuotetun ohjelman tulee läpäistä Symbianin tekemän testin, jossa tarkastetaan, että ohjelma todella tekee mitä sen on sanottu tekevän ja että se tarvitsee todella niitä kykyjä, jotka sille on haettu. Viimeinen taso on Trusted Computing Base (TCB), joka on puhelimen ydin. Tämä taso on itsessään kyky, jota eivät saa käyttöönsä muut kuin puhelimen valmistaja.

3 PELIN SÄÄNNÖT

Hero Quest on vanha lautapeli, joka toimii pohjana HQ-pelisovellukselle. Lautapelissä pelilautana toimii kuvaa 1vastaava alue, joka on kooltaan 26 x 19 ruutua. Ruudut muodostavat huoneita ja käytäviä, joita erottavat valkoiset seinät. Huoneita ja käytäviä voidaan yhdistää ovilla ja salaovilla. Ovia ja salaovia voi avata ja avaamisen jälkeen oven takana mahdollisesti olevat viholliset näkyvät. Viholliset aktivoituvat, kun pelaaja näkee nämä ensimmäistä kertaa. Käytävissä ja huoneissa olevissa ruuduissa voi olla ansoja, aarrearkkuja, esineitä, huonekaluja tai vihollisia. Viholliset voivat kantaa mukanaan esineitä tai rahaa, jotka pelaaja saa surmattuaan vihollisen. Huonekalut voivat myös sisältää piilotettuja esineitä, jotka löytyvät etsimällä. Etsimällä löytyvät myös ansat ja salaovet.



Kuva 1: Hero Questin peruspelialue.

4 SOVELLUKSEN TOTEUTTAMINEN

Luvussa esitellään sovelluksen kehityksessä käytetyt menetelmät sekä kerrotaan toteutuksen lähtökohdista ja varsinaisesta toteutuksesta.

4.1 Kehitys menetelmät

Niin sanotun tavallisen ohjelmistokehityksen lisäksi käytin myös kehitysmetodia nimeltä Test-Driven Development (TDD) eli Testivetoinen ohjelmistokehitys. Jos pohjalla ei olisi ollut jo olemassa oleva sovellus, kuten minulla oli. TDD-menetelmä olisi ollut minulla ensisijaisena menetelmänä. Nyt käytin TDD-menetelmää joissain tapauksissa ja soveltamalla.

Testivetoinen ohjelmistokehitys on kehitystekniikka, jossa kirjoitetaan ensin testitapaus ja sitten vain sen verran koodia, että testi onnistuu. Testivetoisella ohjelmistokehityksellä saadaan nopeasti palaute ja pystytään reagoimaan heti ohjelmointivirheisiin. Tämä tekniikka alkoi saada julkisuutta aivan 2000-luvun alussa aspektina Extreme Programming-metodologiassa.

Testivetoisessa kehityksessä suunnitellaan aluksi testi pienelle osalle ohjelman toiminnallisuutta. Vasta tämän jälkeen aletaan kirjoittaa koodia, joka toteuttaa testissä määritellyn toiminnallisuuden. Testivetoisen kehityksen tuloksena syntyy tarkoituksenmukainen ja selkeä ohjelmakoodi. TDD on erityisen hyödyllinen käytäntö silloin, kun tehdään vaativaa logiikkaa sisältävää koodia.

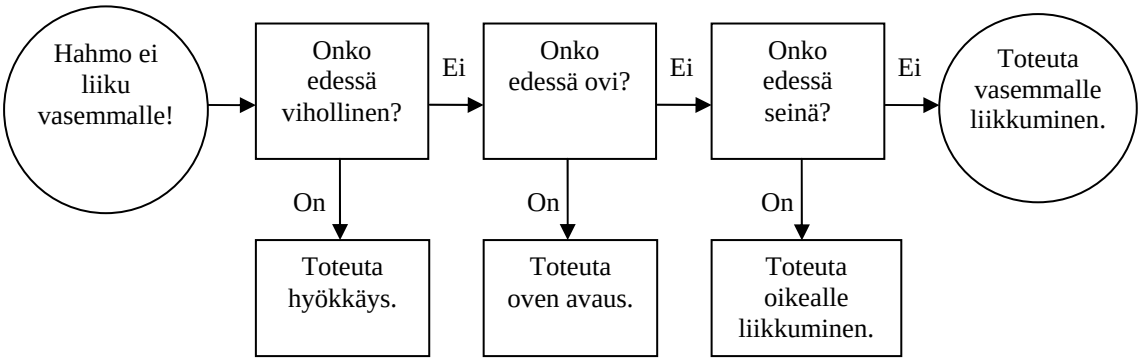
Testit toimivat formaalina määrittelynä toteutettavalle toiminnallisuudelle. Läpi menevät testit varmistavat, että toteutus täyttää testien kuvaamat vaatimukset.

TDD parantaa ohjelmakoodin laatua monella tapaa. Esimerkiksi hyvä yksikkötesti testaa yhtä pientä toiminnon osaa. Jotta tämä onnistuisi, järjestelmän eri osat pitää erottaa toisistaan. Näin syntyvä selkeä moduulijako ja palveluiden käyttö rajapintojen kautta auttavat kehittäjiä tekemään parempaa suunnittelua.

TDD:tä käytettäessä projektiin syntyy tuhansien yksikkötestien joukko, jota voidaan ajaa automaattisesti. Kehitettävän moduulin testijoukkoa ajetaan jatkuvasti, jolloin virheet löytyvät välittömästi ja niiden löytäminen ja korjaaminen on helppoa.

Testivetoisen kehityksen tuotteena syntyneen kattavan testijoukon merkityksen huomaa parhaiten, kun koodiin joudutaan syystä tai toisesta tekemään muutoksia. Jos esimerkiksi järjestelmän suorituskykyä halutaan parantaa optimoimalla, voidaan testit ajamalla varmistua, että kaikki toimii edelleen niin kuin pitikin. Testijoukko toimii myös aina ajan tasalla olevana dokumentaationa toteutuksen yksityiskohdista.

Parempi design, laadukas koodi ja turvaverkkona toimivat yksikkötestit laskevat ohjelmiston kustannuksia varsinkin ylläpitovaiheessa. /2/



Kuva 2: Hahmon eteneminen TDD-ajattelumallilla.

Kuva 2 esittää korkealla tasolla ongelmatilannetta, jossa hahmo ei liiku vasemmalle. Laatikot kuvaavat esteitä, joiden takia liikkuminen ei ole mahdollista haluttuun suuntaan.

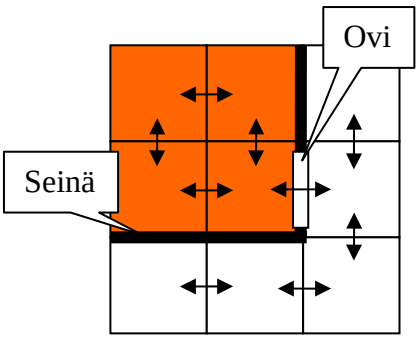
4.2 Toteutuksen lähtökohdat

Sovelluksen pohjana toimi Symbian kurssilla tekemäni peliohjelma, jota lähdin laajentamaan ja muuttamaan. Muutoksilla halusin saavuttaa mahdollisimman suuren yhtäläisyyden alkuperäisen Hero Quest-lautapelin pelimekaniikan kanssa ja

samalla laajentaa pelin mahdollisuuksia lähinnä pelimaailman suhteen. Pelin maailma ei olisi sidottu lautapelin kiinteään lautaan vaan pelialusta voisi olla vapaasti muutettavissa.

4.3 Toteutus

Aluksi minun täytyi muuttaa muutamia asioita sovelluksen alkuperäisestä toiminnasta. Ensin aloin tehdä maailmaan liittyviä muutoksia. Lauta koostuu ruuduista, jotka eivät tiedä toisistaan mitään eli esimerkiksi sitä, mihin mistäkin ruudusta pääsee. Ennen muutosta liikkuminen hoidettiin laskemalla jakojäännöksiä ja jakolaskuja laudan leveydestä, jotta saatiin selville, missä kohtaa pelialuetta hahmo on menossa. Tämä tapa on turhan monimutkainen ja erittäin altis virheille, joten muutin ruutuja siten että ne tietävät, mitkä ruudut ovat niiden ympärillä ja mihin niistä pääsee liikkumaan. Mikäli ruudusta ei pääse tiettyyn suuntaan, niin ruutuun piirretään sille sivulle seinä kuvastamaan, että tähän suuntaan ei pääse kulkemaan. Kuva 3 on esimerkki ruuduista, jotka on yhdistetty toisiinsa. Nuolet kuvaavat, mihin ruuduista pääsee liikkumaan.



Kuva 3: Toisiinsa yhdistetyt ruudut.

Ruutu tietää myös sisältääkö se oven, ja ovi taasen tietää, mitkä ruudut se yhdistää keskenään. Toteutin oven niin, että se voi johtaa muuallekin kuin viereiseen ruutuun. Ovi siis voi johtaa vaikkapa viiden ruudun päähän eli silloin se on niin sanottu teleportti. Ovi muodostetaan antamalla sille parametreina tyyppi, joka voi olla avoin-, suljettu, sala-, ansa- tai teleporttiovi. Oven asettamisen toteuttaminen olikin sitten jo huomattavasti vaativampaa, koska ajattelin vapauttaa kentän tekijän

oven asettamispaikan pohtimiselta, jos ruudussa sattuu olemaan jo ovi. Ovi asetetaan laudalle siten, että ovenasetusfunktiolle annetaan luotu ovi ja ruudut, jotka ovi yhdistää. Sovellus laskee ruutujen sijainnin perusteella, mille seinustalle ovi asetetaan kussakin ruudussa. Sovellus tarkastaa, ettei jo olemassa olevan oven päälle aseteta uutta ovea vaan uusi ovi tehdään ruudun tyhjälle sivulle.

5 PORTTAAMINEN

Alkuperäinen sovellus oli kehitetty Symbianin versiolle 8.0a, joka on jo vanha ja joka tarvitsisi muuttaa eli portata uudelle alustalle sopivaksi.

5.1 Porttaaminen lyhyesti

Porttaamisella tarkoitetaan koodin muuntamista, esimerkiksi ohjelmakoodin muuntamista toimivaksi jollain eri alustalla. Tässä tapauksessa porttaaminen tapahtuu Symbian versiosta 8.0a versioon 9.1, eli S60:n versio 2 FP2 portataan S60:n versioon 3.0.

S60:en 2.0- ja 3.0-ohjelmointirajapinnoista kovin moni ei eroa paljoakaan. Muutokset ovat ohjelman perusrakentaja, jossa on nykyisin tuki teemoille. Laitteen tapa käynnistää ohjelma on uusi, koska ohjelman tyyppi on muuttunut app-tiedostotyypistä exe-tiedostotyyppiin. Myös resursseissa on tapahtunut muutoksia, ja ohjelman rekisteröinti on uusi asia. Suurin muutos on turvatoimien mukaan tuominen. Seuraavat muutokset ovat välttämättömiä perusohjelmille:

- BLD.INF-tiedostoon lisätään mm. bittikarttojen ja ikoneitten rakentaminen.
- MMP-tiedostoon vaihdetaan uusi ohjelman tyyppi, kohde, SECUREID, VENDORID, CABAPILITY, resurssit- ja rekisteröintitiedot. Ohjelmalle tehtävien lokalisaatioresurssien rekisteröintitiedostot tulee ilmoittaa MMP-tiedostossa.
- Ohjelman perusrakentaja ja käynnistystapa
- RSS-resurssitiedosto
- Uudet REG-rekisteröintitiedostot.

5.2 Sovelluksen muutokset

BLD.INF-tiedosto

Aloitin sovelluksen porttaamisen Symbian 9.1:een tekemällä BLD.INF-tiedostosta, jota ei vielä ollut olemassa. Kirjoitin seuraavat rivit uuteen tiedostoon.

```
PRJ_PLATFORMS

        ARMV5 GCCE WINSCW
```

Prj_Platforms ilmoittaa kohdekääntäjät, joita projekti tukee. Tässä tapauksessa ARMV5, GCCE ja WINSCW.
ARMV5 kohdekääntäjä tukee EKA2-kernelarkkitehtuuriin pohjautuvia Symbian OS:n versioita, eli OS versiosta 8.1b ja ylöspäin rakentuvat ARMV5 kohdekääntäjällä. Koodi pyritään kääntämään siten, että se vastaa ARM:n kehittämää binaaristandardia nimeltä EABI. ARM5-arkkitehtuuri on 32-bittinen ja vanha THUMB on 16-bittinen.
GCCE-kohdekääntäjä käyttää ARM-koodin kääntämiseen vapaasti levityksessä olevaa GNU Compiler Collection (GCC) työkalua. GCCE-kääntäjä on tarkoitettu sovellusten kääntämiseen ja sillä ei voi kääntää kokonaista Symbian OS käyttöjärjestelmää, toisin kuin ARMV5-kääntäjä. GCCE-kääntäjän tuottamat binaaritiedostot ovat yhdenmukaisia EABI:n versio 2:n kanssa.
WINSCW-kohdekääntäjä kääntää koodin emulaattorille./3/

```
PRJ_MMPFILES

        gnumakefile scalable_icons.mk

        HQ_v3x.mmp
```

Prj_Mmpfiles ilmaisee projektiin kuuluvat makefilet ja MMP-tuedostot. Tiedostoja on kaksi: scalable_icons.mk ja hq-v3x.mmp. Scalable_icons.mk on gnumakefile, jolla rakennetaan vektori-ikonit. Hq-v3x.mmp on mmp-tiedosto Symbian 9.1 ja uudemmille projekteille. Tiedoston nimessä oleva v3 tulee S60 versio 3.0:sta.

MMP-tiedosto

Seuraavana muutettavaksi joutui vanha MMP-tiedosto, josta muokkaamalla sain uuden tiedoston, joka oli yhteensopiva 9.1 Symbianin kanssa. MMP-tiedoston muutokset olivat tarpeellisia, koska Symbian 9.1:ssä ja uudemmissa on mukana ominaisuus nimeltä Platform Security, mikä osaltaan vaatii uusia asioita MMP-tiedostoon.

```
TARGET          HQ.exe
TARGETTYPE      exe
```

Ensinnä kohteen tyyppi vaihtui app:stä exeen, joten kohde on HQ.exe. Nykyisin sovellukset ovat aina exe-tiedostoja tai sitten dll-tiedostoja.

```
UID             0x0 0xe82fb2fd
SECUREID        0xe82fb2fd
VENDORID        0
```

Seuraavaksi UID piti muuttaa uudeksi testi-UID-alueeksi, jolta voi ottaa uuden UID:n. Tämä alue on 0xEFFFFFFF - 0xEFFFFFFF väliltä. Tämä kyseinen UID-alue on tarkoitettu itse allekirjoitetuille (Self-Signed) sovelluksille, jollainen minun sovellukseni tulee olemaan. Uusia asioita ovat Secureid ja Vendorid.

```
CAPABILITY      ReadUserData
```

Platform Securityn takia minun piti määrittää sovellukselleni kyky ReadUserData, jotta voin lukea tiedostosta, ja vastaavasti WriteUserData pitäisi määrittää jos haluaisin tallentaa tiedostoon. Nämä kyvyt voi käyttäjä hyväksyä asennuksen yhteydessä ja erillistä sertifikaattia ei tarvitse hakea.

```
START RESOURCE      HQ.rss
        TARGETPATH \resource\apps
        HEADER
END
```

Rss-resurssitiedostoille tuli myös määrittää puhelimessa uusi sijainti, joka nykyisin on resource\apps-kansio, jonne talletetaan kaikkien puhelimessa olevien sovellusten resurssitiedostot. Resurssi tiedostojen esittelykin muuttui hieman. Resurssit tulee määrittää Start Resource … End-blokin sisään. Blokista tulee myös löytyä Header-kohta vaikka tämä kohta jäisikin tyhjäksi.

```
START RESOURCE        HQ_reg.rss
            TARGETPATH \private\10003a3f\apps
END
```

Bittikarttojen osalta muutos vaikutti ainoastaan kohdepolkuun, joka muuttui vanhasta polusta (\system\apps\ojelman_nimi) uuteen polkuun (\resource\apps).

```
START BITMAP                        HQ.mbm
            TARGETPATH \resource\apps
```

Resurssi-tiedostot

HQ.rss-tiedosto on resurssitiedosto, joka sisältää muun muassa valikkojen rakenteen ja eri näkymien tiedot. Seuraavaksi tekemäni muutokset RSS-tiedostoon.

```
RESOURCE LOCALISABLE_APP_INFO r_hq_localisable_app_info
    {
    short_caption = qtn_app_short_caption_string;
    caption_and_icon =
    CAPTION_AND_ICON_INFO
            {
            caption = qtn_app_caption_string;
            number_of_icons = 1;
            icon_file = "\\resource\\apps\\HQ.mif";
            };
    }
```

Resurssi tiedostoon lisättiin edellä oleva koodin pätkä, jossa esitellään mitä otsikoita ja ikoneita ohjelma tulee käyttämään. Tässä tapauksessa lyhyt otsikko, joka näytetään valikossa, haetaan lokalisointitiedostosta. CAPTION_AND_ICON_INFO kertoo varsinaisen otsikon sekä ikonien lukumäärän ja tiedoston, jossa ikonit ovat. Otsikkoteksti haetaan myös lokalisointitiedostosta.

Lähdekoodi-tiedostot

Lähdekoodiin tarvittiin myös muutoksia, jotta koodi toimisi uudessa Symbianissa. S60:n versio 3:ssa on määriteltynä kääntäjämakro __SERIES60_3X__, jolla pystyy ehdollistamaan muutokset. Aloitin muutokset muokkaamalla HQApp.h-tiedostoa seuraavasti:

```
#ifdef __SERIES60_3X__
            const TUid KUidHQ = { 0xe82fb2fd };
#else
            const TUid KUidHQ = { 0x031411cd };
#endif
```

Sovelluksen UID-muuttuja tarvitsee ehdollistamisen alueen muutoksen takia.

Seuraavana muutettiin HQApp.cpp.

```
#ifdef __SERIES60_3X__
#include <eikstart.h>
#endif
```

Ensimmäisenä ehdollistetaan eikstart.h-otsikkotiedoston mukaan ottaminen.

Sovellus on EXE, joten tarvitaan aloituskohtafunktioksi E32Main.

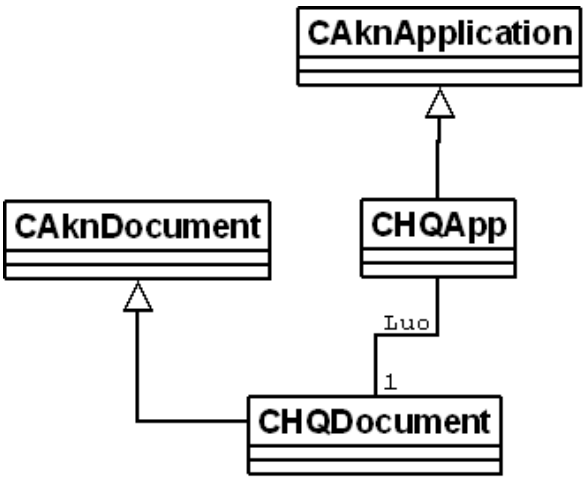
```
#ifdef __SERIES60_3X__
```

```
GLDEF_C TInt E32Main()
    {
    return EikStart::RunApplication( NewApplication );
    }
#else
GLDEF_C TInt E32Dll( TDllReason )
    {
    return KErrNone;
    }
#endif
```

Ehdollistetaan siis sovelluksen aloituskohdat. S60:n versio 2:ssa sovellukset käyttivät samaa aloituskohtaa kuin s60:n versio 3:ssa DLL-tyyppiset sovellukset. Lisäksi sovellus käynnistetään ajamalla funktio EikStart::RunApplication, joka löytyy eikstart.h-otsikkotiedostosta.

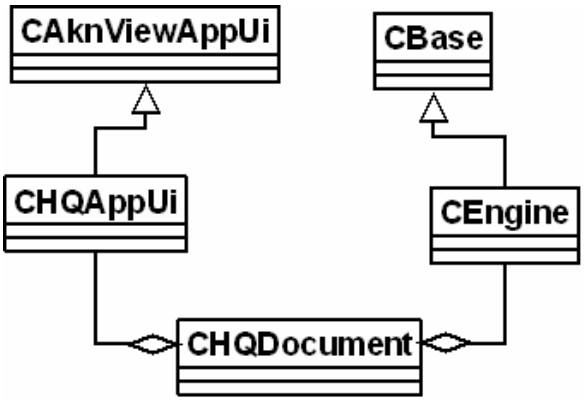
6 SOVELLUKSEN RAKENNE

Sovellus on rakenteeltaan normaali S60 3.0 version sovellus. Luokkarakenne alkaa applikaatioluokasta CHQApp, joka periytyy CAknApplication luokasta. CHQApp luo dokumenttiluokan CHQDocument, joka perii CAknDocument luokan. Sovelluksen perusrakenteen näet kuvassa 4.



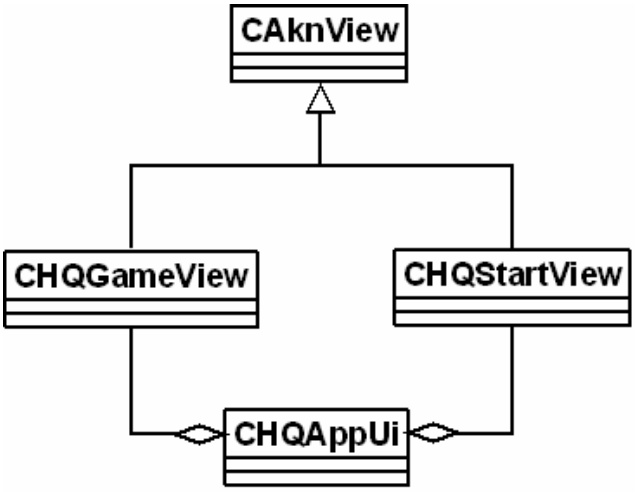
Kuva 4: Sovelluksen perusrakenne.

Dokumenttiluokka CHQDocument sisältää käyttöliittymäluokan CHQAppUi, joka periytetään CAknViewAppUi-luokasta. Dokumenttiluokka luo myös CEngine-luokan, joka on pelisovelluksen moottori. Tämä ilmenee kuvasta 5.



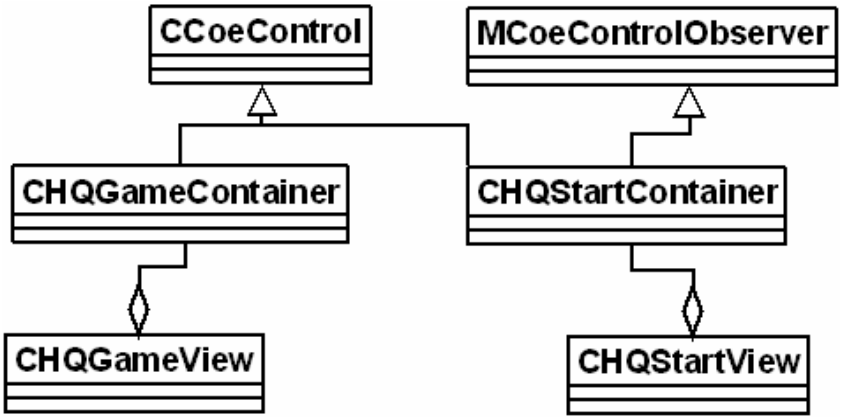
Kuva 5: Dokumenttiluokka koostuu CEnginestä ja CHQAppUi:sta.

CHQAppUi-luokka sisältää pelin kaksi päänäkymää CHQStartViewin ja CHQGameViewin. CHQStartView-näkymä on pelin aloitusnäkymä, joka on esillä kun peli käynnistetään. CHQGameView-näkymä on esillä pelattaessa peliä. Sille piirretään pelialue ja hahmot. Molemmat näkymät periytyvät CAknView-luokasta. Kuva 6 esittelee nämä luokat.



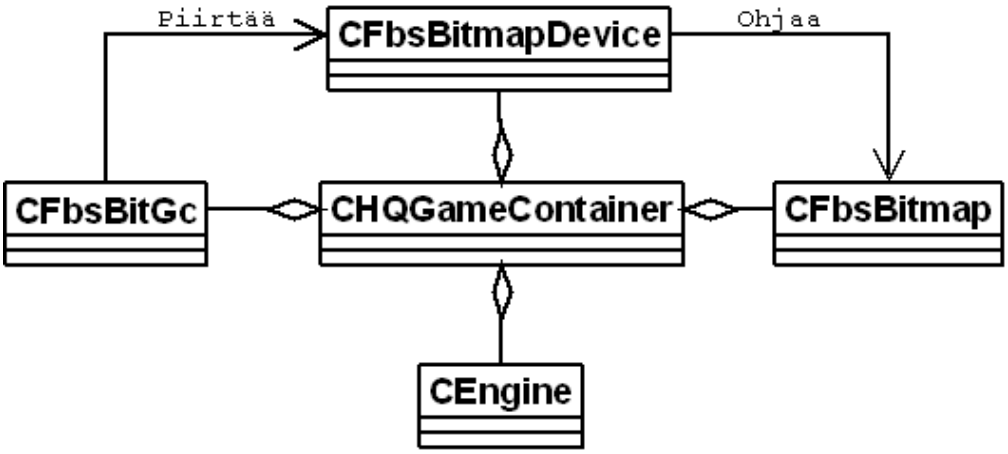
Kuva 6: CHQAppUi-luokka sisältää CHQGameView- ja CHQStartView-näkymät.

Molemmat näkymäluokat sisältävät container-luokat, jotka toteuttavat näkymän piirtämisen ja sisältävät näkymässä olevat kontrollit, kuten näppäinkutsujen käsittelyn ja menu tapahtumat. CHQGameContainer-luokka piirtää pelialueen ja kuuntelee pelaajan näppäinkomentoja. CHQStartContainer piirtää aloitus näkymän ja kuuntelee menu komentoja. Kuvassa 7 on luokkakaavio, josta voit nähdä kuinka Container-luokat asettuvat luokkahierarkiassa.



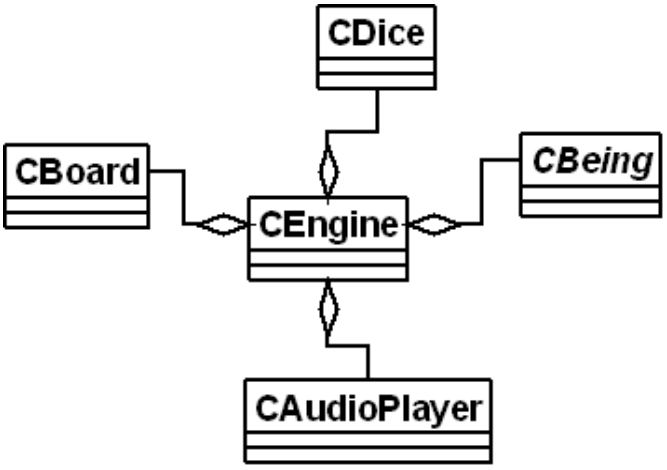
Kuva 7: Molemmat Container-luokat periytyvät CCoeControl-luokasta.

CHQGameContainer sisältää seuraavat merkittävät luokat. CFbsBitmap-luokka sisältää bittikartan pelialueesta. CFbsBitmapDevice-luokka ohjaa CFbsBitmap-luokkaa. CFbsBitGc piirtää CFbsBitmapDevicelle CEnginen määräämät asiat, kuten pelaaja pelihahmon, hirviöt ja pelialueen. Kuva 8 esittelee luokkakaaviona CHQGameContainer-luokan ja siihen liittyvät pääluokat.



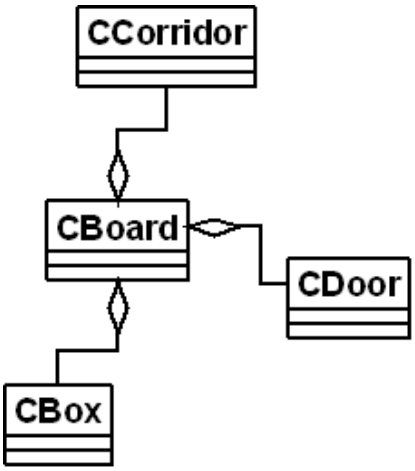
Kuva 8: CFbsBitGc piirtää CFbsBitmapDevicelle, joka ohjaa CFbsBitmapia.

CEngine toimii pelin moottorina ja näin ollen ohjaa pelin loogisia toimintoja. CEngine pitää sisällään, muun muassa seuraavat luokat. CDice-luokka toimii noppana. CAudioPlayer-luokka soittaa ääniä. CBoard on pelialusta, jossa pelaajan hahmo ja hirviöt liikkuvat. CBeing on abstraktikantaluokka pelissä oleville hahmoille. Hahmot ovat talletettu RPointerArrayn, josta CEngine löytää ne. Kuva 9 on UML toteutus CEngine-luokasta ja luokista joista se koostuu.



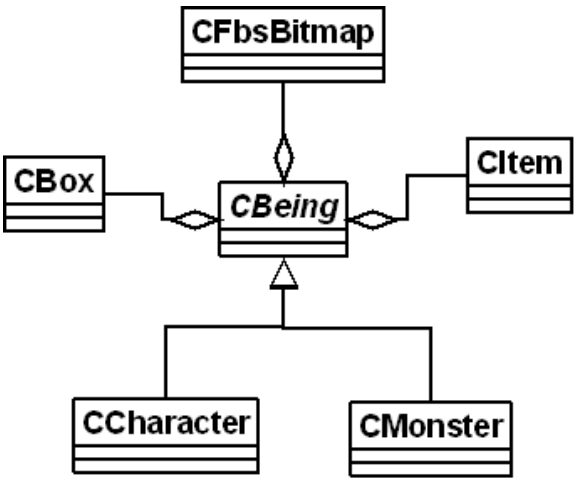
Kuva 9:CEnginen sisältämät luokat.

Pelialueena toimiva CBoard kuvaa maailmaa, jolla hahmot vaikuttavat. CBoard ohjaa luokkia CCorridor, CDoor. ja CBox, kuten CEngine-luokka määrää. CCorridor-luokka esittää aluetta, joka koostuu CBox-olioista. Tämä alue voi olla joko huone tai käytävä. CDoor toimii ovena, jonka kautta päästään kulkemaan. CBox on luokka ruudulle pelialueella, jossa hahmo sijaitsee. Kuva 10 on luokkakaavio, josta näkee CBoard-luokan ja sen sisältämät luokat.



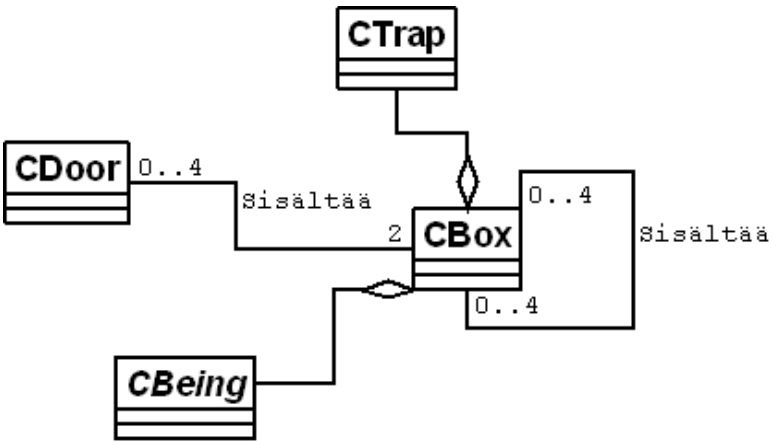
Kuva 10: CBoard koostuu luokista CBox, CCorridor ja CDoor.

CBeing on abstraktikantaluokka, jonka perii luokat CCharacter ja CMonster. CMonster- ja CCharacter-luokkien väliset erot ovat kuinka ne puolustavat ja liikkuvat. CMonster puolustautuu yhdellä mustalla kilvellä ja pelaaja kahdella valkoisella kilvellä. Liikkumisessa erona on, että hirviö liikkuu kiinteän määrän ruutuja ja pelaaja kaksi tai useamman nopan heiton verran. Merkittävimmät CBeing-luokan sisältämät luokat ovat CBox, CItem ja CFbsBitmap. CBox ilmaisee hahmon sijaintia pelilaudalla. CBeing-luokka sisältää CItem tyyppisen RPointerArray:n, joka toimii hahmon reppuna, jonne tallennetaan hahmon löytämät tavarat. CFbsBitmap-luokan olioita on CBeing-luokassa kaksi, jotka ovat hahmon kuvake ja kuvakkeen maski. Kuva 11 esittää, mistä CBeing-luokka koostuu ja mitkä luokat perivät sen.



Kuva 11: CBeing on abstraktikantaluokka CCharacter- ja CMonster-luokalle.

CBox-luokka toimii hahmojen varsinaisena sijoitus paikkana pelimaailmassa. CBox-luokka voi sisältää neljä sen ympärillä olevaa CBox-olion osoitetta. Lisäksi CBox-olio voi sisältää luokat CDoor, CTrap ja CBeing. CBeing on ruudussa oleva hahmo. CTrap toimii ansaa kuvaavana luokkana. CDoor toimii ovena, joka tietää sen yhdistämät CBox-oliot. Kuva 12 esittää UML-luokkakaaviona CBox-luokan suhteet toisiin CBox-olioihin.



Kuva 12: CBox-luokka tietää sen ympäröimät toiset CBox-luokat.

7 TESTAAMINEN

Tässä luvussa esitellään testausta ylisesti ja testaustyökalut, joita käytin. Lisäksi luvussa esitellään, kuinka EUnit-työkalulla tehdään yksikkötestejä.

7.1 Yleisesti testauksesta

Tein sovellukselle yksikkötestit jokaiselle luokalle, joille se oli mielekästä ja käytin yksikkötesteissä EUnit-työkalua. EUnit on Digia Oyj:n tekemä yksikkötestaustyökalu, jolla saa ajettua yksikkötestejä emulaattorissa tai laitteessa. Lisäksi sovellus lopputestattiin, millä varmistettiin se että sovellus todella toimii, kuten sen pitääkin. Kattavalla testaamisella saadaan aikaan luotettavia ohjelmia, sekä tasaisen hyvä laatu.

7.2 Testaustyökalut

EUnit on osa Quality Kit-tuotetta

EUnit on osa Quality Kit-tuotetta, joka on Digia Oyj:n tekemä työkalu Symbianpohjaisten mobiilisovellusten testaamiseen. Quality Kit sisältää EUnit-yksikkötestaustyökalun, sekä AppTest-käyttöliittymätestaustyökalun. Quality Kit tarjoaa myös ympäristön, jossa testejä voi ajaa automaattisesti.

EUnit

EUnit-testejä voidaan tehdä, joko käsin tai wizardin avulla. Wizardin avulla on hyvä tehdä testin pohja, josta sitten aletaan käsin muokata varsinaista testikoodia.

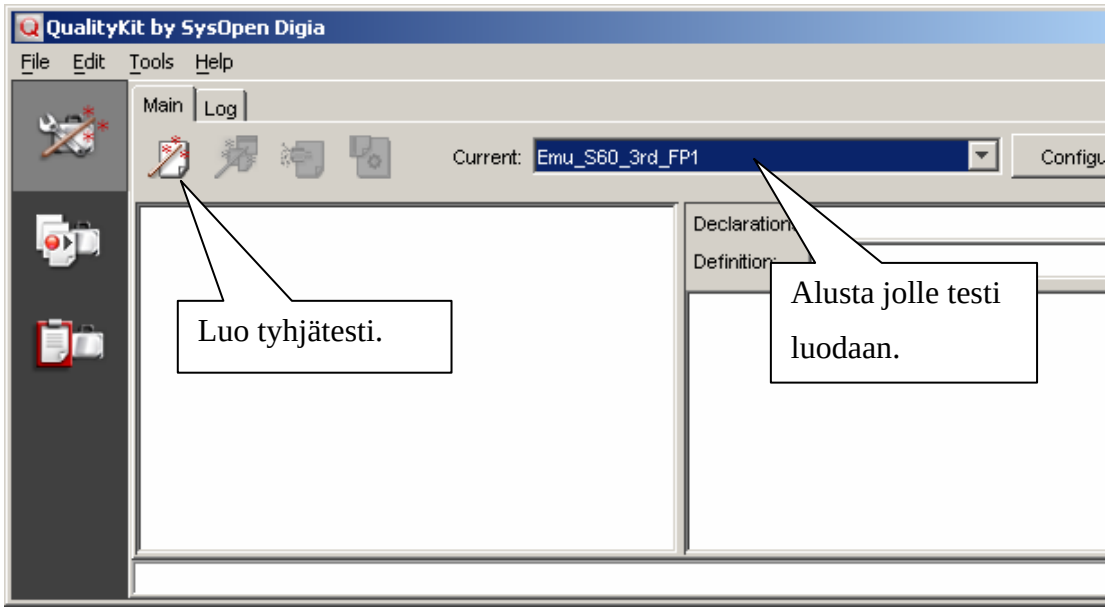
Muita yksikkötestaustyökaluja

Ehkä menestyksekkäin yksikkötestaustyökalu on Javalle tehty JUnit, jonka kehitti Kent Beck ja Erich Gamma. Muita yksikkötestaustyökaluja ovat CPPUnit C++:lle, NUnit C#:lle, PHPUnit PHP:lle ja JSUnit JavaScriptille.

7.3 Testien toteuttaminen

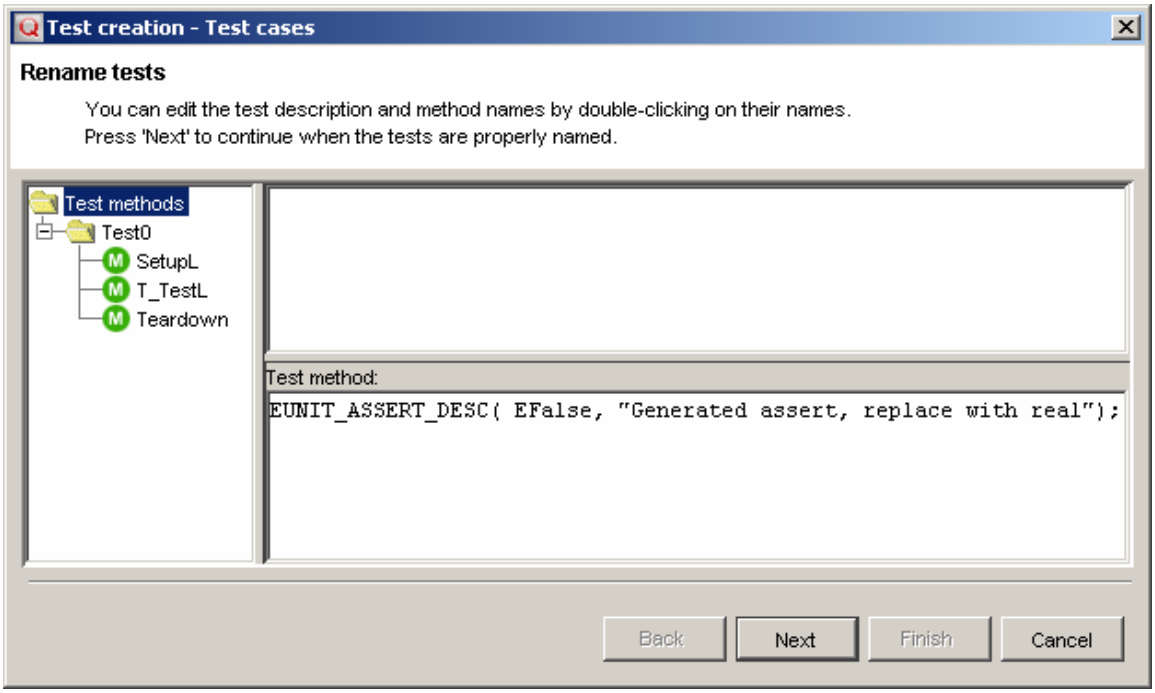
Testin pohjan tekeminen

Tuotin pohjan testille EUnit-työkalun wizardilla. Kuva 13 on QualityKitin Aloitus näkymä, joka tulee kun käynnistää EUnitin.



Kuva 13: Quality Kitin EUnitWizard

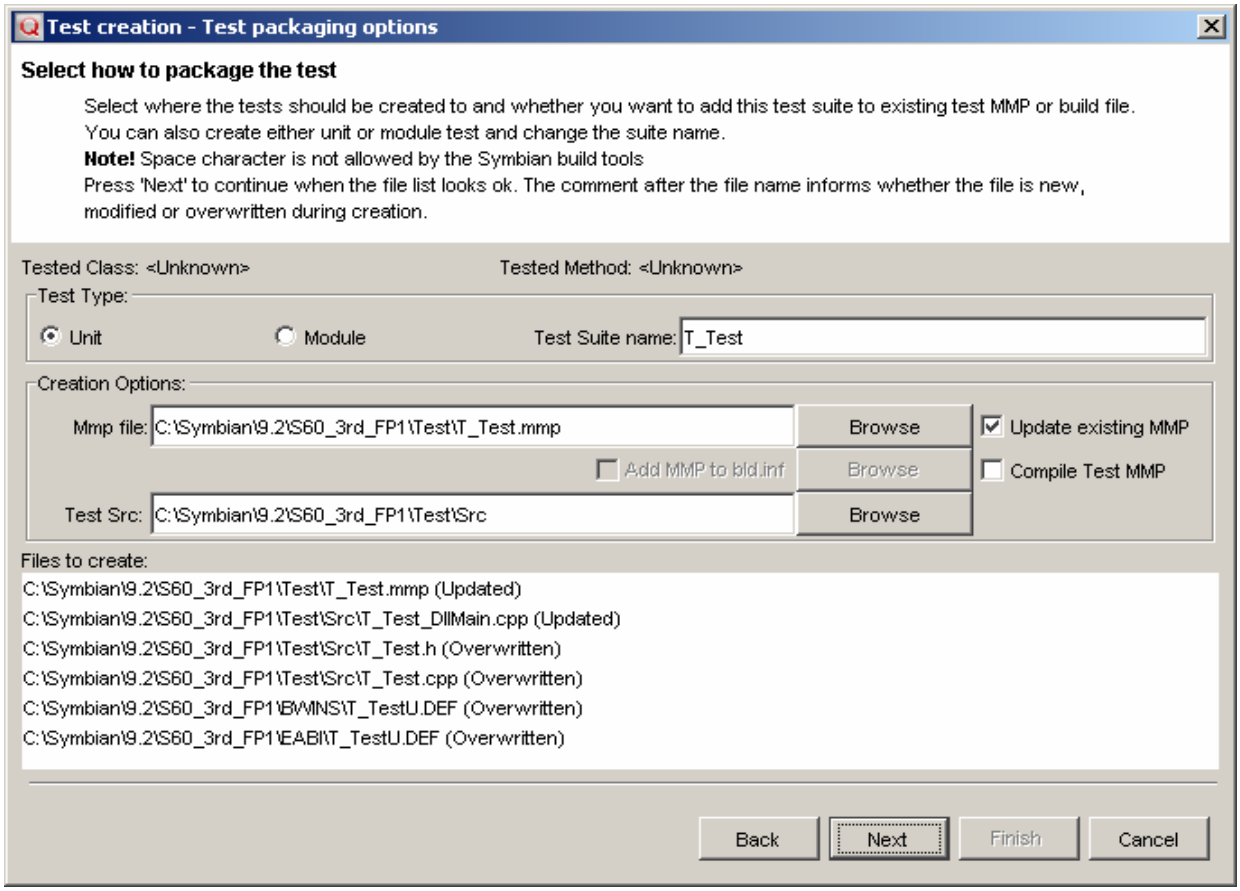
Valitaan alusta, jolle testi halutaan luoda ja painetaan luo tyhjätesti-nappia. Tämä aloittaa testin luomisen ja avaa samankaltaisen näkymän, kuin kuvassa 14. Ensinnä voidaan muuttaa testiluokan ja perusmetodien nimiä. On myös mahdollista kirjoittaa testikoodia haluttuun metodiin valitsemalla metodi ja kirjoittamalla koodi kohtaan Test method.



Kuva 14: Tästä näkymästä luodaan uusi testi ja testin sisältämät vakiometodit.

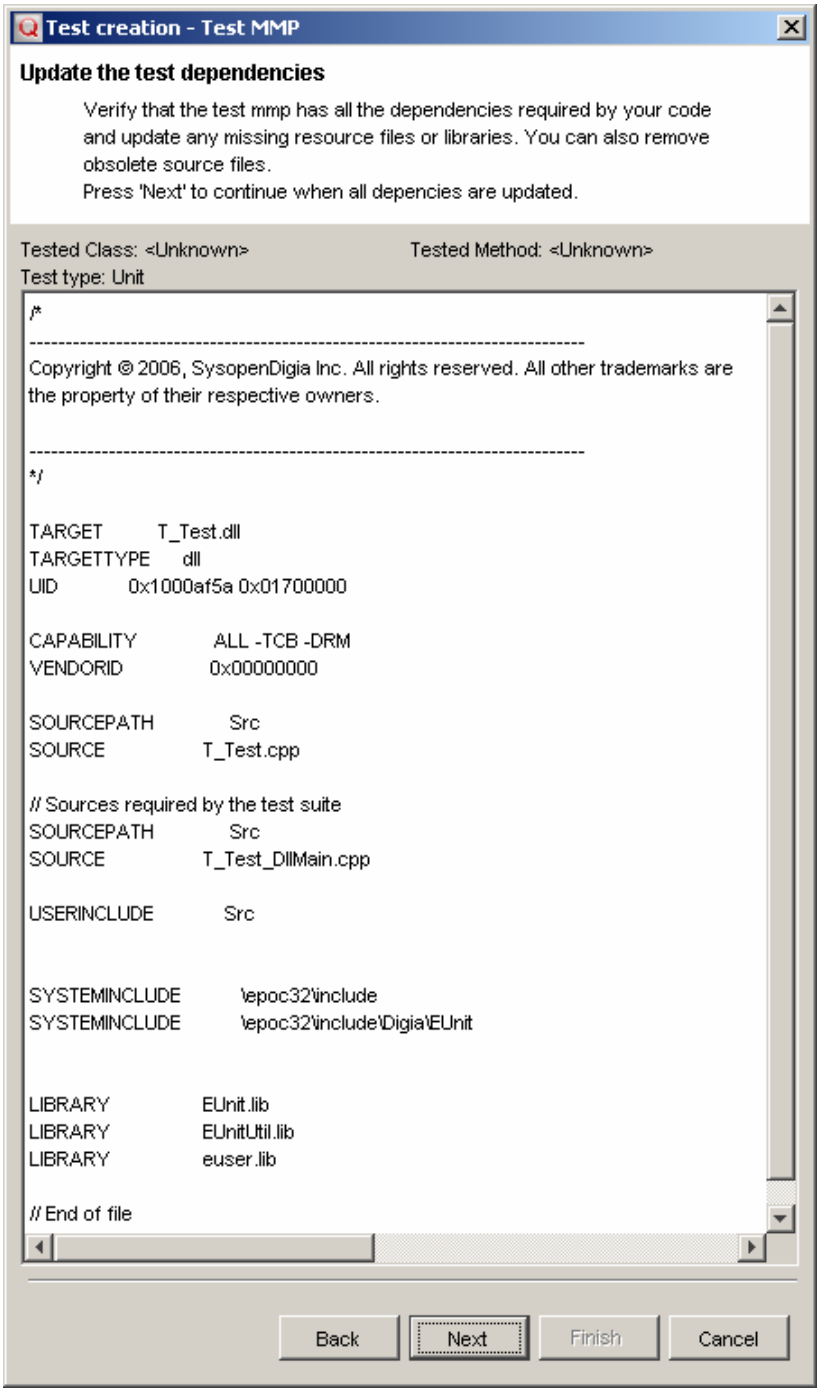
Testissä on seuraavat perusmetodit, jotka ajetaan ennen ja jälkeen jokaisen testin. Metodit ovat SetupL ja Teardown. SetupL ajetaan ennen testin suorittamista ja siinä voidaan alustaa testiluokan muuttujia testeille yhteisiin arvoihin. Teardown ajetaan jokaisen testin jälkeen ja sen tarkoitus on palauttaa ympäristö samaan tilaan kuin se oli ennen testin ajamista, esimerkiksi testissä luodut luokan jäsenmuuttuja nollataan ja poistetaan.

Seuraavana on vuorossa testin paketointivalikko, josta voidaan määrittää mihin uusi testi sijoitetaan. Näkymästä voidaan myös määrittää tehdäänkö testissä moduuli- vai yksikkötesti. Tämä näkymä on esitetty kuvassa 15.



Kuva 15: Testin paketointi valikko.

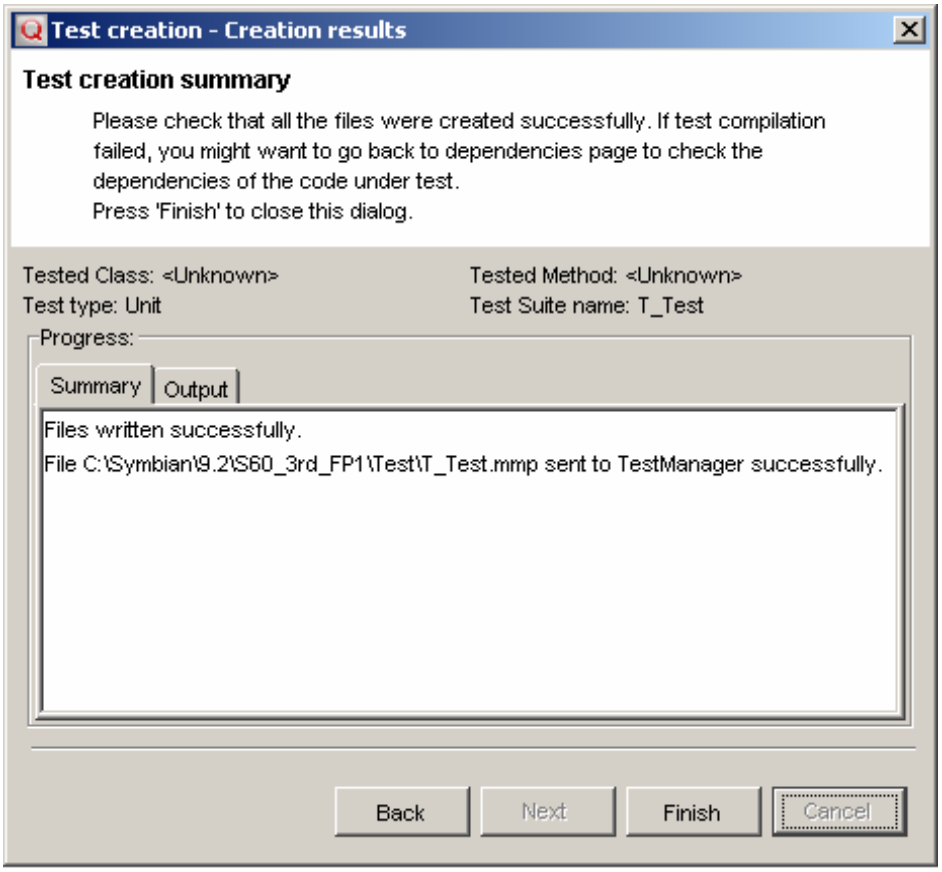
Testille voidaan antaa kuvaavampi nimi, esimerkiksi T_LuokkanNimi, jota testataan. Testi voidaan myös lisätä olemassa olevaan MMP-tiedostoon, mutta testi voidaan tehdä esimerkiksi luokkakohtaiseksi. Näin testit ovat paremmin hallittavia ja selkeämpiä kokonaisuuksia. Näkymästä voi myös nähdä mitä tiedostoja luodaan, jos testiä ei lisätä olemassa olevaan MMP-tiedostoon.



Kuva 16: Yksikkötestin MMP-tiedosto.

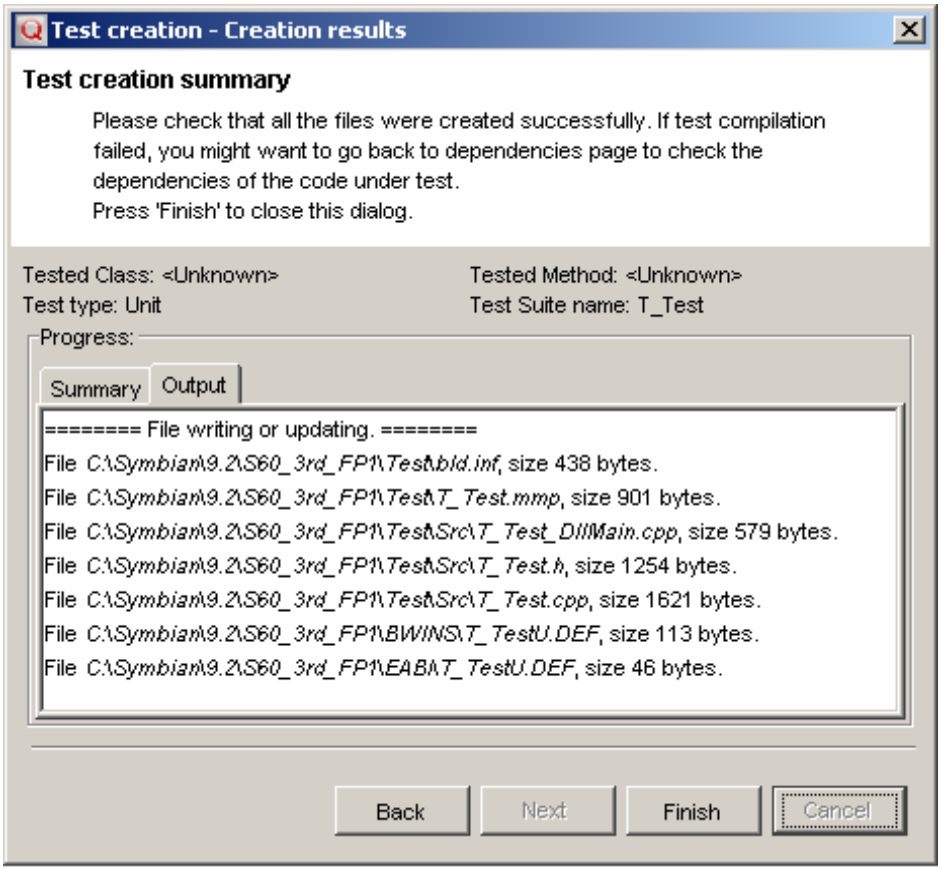
Kuva 16 esittää EUnitin luoman MMP-tiedoston. Tiedostoon tulee lisätä USERINCLUDE ja polku, jossa testattavakoodi sijaitsee.

Seuraavaksi EUnit generoi tiedostot ja antaa selvityksen, kuinka testin luonti onnistui. Testi lähetetään myös Quality Kitin TestManager-sovellukselle, josta testin voi ajaa. Kuvassa 17 on yhteenveto välilehti.



Kuva 17: Yhteenveto testin luonnista.

Kuva 18 esittelee Output välilehden, josta ilmenee tiedostot, jotka luotiin ja tiedostojen sijainnit. EUnit on tuottanut, muun muassa seuraavat tiedostot: bld.inf, MMP-tiedoston, testit sisältävät otsikko ja lähdekoodi tiedostot. Välilehdeltä näkee myös tiedostojen koot tavuina.



Kuva 18: Luodut tiedostot ja niiden sijainti.

Testin kirjoittaminen

Testikoodin kirjoittaminen etenee seuraavasti:

- Lisätään uuden testifunktion esittely otsikkotiedostoon.
- Lisätään funktio testitauluun, jotta EUnit osaa ajaa funktion.
- Toteutetaan testifunktio TDD-periaatetta noudattaen.

Edellisen listan mukaan esittelin ensin testifunktion otsikko tiedostoon. Esimerkkinä toimii T_CBox-luokka, jolla testasin CBox-luokan toimintaa. Lisäsin T_CBox.h-tiedostoon seuraavan funktion.

```
void T_CreateBoxL();
```

Testitauluun lisätään seuraava rivi.

```
Simple_Tests(T_CreateBoxL, CBox)
```

Ensinnä rivillä kerrotaan, että kyseessä on niin sanottu yksinkertainen testi. Missä testifunktio on T_CreateBoxL ja seliteteksti. Testifunktio testaa CBox-luokan rakentajaa. Kaikki toteuttamani yksikkötestit ovat yksinkertaisia testejä, koska niillä saatiin toteutettua testit tarvittavalla tarkkuudella.

Testifunktio toteutetaan seuraavasti:

```
iBox = new (ELeave) CBox(0,0);
EUNIT_ASSERT( iBox );
```

Testissä testataan saadaanko CBox-olio luotua. Luodaan uusi CBox ja vertaillaan EUNIT_ASSERT-makroa käyttäen onko iBox-muuttuja erisuuri, kuin NULL. Jos iBoxin arvo on erisuuri, kuin NULL niin voidaan olettaa, että olio on luotu. Tämä testifunktio ei ota kantaa mihinkään muuhun CBox-olion toiminnoista, koska ne tullaan testaamaan myöhemmin.

Seuraavaksi testataan muita CBox-luokan funktioita. Aloitetaan Set- ja GetCorridorId-funktiosta. Funktioilla asetetaan ja haetaan ns. huoneen tunnus, jolla yksilöidään huoneet ja käytävät. Testifunktio on seuraavanlainen.

```
TUint id = 1;
iBox = new (ELeave) CBox(0,0);
iBox->SetCorridorId( id );
EUNIT_ASSERT_EQUALS( id, iBox->GetCorridorId( ) );
```

Testissä luodaan uusi CBox-olio ja ID. ID asetetaan Cbox-olioon SetCorridotId-funktiolla, jonka jälkeen tarkastetaan vastaako GetCorridorId-funktiolla haettu ID ja asetettu ID toisiaan. Mikäli ID:t vastasivat toisiaan, niin testin tulos on hyväksytty.

IsFree-funktiolla selvitetään onko CBox-olio vapaa vai sisältääkö se jonkin asian, joka estää hahmoa tai vihollista liikkumasta kyseessä olevaan CBox-olioon. SetFree-funktiolla asetetaan kyseinen arvo todeksi tai epätodeksi. Testaus tapahtuu seuraavasti:

```
TBool free = EFalse;
iBox = new (ELeave) CBox(0,0);
iBox->SetFree( free );
EUNIT_ASSERT_EQUALS( free, iBox->IsFree( ) );
```

Luodaan uusi CBox-olio sekä Boolean-muuttuja free, joka asetetaan arvoon epätosi. SetFree-funktiolla asetetaan muuttuja free CBox-oliolle ja vertaillaan muuttuja freen arvoa ja IsFree-funktion paluuarvoa keskenään. Hyväksytty tulos saadaan, jos arvot vastaavat toisiaan.

Seuraavana toteutettiin testifunktio, jolla testataan SetVisible- ja IsVisible-funktiot. Nimensä mukaan näillä funktiolla asetetaan ja kysytään onko CBox-olio näkyvissä.

```
TBool visible = ETrue;
iBox = new (ELeave) CBox(0,0);
iBox->SetVisible( visible );
EUNIT_ASSERT_EQUALS( visible, iBox->IsVisible( ) );
```

Tuttuun tapaan luodaan uusi CBox-olio ja Boolean-muuttuja visible, joka asetetaan arvoon tosi. Muuttuja visible asetetaan CBox-oliolle SetVisible-funktiolla. Mikäli IsVisible-funktion palauttama arvo vastaa visible-muuttujan arvoa on tulos hyväksytty.

CBox-olion koon saa selville GetSize-funktiolla ja tuon funktion testi näyttää seuraavalta:

```
iBox = new (ELeave) CBox(0,0);
```

```
TSize size = iBox->GetSize( );
EUNIT_ASSERT_EQUALS( size, KBoxSize );
```

Luodaan vaadittava CBox-olio ja muuttuja size kokoa varten. CBox-olion GetSize-funktiolla noudetaan CBox-olion koko ja se sijoitetaan size-muuttujaan. Jos size vastaa KBoxSize-vakiota, joka sijaitsee CBox-olion otsikko tiedostossa, niin testin tulos on hyväksytty. Kaikki CBox-luokan olioiden koot asetetaan rakentajassa arvoon KBoxSize.

Seuraavilla testeillä testattiin CBox-olion SetSideBox- ja GetSideBox-funktioita. Testifunktion voit nähdä alla.

```
iBox = new (ELeave) CBox(0,0);
iBox2 = new (ELeave) CBox(0,1);
iBox->SetSideBox( iBox2, CBox::ELeft );
CBox* box = iBox->GetSideBox( CBox::ELeft );
EUNIT_ASSERT_EQUALS( iBox2, box );
```

Testissä luotaan kaksi CBox-oliota nimeltään iBox ja iBox2, sekä CBox-osoitinmuuttuja box. Olioon iBox asetetaan SetSideBox-funktiolla vasemmalle sivulle iBox2. Osoitinmuuttujaan box sijoitetaan GetSideBox-funktion palauttama arvo kun parametriksi on annettu vasensivu. Testi on hyväksytty kun iBox2 ja box vastaavat toisiaan. Testi toteutettiin myös muille sivuille. Eri sivujen nimet sijaitsevat CBox-olion nimiavaruudessa. Sivuja ovat ELeft, ERight, EUp ja EDown.

CBox-olio voi sisältää ovia, joten funktiot ovien asettamiselle ja hakemiselle oli tuotettava. Alla olevalla testillä testataan CBox-luokan funktioita GetDoor ja SetDoor.

```
iDoor = new (ELeave) CDoor(CDoor::ECloseDoor);
    iBox = new (ELeave) CBox(0,0);
```

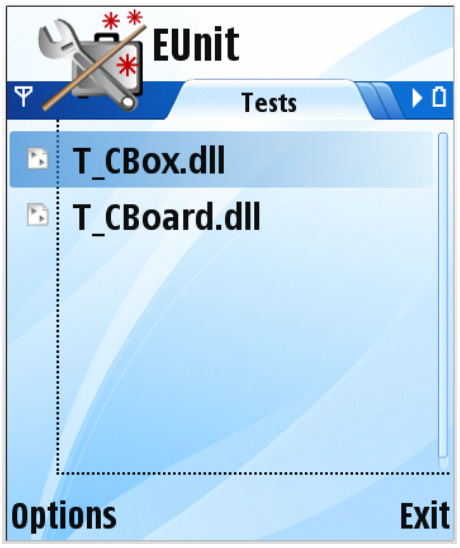
```
iBox2 = new (ELeave) CBox(0,1);
iBox->SetDoor( CBox::ELeft, iDoor );
iBox2->SetDoor( CBox::ERight, iDoor );
CDoor* door1 = iBox->GetDoor( CBox::ELeft );
EUNIT_ASSERT_EQUALS( door1, iDoor );
CDoor* door2 = iBox2->GetDoor( CBox::ERight );
EUNIT_ASSERT_EQUALS( door2, iDoor );
```

Aluksi testissä luodaan CDoor-tyyppinen olio iDoor, joka on nimensä mukaan ovi. Lisäksi luodaan kaksi CBox-oliota iBox ja iBox2, joiden välille ovi asetetaan. Ovi iDoor asetetaan iBox-olion vasemmalle sivulle ja iBox2-olion oikealle sivulle SetDoor-funktiolla. Funktiolle annetaan lisäksi parametri, jolla määrätään mille sivulle ovi asetetaan. Testissä luodaan myös kaksi paikallista CDoor-oliota door1 ja door2, joihin noudetaan GetDoor-funktiolla iBox-olion vasemmalla sivulla oleva ovi ja iBox2-olion oikealla sivulla oleva ovi. Kumpaakin noudettua ovea verrataan alkuperäiseen iDoor-olioon, jos ovet vastaavat toisiaan on tulos hyväksytty. Ylä- ja alasivuille vastaava testi on muutoin sama, mutta sivut ovat EUp ja EDown.

7.4 Testien suorittaminen

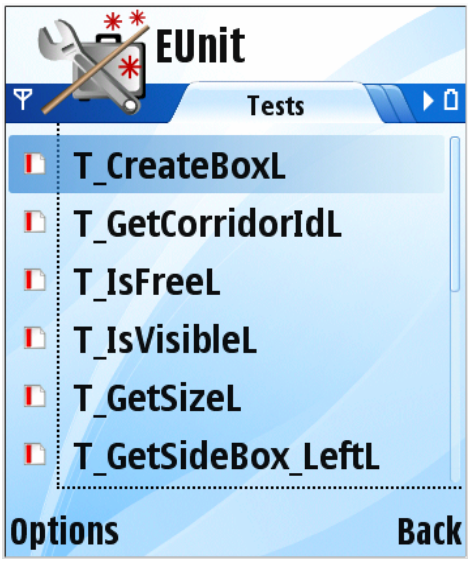
EUnit-testit

Yksikkötestauksessa testipetinä toimi Digia Oyj:n tekemä EUnit-yksikkötestaustyökalu. EUnit-testit voidaan suorittaan, joko laitteessa, emulaattorissa tai Quality Kit:ssä olevalla TestManager sovelluksella. Suoritin testit emulaattorissa, koska emulaattorista oli helpompi saada kuvakaappauksia työtäni varten. EUnit-ohjelmassa on kolme välilehteä Tests eli testit, Results eli tulokset ja Logs, josta näkee lokitiedoston nimen. Kuva 19 esittelee EUnitin testit välilehden. Testejä on kaksi T_CBox.dll ja T_CBoard.dll.



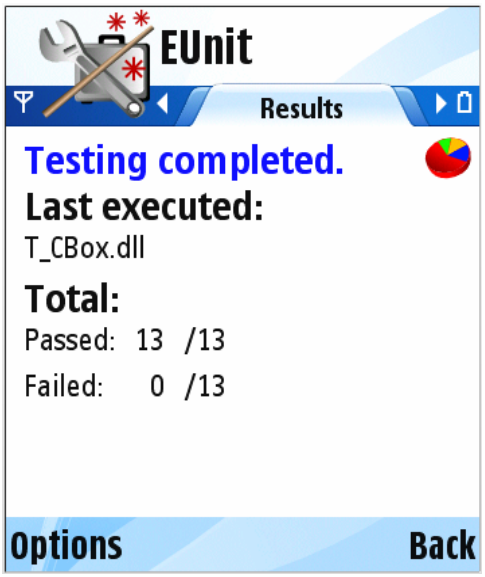
Kuva 19: EUnit-testit EUnit-ohjelmassa.

Testit saa avattua ja niiden sisältämiä alitestejä voi ajaa erikseen. Kuva 20 esittää mitä T_CBox.dll sisältää. Täällä sijaitsee luvussa Testin kirjoittaminen tehdyt testit.



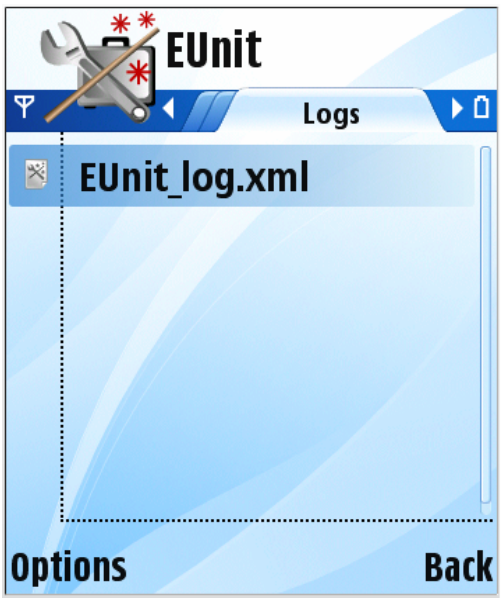
Kuva 20: Avattu T_CBox.dll EUnit-testi.

Kuva 21 esittää T_CBox-testin tuloksia. Tästä näkymästä näkee läpimenneiden ja epäonnistuneiden testien määrän ja sen, että testaus on päättynyt. Kuvasta nähdään, että kaikki ajetut testit on suoritettu onnistuneesti.



Kuva 21: T_CBox-testin tulokset.

Testit generoivat lokitiedostoa ja kuva 22 esittelee lokitiedoston nimen. Tiedosto sijaitsee varsinaisesti Epoc32\winscw\c\shared\EUnit\Logs hakemistossa kun testit ajetaan emulaattorissa. Lokitiedoston voit nähdä liitteessä 1. Lokitiedosto on tyypiltään xml ja siitä näkee mahdolliset virheet ja millä testin rivillä virhe on tapahtunut. Liitteessä 2 on esimerkki millainen lokitiedosto syntyy kun testi epäonnistuu.



Kuva 22: EUnit-testin generoima lokitiedosto.

Lopputestaus

Testaus suoritetaan yksinkertaisesti pelaamalla peliä. Pelaamisen ohessa yritetään aiheuttaa virhetilanne peliin tavalla tai toisella. Mikäli peli kaatuu tai aiheutuu virhe se kirjataan ylös ja pyritään löytämään korjaus. Jos korjauksen tekeminen ei ole mahdollista tehdään uudelleen funktio missä virhe tapahtui, jollakin toisella lähestymistavalla ja näin koetetaan estää virhetilanteen syntyminen.

8 KEHITYS IDEAT

Luonnollisia jatkokehitysideoita ovat, että pelin voisi tallentaa ja tallennuksesta voisi jatkaa myöhemmin. Peliin voisi lisätä sisäisen kenttäeditorin. Kenttäeditoriin voisi toteuttaa mahdollisuuden generoida satunnaisia kenttiä. Tämä kenttien satunnaisuus voisi olla myös pelissä. Kaupan lisääminen peliin siten, että pelaaja voisi käydä kaupassa kenttien välissä, kuten alkuperäisessä pelissä. Kentät voisivat sisältää tarinoita siitä, miksi pelaajan täytyy mennä kenttään. Pelaaja pystyisi taikomaan. Vaikeustaso olisi säädettävissä, joka toisi mukaan monimutkaisemman taistelu logiikan. Kyky tiirikoida arkkuja tai ovia, mikä taas luo tarpeen, että ovet ja arkut olisivat lukossa. Taistelu näkymä, joksi näkymä muuttuu taistelun alettua. Tässä näkymässä vain muutama ruutu hahmojen ympärillä näkyisi. Ruudun alalaitaan ilmestyisi vahinkomittari ja hahmojen nopan heittojen tulokset. Moninpelin mahdollisuus, jonkun langattoman protokollan kautta.

9 YHTEENVETO

Päättötyön tavoitteisiin päästiin mielestäni hyvin ja opin paljon uutta Symbianista ja sen käyttöliittymä komponenteista. Opin myös Test Driven Developmentia, sekä käyttämään EUnit-työkalua sovelluskehityksen ohessa. TDD on mainio tapa kehittää sovellusta. Ihmettelen suuresti, että sitä käytetään varsin vähän ohjelmistokehityksessä. Tähän voi olla syynä se, että ihmiset pelkäävät vaihtaa niin sanottuja tuttuja ja turvallisia tapojaan uusiin. Sama pätee myös Extreme Programmingia, jota käsitykseni mukaan käytetään varsin vähän suomalaisissa ohjelmistotaloissa. Mikäli Extreme Programmingia käytetäänkin, niin siitä ei liiemmälti huudella ulospäin.

Lopputestaus on työlästä ja melko hankalaa suunnitella etukäteen niin, että se kattaisi kaiken mahdollisen. Yksikkötestauksen kirjoittaminen jälkikäteen on työlästä ja melko kuluttavaa, mutta TDD tarjoaa tähän oivan keinon kirjoittaa testit samalla kun suorittaa kehitystyötä. Ongelmana on valmiiden testipetinä toimivien ohjelmien vähyys Symbianille. EUnit tarjoaa kuitenkin varsin mainion testipetisovelluksen, tosin siitä ei ole ilmaista versiota saatavilla.

Symbian ohjelmointi kielenä ei ole sen kummallisempaa kuin normaali c++, mutta dokumentoinnin puutteellisuus tekee siitä varsin haastavan. Symbianin koodaamisen aloittaminen tyhjältä pöydältä ilman mitään kurssia tai opastajaa, on todella haastavaa luokkien paljouden ja hieman erikoisten käytäntöjen takia. Dokumentointi on kyllä parantunut huomattavasti siitä, mitä se oli kun aloin tekemään tätä työtä, mutta parantamisen varaa silti on.

LÄHTEET

Sähköiset lähteet

1 Symbian Limited, [www-sivu], [viitattu 20.6.2007] Saatavissa: http://www.symbian.com/about/index.html

2 Test-Driven Development, [www-sivu], [viitattu 13.8.2007] Saatavissa: http://www.ri.fi/web/fi/teknologia-ja-tutkimus/tdd

3 SYMBIAN OS V9.2 [www-sivu], [viitattu 15.8.2007] Saatavissa: http://www.symbian.com/developer/techlib/v9.2docs/doc_source/ToolsAndUtilities/BuildTools/native/index.html

LIITTEET

LIITE 1

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<testreport version="3.0" date="2008-04-12T08:47:16Z" xmlns="xmlreporter.xsd">
<testparameters>
<testparameter name="EUnit version">EUnit 4.0.3</testparameter>
<testparameter name="Test executor">GuiRunner</testparameter>
<testparameter name="Target">WINSCW</testparameter>
<testparameter name="Platform">256</testparameter>
<testparameter name="Device">0x00004D24</testparameter>
<testparameter name="Test dll&apos;s">T_CBox</testparameter>
<testparameter name="Environment">S60AppEnv</testparameter>
<testparameter name="Logger in use">RDebug</testparameter>
<testparameter name="Logger in use">XML</testparameter>
<testparameter name="Test case timeout in seconds">30</testparameter>
<testparameter name="Resource checking mode">Error</testparameter>
<testparameter name="Panic note dismissal">Asserted</testparameter>
</testparameters>
<dll name="Z:\sys\bin\T_CBox.dll" size="13">
<testsuite name="These test are for CBox class." type="UNIT" size="13">
<testcase name="T_CreateBoxL" class="CBox" method="CBox" type="FUNCTIONALITY">
<result status="OK"></result>
</testcase>
<testcase name="T_GetCorridorIdL" class="CBox" method="GetCorridorId" type="FUNCTIONALITY">
<result status="OK"></result>
</testcase>
<testcase name="T_IsFreeL" class="CBox" method="IsFree" type="FUNCTIONALITY">
<result status="OK"></result>
</testcase>
<testcase name="T_IsVisibleL" class="CBox" method="IsVisible" type="FUNCTIONALITY">
<result status="OK"></result>
</testcase>
<testcase name="T_GetSizeL" class="CBox" method="GetSize" type="FUNCTIONALITY">
<result status="OK"></result>
</testcase>
<testcase name="T_GetSideBox_LeftL" class="CBox" method="GetSideBox" type="FUNCTIONALITY">
<result status="OK"></result>
</testcase>
<testcase name="T_GetSideBox_RightL" class="CBox" method="GetSideBox" type="FUNCTIONALITY">
<result status="OK"></result>
</testcase>
<testcase name="T_GetSideBox_UpL" class="CBox" method="GetSideBoxL" type="FUNCTIONALITY">
<result status="OK"></result>
</testcase>
<testcase name="T_GetSideBox_DownL" class="CBox" method="GetSideBox" type="FUNCTIONALITY">
<result status="OK"></result>
</testcase>
<testcase name="T_GetSideBox_NullL" class="CBox" method="GetSideBox" type="FUNCTIONALITY">
<result status="OK"></result>
</testcase>
<testcase name="T_GetDoor_LeftRightL" class="CBox" method="GetDoor" type="FUNCTIONALITY">
<result status="OK"></result>
</testcase>
<testcase name="T_GetDoor_UpDownL" class="CBox" method="GetDoor" type="FUNCTIONALITY">
<result status="OK"></result>
</testcase>
<testcase name="T_GetDoor_NullL" class="CBox" method="GetDoor" type="FUNCTIONALITY">
<result status="OK"></result>
</testcase>
</testsuite>
</dll>
</testreport>
```

LIITE 2

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<testreport version="3.0" date="2008-04-12T09:50:30Z" xmlns="xmlreporter.xsd">
<testparameters>
<testparameter name="EUnit version">EUnit 4.0.3</testparameter>
<testparameter name="Test executor">GuiRunner</testparameter>
<testparameter name="Target">WINSCW</testparameter>
<testparameter name="Platform">256</testparameter>
<testparameter name="Device">0x00004D24</testparameter>
<testparameter name="Test dll&apos;s">T_CBoard</testparameter>
<testparameter name="Environment">S60AppEnv</testparameter>
<testparameter name="Logger in use">RDebug</testparameter>
<testparameter name="Logger in use">XML</testparameter>
<testparameter name="Test case timeout in seconds">30</testparameter>
<testparameter name="Resource checking mode">Error</testparameter>
<testparameter name="Panic note dismissal">Asserted</testparameter>
</testparameters>
<dll name="Z:\sys\bin\T_CBoard.dll" size="16">
<testsuite name="These test are for CBoard class." type="UNIT" size="16">
<testcase name="T_CreateBoardL" class="CBoard" method="NewL" type="FUNCTIONALITY">
<result status="OK"></result>
</testcase>
<testcase name="T_GetViewIndexL" class="CBoard" method="GetViewIndex" type="FUNCTIONALITY">
<result status="OK"></result>
</testcase>
<testcase name="T_GetViewIndex_NegativeL" class="CBoard" method="GetViewIndex" type="FUNCTIONALITY">
<result status="OK"></result>
</testcase>
<testcase name="T_GetViewIndex_TooBigL" class="CBoard" method="GetViewIndex" type="FUNCTIONALITY">
<result status="OK"></result>
</testcase>
<testcase name="T_GetCorridorIdOfBoxL" class="CBoard" method="GetCorridorIdOfBox" type="FUNCTIONALITY">
<result status="OK"></result>
</testcase>
<testcase name="T_GetCorridorIdOfBox_FailL" class="CBoard" method="GetCorridorIdOfBox" type="FUNCTIONALITY">
<result status="OK"></result>
</testcase>
<testcase name="T_GetDoorOfBoxL" class="CBoard" method="GetDoorOfBox" type="FUNCTIONALITY">
<result status="OK"></result>
</testcase>
<testcase name="T_GetBoardSizeL" class="CBoard" method="GetBoardSize" type="FUNCTIONALITY">
<result status="OK"></result>
</testcase>
<testcase name="T_GetViewAreaL" class="CBoard" method="GetViewSize" type="FUNCTIONALITY">
<result status="OK"></result>
</testcase>
<testcase name="T_IsBoxFreeL" class="CBoard" method="IsBoxFree" type="FUNCTIONALITY">
<result status="OK"></result>
</testcase>
<testcase name="T_GetBoxVisibleL" class="CBoard" method="GetBoxVisible" type="FUNCTIONALITY">
<result status="OK"></result>
</testcase>
<testcase name="T_GetBoxVisible_FailL" class="CBoard" method="GetBoxVisible" type="FUNCTIONALITY">
<result status="OK"></result>
</testcase>
<testcase name="T_IsCorridorL" class="CBoard" method="IsCorridor" type="FUNCTIONALITY">
<result status="OK"></result>
</testcase>
<testcase name="T_IsCorridor_NotL" class="CBoard" method="IsCorridor" type="FUNCTIONALITY">
<event>
<print>EUNIT_ASSERT_EQUALS failure: asserted iBoard-&gt;IsCorridor( box ) == EFalse, actual values: 1 and 0</print>
</event>
<event>
<assertionfailed>
<condition>EFalse</condition>
<description>EUNIT_ASSERT_EQUALS</description>
<sourcefile>T_CBoard.cpp</sourcefile>
<sourceline>244</sourceline>
</assertionfailed>
</event>
<result status="FAIL"></result>
</testcase>
<testcase name="T_SetRoomL" class="CBoard" method="SetRoomL" type="FUNCTIONALITY">
<result status="OK"></result>
</testcase>
<testcase name="T_SetCorridorL" class="CBoard" method="SetCorridorL" type="FUNCTIONALITY">
<result status="OK"></result>
</testcase>
</testsuite>
</dll>
</testreport>
```