TAMPEREEN AMMATTIKORKEAKOULU TUTKINTOTYÖ
Tietotekniikan koulutusohjelma, ohjelmistotekniikka
Petri Alanenpää

Tutkintotyö

Petri Alanenpää

MobTactics-mobiilipeli

Työn valvoja Lehtori Tony Torp
Tampere 2008

TAMPEREEN AMMATTIKORKEAKOULU TUTKINTOTYÖ
Tietotekniikan koulutusohjelma, ohjelmistotekniikka
Petri Alanenpää

TAMPEREEN AMMATTIKORKEAKOULU
Tietotekniikan koulutusohjelma
Ohjelmistotekniikka
Alanenpää, Petri MobTactics-mobiilipeli
Tutkintotyö 40 sivua + 1 liitesivu
Työn valvoja Lehtori Tony Torp
Elokuu 2008
Hakusanat mobiilipeli, Symbian

TIIVISTELMÄ

Tämän insinöörityön tarkoituksena on tutustua peliohjelmointiin sekä Symbian OS -ympäristöön. Symbian OS on matkapuhelimille kehitetty oliopohjainen käyttöjärjestelmä, joka perustuu vahvasti asiakas – palvelin arkkitehtuuriin. Toteuttamani MobTactics -peli on vuoropohjainen strategiapeli, joka on ohjelmoitu Symbian C++-ohjelmointikielellä.

Matkapuhelimien rajalliset resurssit täytyy ottaa hyvin huomioon peliä tehdessä. Suuret bittikartat ja nopea ruudunpäivitys vaativat paljon muistia ja näyttöruudun rajallinen koko tuovat haasteita grafiikan toteutukseen. Hyvä ja huolellinen suunnittelu takaa sen, että toteutusvaiheessa suuremmilta ongelmilta vältytään ja pelistä tulee suunnitelman mukainen.

Lopputuloksena on suunnitelmien mukainen peli, jota on helppo laajentaa esimerkiksi uusilla kentillä sekä lisäominaisuuksilla.

TAMPEREEN AMMATTIKORKEAKOULU TUTKINTOTYÖ
Tietotekniikan koulutusohjelma, ohjelmistotekniikka
Petri Alanenpää

TAMPERE UNIVERSITY OF APPLIED SCIENCES

Computer Systems Engineering
Software Engineering
Alanenpää, Petri MobTactics -mobilegame
Engineering Thesis 40 pages + 1 appendice
Thesis Supervisor Tony Torp (MSc)
August 2008
Keywords Mobilegame, Symbian

ABSTRACT

The purpose of this thesis is to learn game development and to learn more about Symbian OS. Symbian OS is an operating system designed for mobile devices. It uses object-orientated client-server-architecture. The developed game, MobTactics, is a turn-based strategy-game and it is programmed using Symbian C++ programming language.

Limited resources of mobile devices must be acknowledged when developing a game. Large bitmaps and fast frame rate demand lots of memory and the relatively small screen size brings challenges to graphics design. Careful and thorough planning guarantees that implementing of the game is problem-free and that the game will come out as planned.

In the end, the game had all the features that were designed - and even more. It is easy to expand the game, for example, by adding more maps or features.

SISÄLLYSLUETTELO

SANASTO

Symbian	Mobiililaitteille suunniteltu käyttöjärjestelmä.
S60	Series 60, ohjelmistoalusta Symbianille.
DLL	Dynamically Linked Library, jaettu kirjasto.

1 JOHDANTO

Tämän työn aiheena oli tuottaa vuoropohjainen strategiapeli Symbian S60 3rd Edition -matkapuhelimille. Työllä ei ollut varsinaista teettäjää, vaan tein sellaisen pelin kuin itse halusin. Tässä dokumentissa keskitytään enimmäkseen pelin suunnitteluun ja toteutukseen. Symbian OS -käyttöjärjestelmä sekä S60-alusta esitellään vain pintapuolisesti.

Luvussa 2 kerrotaan Symbianin historiasta ja kehityksestä sekä S60-alustan rakenteesta. Pelin suunnittelusta, arkkitehtuurista ja siihen haluamistani toiminnoista kerrotaan luvussa 3. Pelin toteutuksen tärkeimmät ja haastavimmat asiat ovat luvussa 4. Lopuksi kerron jatkokehitysideoista luvussa 5, luvussa 6 on yhteenveto siitä, miten työ mielestäni onnistui.

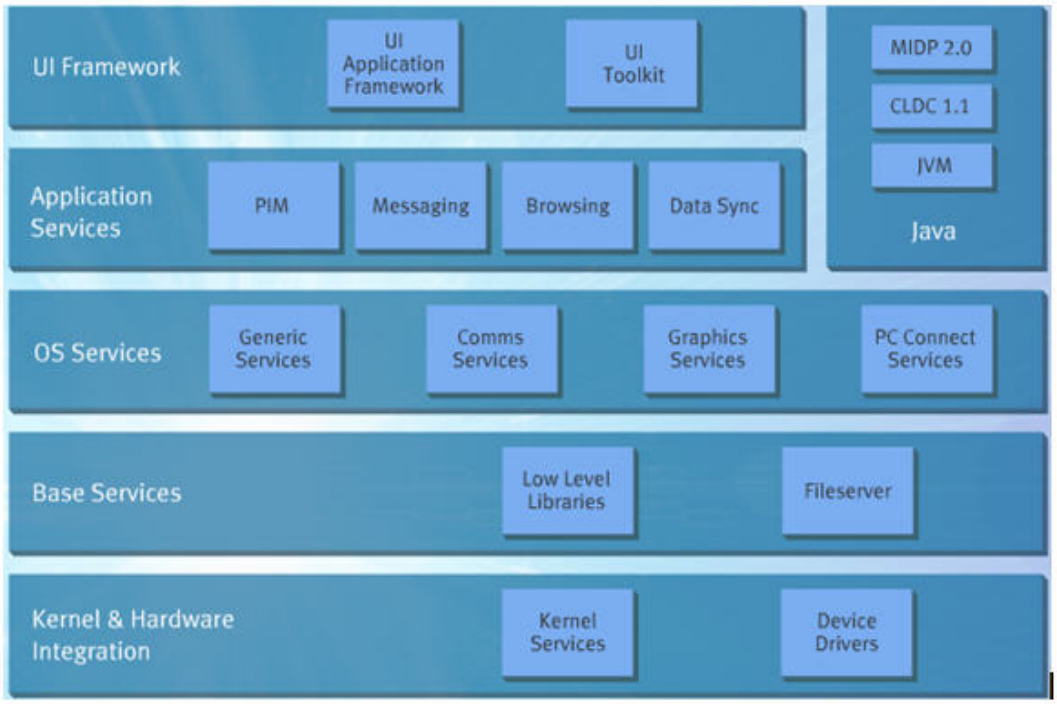
Työn tarkoituksena oli oppia, kuinka peliohjelmointi etenee ideasta toteutustasolle. Vaikka olenkin aikaisemmin koulun ohjelmointikursseilla ollut mukana toteuttamassa pelejä Symbian-ympäristöön, niin halusin kokeilla, pystynkö toteuttamaan pelin alusta loppuun asti itsenäisesti. Samalla halusin oppia myös lisää Symbian-ympäristöstä.

2 SYMBIAN OS JA S60-ALUSTA

Symbian OS on matkapuhelimia ja muita kannettavia laitteita varten kehitetty käyttöjärjestelmä. Sen kehitti Symbian Ltd käyttäen perustanaan Psionin kehittämää EPOC-käyttöjärjestelmää. Symbian Ltd perustettiin vuonna 1998 omistajinaan Nokia, Ericsson, Motorola ja Psion. Nykyään Symbian OS-käyttöjärjestelmää lisensoivat myös mm. Fujitsu, LG, Mitsubishi, Samsung ja Sharp. /1/

Symbian OS on moniajoa tukeva mikrokernelpohjainen käyttöjärjestelmä, joka voidaan jakaa viiteen moduuliin:

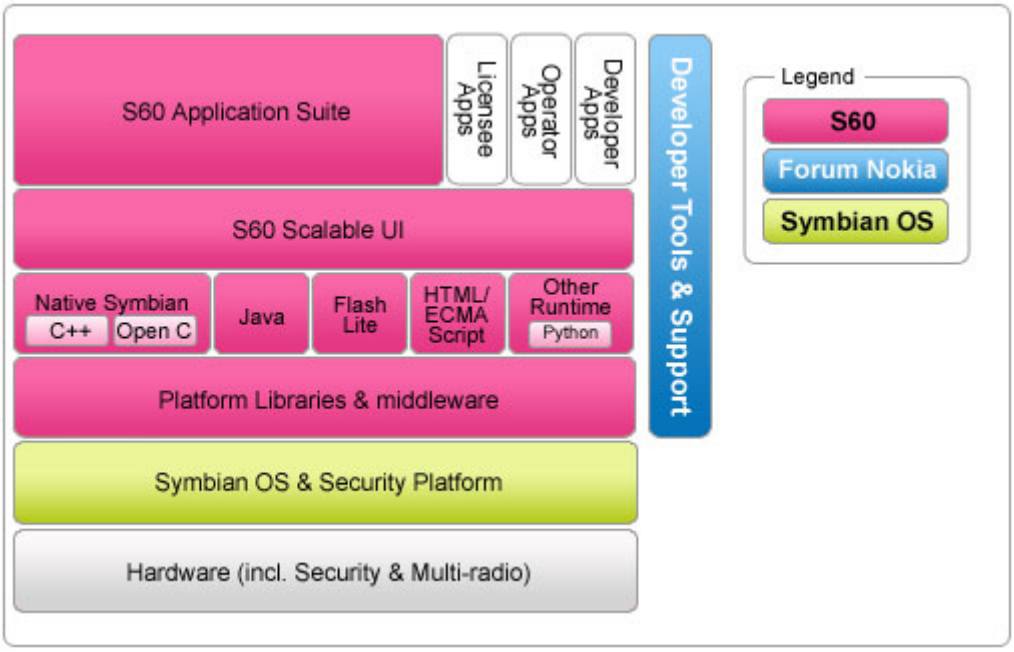
- **Kernel** eli käyttöjärjestelmän ydin on käyttöjärjestelmän alimman tason osa, joka toimii esimerkiksi rajapintana laiteajureille. Kernel myös hoitaa matkapuhelimen järjestelmämuistin käyttöä, ja sen toimintoja voidaan lisätä esimerkiksi ajureilla tai DLL-kirjastoilla.
- **Peruskirjastot** sisältävät lukuisia rajapintoja esimerkiksi tiedostonhallintaan, virheiden hallintaan ja ajastimiin. Näiden kirjastojen kautta pääsee myös käsiksi kernelin toimintoihin.
- **Järjestelmäpalvelut** laajentavat peruskirjastoja tarjoamalla kehyksiä ja kirjastoja esimerkiksi tietoliikennepalveluihin ja multimediapalveluihin.
- **Applikaatiopalvelut** sisältävät peruskehykset sovelluksia varten sekä palveluita esimerkiksi tietoliikenne- tai multimediaprotokollia varten.
- **Käyttöliittymäkehys** sisältää esimerkiksi kehyksiä ja kirjastoja graafisen käyttöliittymän toteutukseen.



Kuva 2.1 Symbian OS -käyttöjärjestelmän moduulit /5/

Kuvasta 2.1 nähdään, että moduulien sisäiset toiminnot ovat jaettu omiin järjestelmiinsä helpottamaan niiden käyttöä. Symbian-käyttöjärjestelmän toiminta perustuu vahvasti asiakas-palvelin-arkkitehtuuriin. Esimerkiksi käyttöjärjestelmän ydin eli kernel, on palvelin, joka hoitaa alimman tason tehtävät ja bittikarttapalvelin (Font and Bitmap Server) huolehtii näytölle piirrettävistä bittikartoista. Symbianissa jokainen sovellus suoritetaan omassa prosessissaan eivätkä prosessit näe suoraan toisten prosessien muistialueelle. Tämä sekä arkkitehtuurin oliomallinen rakenne tekevät Symbianista hyvin suojatun järjestelmän.

S60 on Nokian kehittämä käyttöliittymä ja kehitysalusta Symbian OS-käyttöjärjestelmälle. Visuaalinen käyttöliittymä mahdollistaa esimerkiksi uusien sovellusten asentamisen puhelimeen sekä helpottaa puhelimen ja sen eri toimintojen käyttöä. Se myös auttaa sovellusten kehittäjiä tuomalla lisää rajapintoja, joilla puhelimen toimintoja voidaan ohjata. Kuvasta 2.2 nähdään S60-alustan eri kerrokset.



Kuva 2.2 S60-alustan eri kerrokset

Käytettiin alustana sitten matkapuhelinta tai tietokonetta, niin peliohjelmointi ja pelien yleinen rakenne on periaatteiltaan samanlaista ohjelmointikielestä riippumatta. Matkapuhelimille pelejä tai sovelluksia ohjelmoitaessa ja suunniteltaessa täytyy kuitenkin joitakin asioita ottaa erityisesti huomioon, joten seuraavat seikat kannattaa pitää mielessä:

- rajoitettu muistin määrä
- näyttöruudun koko ja resoluutio
- prosessorin teho.

Edellä mainittujen asioiden lisäksi kannattaa myös suunnitella kuinka käsitellään esimerkiksi tulevat puhelut tai akun loppuminen. Jos verrataan matkapuhelimia ja tietokoneita, niin matkapuhelimissa muistin määrä on erittäin pieni. On siis erittäin tärkeää, että muistia käytetään tehokkaasti ja kaikki varattu muisti tyhjennetään heti kun sitä ei enää tarvita. Peleissä yleensä grafiikka ja bittikartat kuluttavat paljon muistia, joten grafiikan muistinkulutusta kannattaa optimoida esimerkiksi bittikartan värisyvyyttä vähentämällä.

Symbiania on aikaisempina vuosina vaivannut dokumentaation puute, ja ehkä juuri sen takia sillä on ollut vähän huono ja vaikeasti lähestyttävän maine ohjelmistoke-

hittäjien keskuudessa. Nykyään dokumentaatiot ovat paljon parempia ja internetistä löytyy paljon ohjeita ja apuja kehittäjille. Yksi suurimmista ongelma pelien kehittämisessä Symbian-laitteille on se, että puhelimia on useita erilaisia. Näyttöjen resoluutio voi vaihdella mallin mukaan, joten esimerkiksi pelien bittikartat, ja joskus jopa pelin logiikka, pitää tehdä eri malleille sopivaksi. Tekemäni MobTactics-pelin grafiikat on suunniteltu vain puhelimille, joiden resoluutio on 240x320 pikseliä. Ehkäpä jos Symbian-sovelluksilla olisi yhtä hyvä ja suosittu jakelukanava kuin Applen iPhonen käyttämä App Store, niin Symbian C++:lla tehtyjä pelejä olisi markkinoilla enemmän.

3 MOBTACTICS-SUUNNITTELU

Tässä kappaleessa kerron pelin ideasta, rakenteesta sekä arkkitehtuurin suunnittelusta.

3.1 Peli-idea

Strategia- ja taktiikkapeleissä yhden pelin kesto voi helposti olla useita tunteja. Niin pitkät pelit eivät oikein sovellu matkapuhelimella pelattavaksi, joten halusin tehdä MobTactics-pelistä nopeatempoisen antaen kuitenkin pelaajille mahdollisuuden käyttää erilaisia strategioita ja taktiikoita vastustajan voittamiseen. Sopivan pieni ja intensiivinen pelimaailma sekä yksinkertaiset pelityypit mahdollistavat sen, että yksi pelikerta kestä kauan. Pelin suunnittelin kaksinpeliksi, mutta lopuksi lisäsin mahdollisuuden pelata myös tekoälyä vastaan.

3.2 Joukkueet ja vuoropohjaisuus

Pelissä on kaksi joukkuetta, SWAT ja Terrorist, joita pelaajat ohjaavat. Kummassakin joukkueessa on pelaajien valinnan mukaan joko 4 tai 6 pelihahmoa. Joukkueita ohjataan vuoropohjaisesti, eli kun pelaaja on ohjannut oman joukkueensa pelihahmoja halunsa mukaan, niin tulee toisen pelaajan vuoro. Kummallakin joukkueella on omat tehtävänsä pelityypin mukaan ja se joukkue, joka suorittaa tehtävänsä ensin, voittaa pelin.

3.3 Pelityypit

MobTactics-pelissä on kolme erilaista pelityyppiä:

- Eliminate
- Escape
- Capture.

Eliminate-pelityyppi on yksinkertaisin. Siinä voitaa se joukkue, joka ensiksi tappaa toisen joukkueen pelihahmot.

Escape-pelityypissä pelaajan, joka ohjaa Terrorist-joukkuetta, on tavoitteena saada omat pelihahmonsa kentällä näkyvälle Escape-alueelle. SWAT-joukkueen tehtävänä on yrittää pysäyttää Terrorist-joukkue. Terrorist-joukkue voittaa pelin jos se saa puolet joukkueen pelihahmoista Escape-alueelle, kun taas SWAT-joukkue voittaa, jos puolet Terrorist-joukkueesta kuolee.

Capture-pelityypissä pelaajien ohjaamien joukkueiden tehtävänä on kerätä kaksi esinettä kentältä. Kun molemmat esineet ovat joukkueen hallussa, esineitä kantavien pelihahmojen pitää päästä Escape-alueelle, jotta joukkue voittaa pelin.

Taktisessa mielessä haastavin pelityyppi on Capture. Siinä pelaajan täytyy esimerkiksi harkita, lähettääkö koko joukkueen hakemaan esineitä vai lähettääkö osan pelihahmoista suojaamaan Escape-aluetta. Erilaisia taktisia mahdollisuuksia on paljon, mikä tuo peliin haastavuutta ja pelattavuutta. Lisää haastavuutta Capture-pelityyppiin saa asetuksista, joissa voi valita, näkyvätkö esineet kentällä koko ajan vai ovatko esineet piilossa, kunnes pelihahmo löytää ne. Kuvassa 3.1 vasemmalla puolella nähdään huoneessa oleva kerättävä esine ja oikealla puolella Escape-alue.

Kuva 3.1 Kerättävä esine ja Escape-alue

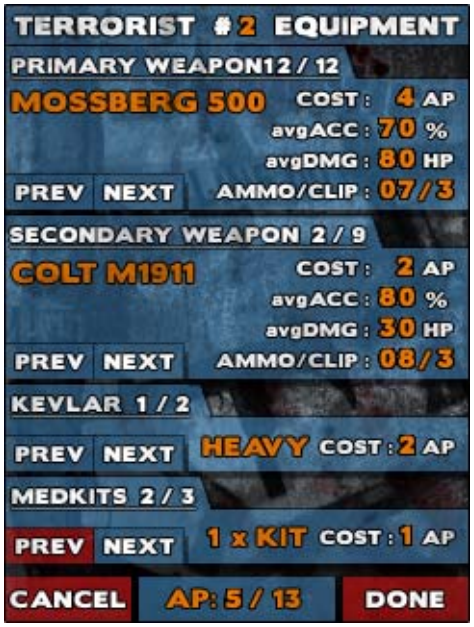
3.4 Toimintapisteet

Toimintapisteitä käytetään pelissä eri toimintojen toteutukseen kuten liikkumiseen ja ampumiseen. Pelaajan jokaisella pelihahmolla on käytössään 13 toimintapistettä, joilla ”ostetaan” hahmojen aseet ja varusteet ennen pelin alkua. Jäljelle jääneet toimintapisteet ovat pelihahmon käytössä pelin aikana. Koska kaikki toiminnot pelissä kuluttavat toimintapisteitä, vaaditaan pelaajalta strategista ja taktista suunnittelua tavoitteen saavuttamiseksi.

3.5 Aseet ja varusteet

Pelissä on valittavana useita erilaisia aseita ja varusteita, joita ovat lääkepakkaukset ja erilaiset suojaliivit. Aseiden ja varusteiden hankkimisesta jääneet toimintapisteet ovat siis pelihahmon käytössä pelin aikana. Pelihahmo voi kantaa kahta asetta kerrallaan. Ensisijainen ase on tehokkaampi ja toissijaista asetta käytetään yleensä vain silloin kun ensisijaisesta aseesta loppuu panokset. Pelin aikana pelihahmo voi vaihtaa ensisijaisen aseensa toissijaiseksi tai päinvastoin, mikä maksaa 3 toimintapistettä.

Eri aseiden eroja ovat optimaalinen kantomatka, panosten määrä lippaassa, tehokkuus ja tarkkuus sekä se, kuinka monta toimintapistettä sillä ampuminen maksaa. Suojaliivejä on kahta tyyppiä, kevyet ja raskaat. Kevyet kuluttavat vähemmän toimintapisteitä, mutta ne eivät suojaa niin hyvin kuin enemmän toimintapisteitä kuluttavat raskaat suojaliivit.



Kuva 3.2 Pelihahmojen varusteiden valinta -valikko

Heti alussa pelaajalta vaaditaan strategista suunnittelua, eli hänen pitää päättää, millaisella taktiikalla hän haluaa peliä pelata. Esimerkiksi Escape-pelityypissä Terrorist-joukkeen ei välttämättä kannata hankkia kaikille pelihahmoille liikaa varusteita, jotta liikkumiseen jää enemmän toimintapisteitä ja mahdollisuudet päästä nopeammin Escape-alueelle paranevat.

3.6 Näkökenttä

Pelihahmon näkökentällä (engl. line of sight) tarkoitetaan kaikkia interaktiivisia asioita, joita hahmo näkee pelimaailmassa. Näitä interaktiivisia asioita ovat siis vihollisen pelihahmot sekä Capture-pelityypissä kerättävät esineet. Pelihahmon näkökenttää estävät kiinteät objektit, kuten esimerkiksi seinät. Jos pelihahmon näkökentässä ei ole vihollista, niin vihollista ei piirretä ruudulle. Pelaajan valinnan mukaan Capture-pelityypissä kerättäviä esineitä ei myöskään piirretä ruudulle, jos pelihahmo ei niitä näe.

3.7 Toiminnot

Kamera

Pelaaja voi vuorollaan liikuttaa kuvakulmaa eli kameraa vapaasti ympäri pelimaailmaa. Kamera on myös animoitu niin, että kun pelaaja selaa omia pelihahmojaan, niin kamera kohdistetaan automaattisesti valitun pelihahmon kohdalle. Kamera kohdistetaan myös vihollishahmoihin, jos niitä tähdätään tai ammutaan.

Liikkuminen

Pelihahmoa liikutettaessa ilmestyy pelimaailmaan näkyviin numeroita, jotka ilmoittavat, kuinka monta toimintapistettä liikkuminen kuhunkin ruutuun maksaa. Tätä numeroitua aluetta kutsutaan liikkumisalueeksi, ja se sisältää vain ne ruudut, joihin pelihahmon käytettävissä olevilla toimintapisteillä on mahdollista liikkua.

Ampuminen

Pelihahmon näkökentässä täytyy olla vihollishahmoja, jotta voidaan ampua. Tämä on mahdollista , kun näytölle ilmestyy punainen nuoli vihollishahmon ylle ja kamera kohdistetaan valittuun viholliseen. Ampumisen tehokkuus riippuu pelaajan hahmon ja vihollisen välimatkasta, välillä olevista mahdollisista esteistä, vihollisen suojaliivien tehokkuudesta sekä käytettävästä aseesta. Ampuminen myös kuluttaa toimintapisteitä ja kulutus vaihtelee aseen mukaan.

Aseen lataus ja vaihtaminen

Jos aseen lippaasta loppuu panokset, niin se täytyy ladata uudella lippaalla. Lippaiden määrä ja lippaassa olevien panosten määrä vaihtelevat aseiden mukaan. Aseen lataus kuluttaa 3 toimintapistettä. Pelaaja voi vaihtaa pelihahmolleen käteen ensisijaisen tai toissijaisen aseen. Aseen vaihtaminen kuluttaa 3 toimintapistettä.

Vartiointi

Pelaaja voi asettaa pelihahmonsa vartioimaan. Tämä tarkoittaa sitä, että kun vihollisen hahmo kävelee omalla pelivuorollaan vartioivan hahmon näkökenttään, niin vartioiva hahmo ampuu vihollista. Pelihahmon vartiointiin asettaminen kuluttaa 4 toimintapistettä.

Lääkepakkaukset

Pelihahmo voi vuorollaan käyttää mahdollista lääkepakkausta, joka antaa pelihahmolle energiaa 25 energiapistettä (HP). Lääkepakkauksen käyttö kuluttaa 4 toimintapistettä.

3.8 Arkkitehtuuri

Pelin luokkasuunnittelua aloin tehdä, kun pelin idea ja toiminnot olivat selvillä. Kuitenkin toteutuksen edetessä tuli lisää toimintoja, joita halusin mukaan peliin, joten alkuperäinen luokkasuunnitelma piti osittain uusia. Peli on jaettu neljään pääluokkaan eli moduuliin, jotka ovat logiikkamoduuli, pelidatamoduuli, animaatiomoduuli ja grafiikkamoduuli. Moduulit käyttävät omia apuluokkia tai tietotyyppejä tehtäviensä suorittamiseen.

Logiikkamoduuli hoitaa pelin logiikan. Se esimerkiksi käsittelee näppäinten painallukset pelitilojen mukaan sekä etsii pelihahmon näkökentän ja liikkumisalueen. Logiikkamoduulissa yleensä käsitellään pelidatamoduulin sisältämää tietoa. Logiikkamoduulissa tehdään instanssit pelidatamoduulista, grafiikkamoduulista sekä animaatiomoduulista.

Pelidatamoduuli sisältää pelin ja pelimaailman tietoja. Sitä voisi kutsua eräänlaiseksi tietokannaksi. Se sisältää esimerkiksi pelin säännöt ja kaikki pelihahmot, joita voidaan käsitellä julkisilla funktioilla. Animaatiomoduuli ja grafiikkamoduuli saavat viittauksen pelidatamoduuliin, jotta pelimaailman tietojen käsittely ja vastaanottaminen olisi suoraviivaista ja helppoa.

Animaatiomoduulin tarkoituksena on hoitaa animaatioiden päivitys. Se myös tarpeen mukaan käynnistää uusia animaatioita.

Grafiikkamoduuli hoitaa pelin grafiikan piirtämisen näytölle eri pelitilanteiden mukaan. Siellä myös animoidaan kameran liike tarpeen mukaan.

Pelin suoritus on tapahtumapohjaisesta. Peli reagoi käyttäjän näppäinten painalluksiin eli syötteisiin ja tekee toimintoja pelin tilojen perusteella. Yleensä pelaajan an-

tama syöte muuttaa pelin tilaa jollain tavalla ja silloin päivitetään logiikkaa ja näyttöä. Pelissä ei siis käytetä ajastimia logiikan tai näytön jatkuvaan päivitykseen muuten kuin animaatioissa, kameran liikkeessä ja tekoälyn päivityksessä. Liitteessä A on kuva, jossa näkyy pelin kaikki luokat sekä niiden väliset yhteydet.

4 MOBTACTICS-TOTEUTUS

Tässä kappaleessa kerron MobTactics -pelin toteutuksesta. Esittelen arkkitehtuurin toteutuksen sekä tärkeimpien ominaisuuksien toimintaperiaatteet.

4.1 Arkkitehtuurin toteutus

Pelissä on erilaisia tiloja, joiden perusteella esimerkiksi näppäinten painallukset ja näytölle piirtäminen käsitellään. Esimerkiksi päätilat, jotka on esitelty kuvassa 4.1, kertovat, ollaanko pelitilanteessa vai valikoissa.

```
enum TMainStates
{
EInGame,                          // InGame-state
EMainMenu,                        // 'Main'-menu
EOptionsMenu,                     // 'Options'-menu
EInstructionsMenu,                // 'Instructions'-menu
ECreditsMenu,                     // 'Credits'-menu
EGameTypeMenu,                    // 'Game type'-menu
ESwatUnitSetupMenu,               // 'Swat unit setup'-menu
ESwatMemberEquipmentMenu,         // 'Swatmember equipment'-menu
ETerroristUnitSetupMenu,          // 'Terrorist unit setup'-menu
ETerroristMemberEquipmentMenu,    // 'Terrorist equipment'-menu
EPlayersMenu,                     // Selecting how many players
ETeamSelectionMenu                // Selecting team
};
```

Kuva 4.1 Pelin päätilat

Pelitilat (engl. InGameState) ovat tiloja, jotka ovat käytössä pelaamisen aikana. Tilat vaihtelevat esimerkiksi sen perusteella, onko pelaaja liikuttamassa pelihahmoaan vai ampumassa. Pelitilat on esitelty kuvassa 4.2.

```
enum TInGameStates
{
  EUnitViewState,              // Player is viewing unitmember
  EFreeCameraState,            // Player is using free camera
  EChangeTurn,                 // Player changes turn
  EAnimationState,             // Animation is running
  EUnitMemberAimMode,          // Player is aiming
  EUnitMemberMoveMode,         // Player is moving unitmember
  ECameraAnimation,            // Camera is moving
  EInGameActionsMenu,          // InGame-actions menu
  EInGameQuitMenu,             // InGame quit-menu
  EStartRound,                 // New round is about to start
  EGameOver,                   // Game is over
  EInGameQuitMenuOptions,      // InGame options
  EInGameQuitMenuInstructions, // InGame instructions
  EAiCalculation               // AI calculating
};
```

Kuva 4.2 Pelitilat

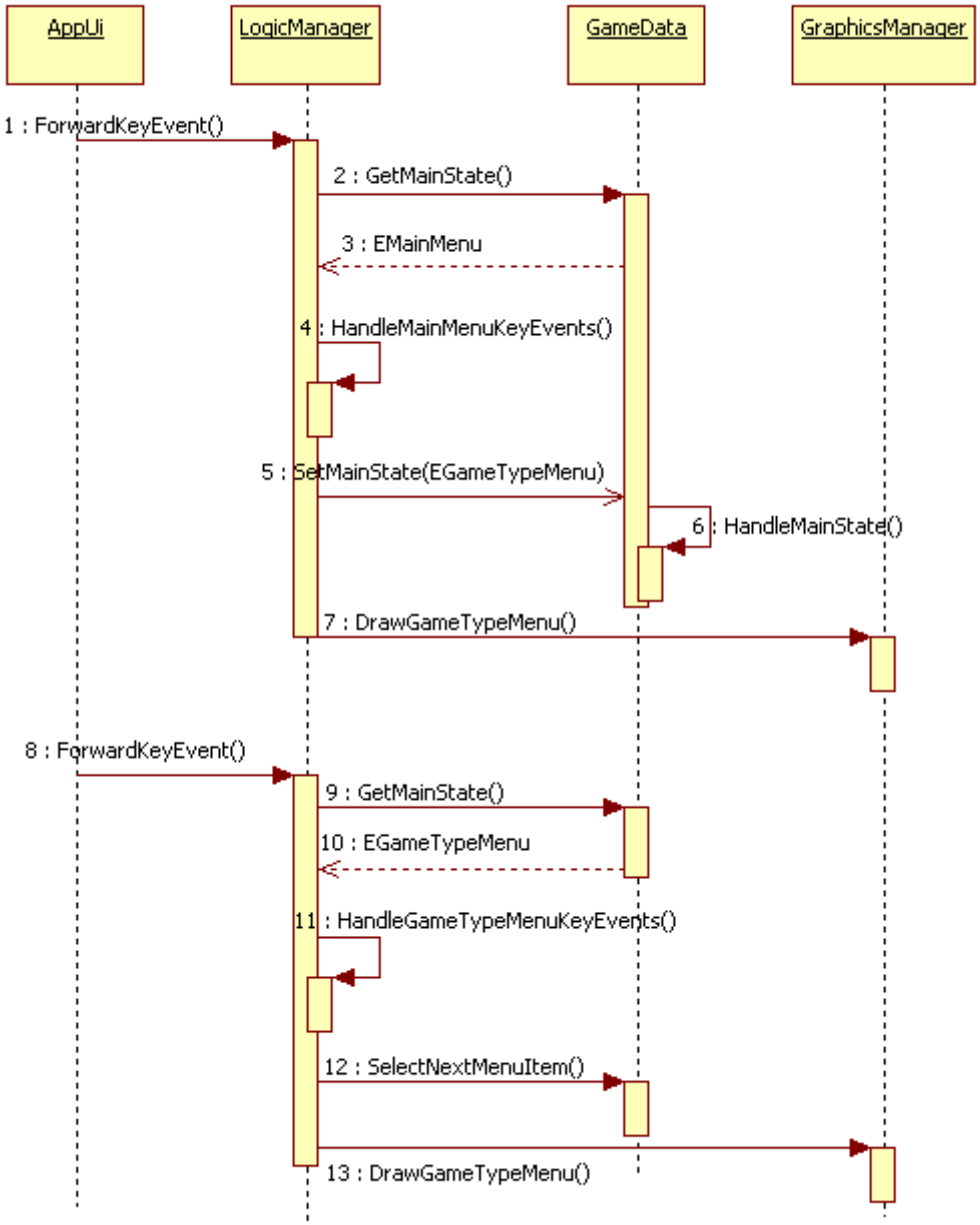
Animaatiotilat kertovat, mikä animaatio on käynnissä tai minkä animaation täytyy käynnistyä. Ne on esitelty kuvassa 4.3.

```
enum TAnimationStates
{
  EAnim_UnitMemberShooting,           // Unitmember shooting
  EAnim_GuardShooting,                // Guard shooting
  EAnim_EnemyGetDamage,               // Enemy unitmember damage
  EAnim_DisplayUnitMemberDamageValue, // Displays damage value
  EAnim_DisplayEnemyDamageValue,      // Displays damage value
  EAnim_UnitMemberGetDamage,          // Unitmember takes damage
  EAnim_UnitMemberMoving,             // Unitmember is moving
  EAnim_UnitMemberDies,               // Unitmember dies
  EAnim_EnemyDies,                    // Enemy unitmember dies
  EAnim_OffsetCamera                 // Animates camera movement
};
```

Kuva 4.3 Animaatiotilat

Pelaajan näppäintenpainallukset ohjataan pelitilan mukaiselle funktiolle, jossa ne käsitellään ja tehdään käyttäjän haluamat toiminnot. Kun näppäin on käsitelty, niin piirretään pelitilan mukaan ruudulle grafiikka.

Kaaviossa 4.1 on sekvenssikaaviona esitetty tilanne, jossa pelaaja menee päävalikosta (Main Menu) pelityyppivalikkoon (Game Type). Tämän jälkeen pelaaja liikuttaa pelityyppivalikossa valintaa alaspäin.



Kaavio 4.1 Sekvenssikaavio esimerkkitilanteesta

1. Tässä tapauksessa oletetaan, että päätilana on EMainState ja olemme ”Main Menu” -valikossa ”Start Game” -valinnan kohdalla. Käyttäjä painaa matkapuhelimessa nappia ”5”, joka lähetetään ForwardKeyEvent-funktion avulla LogicManager-moduuliin.
2. LogicManager kysyy GameData-moduulilta, missä päätilassa ollaan.
3. GameData-moduuli palauttaa päätilan LogicManager-moduulille.

4. LogicManager-moduuli ohjaa päätilan perusteella käyttäjän syötteen HandleMainMenuKeyEvents-funktiolle.
5. HandleMainMenuKeyEvents-funktiossa huomataan, että ”Start Game” on valittuna ja käyttäjän syöte on ”5”, joten pelin päätila pitää vaihtaa ”Game Type” -valikoksi.
6. Kun päätila vaihdetaan, GameData-moduulin täytyy käsitellä uusi tila HandleMainState-funktiossa. Tässä tapauksessa siellä tuhotaan ”Main Menu” -valikko ja tehdään ”Game Type” -valikko.
7. Kun uusi valikko on tehty, se piirretään ruudulle käyttäen GraphicsManager-moduulia. Tämän jälkeen pelin koodia ei enää suoriteta, ennen kuin käyttäjä antaa uuden syötteen.
8. Käyttäjä antaa uuden syötteen ”8”, joka tarkoittaa sitä, että hän haluaa liikuttaa valikossa valintaa alaspäin. Syöte lähetetään LogicManagerille.
9. Kysytään päätila GameData-moduulilta.
10. Vastaanotetaan päätila EGameTypeMenu.
11. Päätilan perusteella kutsutaan HandleGameTypeMenuKeyEvents-funktiota.
12. Syötteen perusteella pyydetään GameData-moduulia liikuttamaan valintaa valikosta.
13. GraphicsManager piirtää uuden tilanteen ruudulle ja tämän jälkeen ohjelma jää taas odottamaan käyttäjän uutta syötettä.

Kaaviossa 4.1 esitelty tilanne kuvaa hyvin pelin toimintatapaa ja arkkitehtuuria. Pelin logiikkaa ja grafiikkaa päivitetään vain kun käyttäjä on antanut syötteen. Poikkeuksena ovat hahmojen animaatiot ja kameran liike, jolloin päivitys tapahtuu ajastimen avulla eikä käyttäjän syötteitä lueta ollenkaan.

4.2 Pelimaailma ja ruututyypit

Pelimaailman koko on 1280 x 1200 pikseliä, ja siitä piirretään näytölle kerrallaan 240 x 320 pikselin kokoinen alue. Vaikka pelimaailma on graafisesti yhtenäinen alue, niin pelikoodissa sitä käsitellään 25 x 25 pikselin kokoisina alueina, joita kutsun ruuduiksi. Tämä ruudukkoon jako on olennaista, koska jokainen ruutu sisältää tiedon siinä olevasta objektista. Tätä tietoa kutsun ruututyypiksi. Erilaisia ruututyyppejä on 8 kappaletta, ja ne on esitelty taulukossa 4.1.

Ruututyyppi	Käsittely	Kuvaus
EWall	- Ei voi kävellä läpi - Ei voi nähdä läpi - Ei voi ampua läpi	Korkeita kiinteitä objekteja, esimerkiksi seinät, hyllyt, automaatit.
ESeeAnd-ShootThrough	- Ei voi kävellä läpi - Voi nähdä läpi - Voi ampua läpi	Matalia kiinteitä objekteja, esimerkiksi pöydät, tuolit.
EEmpty	- Voi kävellä läpi - Voi nähdä läpi - Voi ampua läpi	Tyhjä ruutu, johon pelihahmot voivat kävellä.
ESwatMember, ETerroristMember	- Ei voi kävellä läpi - Voi nähdä läpi - Voi ampua läpi	Pelihahmot sijaitsevat näissä ruuduissa.
EEscape	- Voi kävellä läpi - Voi nähdä läpi - Voi ampua läpi	Escape- ja Capture-pelityypissä käytettävä ruutu, johon peli päättyy.
EObjectiveItem1, EObjectiveItem2	- Voi kävellä läpi - Voi nähdä läpi - Voi ampua läpi	Capture-pelityypissä käytettävä ruutu, jossa sijaitsee kerättävä objekti.

Taulukko 4.1 Eri ruututyyppien esittely sekä niiden käsittelytapa

Ohjelmakoodissa näitä ruututyyppejä käytetään melkein kaikissa pelin toiminnoissa.

4.3 Joukkueiden tekeminen

Pelihahmon rakentamisessa tärkeimmät luokat on CUnitMember ja TWeapon. CUnitMember-luokka sisältää tietoja esimerkiksi pelihahmon energiasta ja sijainnista, kun taas TWeapon-luokka sisältää tietoja aseista. Pelihahmo tehdään antamalle CUnitMember-luokan rakentajalle parametreinä hahmon joukkue ja järjestysnumero, minkä jälkeen pelihahmo alustetaan vakioarvoilla. Kuvissa 4.4 ja 4.5 on esitelty osa pelihahmon ja aseen alustamisesta.

```
void CUnitMember::ConstructL()
{
    iIsSelected = EFalse;
    iActionPoints = KActionPoints;
    iIsAlive = ETrue;
    iHealth = KMaxHealth;
    iArmorMultiplier = KArmorMultiplier;
    iMedKits = KStartingMedKits;
    iKevlar = EKevlarHeavy;

    if(iTeam == ESwat)
    {
       iPrimaryWeapon.InitPrimaryWeapon(EPRIWEP_COLTM4);
       iSecondaWeapon.InitSecondaryWeapon(ESECWEP_BERETTA92F);
```

Kuva 4.4 Pelihahmon alustusta oletusarvoilla

```
case EPRIWEP_COLTM4:
            iBitmapID = EMbmMobtacticsWeapon_name_coltm4;
            iBitmapMaskID = EMbmMobtacticsWeapon_name_coltm4_mask;
            iApConsume = KColtM4ApConsume;
            iAverageDamage = KColtM4AverageDamage;
            iClipSize = KColtM4ClipSize;
            iClipCount = KColtM4ClipCount;
            iWeaponType = EAssaultRifle;
            iMaxDamage = KColtM4MaxDamage;
            iMaxDistance = KColtM4MaxDistance;
```

Kuva 4.5 Aseen alustus

CUnitMember-luokka sisältää useita julkisia funktioita, joilla saadaan haettua ja muokattua tietoja pelihahmosta sekä hänen aseistuksestaan. TWeapon- ja CUnitMember-luokat sisältävät myös pelihahmojen ja aseiden bittikartat, jotta niiden piirtäminen näytölle on nopeaa ja suoraviivaista.

CGameData-luokka omistaa kaksi CUnitMember-tyyppistä RPointerArraytä jotka sisältävät joukkuiden pelihahmot: iSwatUnit ja iTerroristUnit. Näiden kahden lisäksi CGameData-luokassa on myös samantyyppiset RPointerArrayt eli iCurrentUnit ja iEnemyUnit, jotka osoittavat pelivuoron mukaan joukkueiden varsinaisiin pelihahmoihin. Tein näin siksi, että koodissa olisi helpompi käsitellä joukkueita pelivuorosta riippumatta.

4.4 Liikkumisalue ja polun löytäminen

Koska liikkuminen kuluttaa toimintapisteitä, tarvitaan algoritmi, joka etsii lyhimmän reitin hahmon lähtöpaikasta valittuun paikkaan pelimaailmassa. Koska toimintapisteitä ei ole rajattomasti, täytyy algoritmin etsiä vain sellaiset paikat, jonne on mahdollista liikkua käytössä olevilla toimintapisteillä. Pelissä liikkuminen viereiseen ruutuun maksaa yhden toimintapisteen. Kuvassa 4.6 on nähtävissä liikkumisalue sekä liikkumiseen käytettävä toimintapisteiden kulutus.

Kuva 4.6 Liikkumisalue sekä toimintapisteiden kulutus

Algoritmeja, jotka etsivät lyhyimmän reitin paikasta A paikkaa B, kutsutaan path-finding-algoritmeiksi. Näitä algoritmeja käytetään esimerkiksi reittihakupalveluissa sekä peleissä tekoälyn ohjaamisessa paikasta toiseen. Koska MobTactics-pelissä pelihahmon toimintapisteet rajoittavat liikkumisen suhteellisen pienelle alueelle, päätin soveltaa Dijkstran-algoritmin perusteita liikkumisalueen laskemiseen. /2/

Liikkumisalueen etsintä on perusteiltaan aika suoraviivainen toimenpide:

- Lähtöpaikka on ruutu, jossa pelihahmo sijaitsee. Merkitään lähtöruutu tarkistetuksi ja katsotaan ruututyypin perusteella, voidaanko liikkua ruutuihin, jotka sijaitsevat lähtöruudun sivuilla tai ylä- ja alapuolella. Jos voidaan liikkua, niin merkitään ne tarkistamattomiksi.

- Käydään läpi kaikki tarkistamattomiksi merkityt ruudut ja etsitään myös niiden sivuilta ja ylä- ja alapuolilta ruutuja, joita ei ole tarkistettu ja joihin on mahdollista liikkua.
- Tätä jatketaan niin pitkään kunnes tullaan siihen pisteeseen asti, että tarkistamattomia ruutuja ei enää ole

Jotta ruutuun voi liikkua, siinä ei saa olla mitään kiinteää objektia, kuten seinää tai toista pelihahmoa. Tietenkin täytyy myös ottaa huomioon se, onko pelihahmolla tarvittava määrä toimintapisteitä liikkumista varten. Liikkumisalueen toteutuksessa on käytetty kahta luokkaa, TPathNode ja TPathFinder. Kuvassa 4.7 esiteltävää TPathNode-luokkaa käytetään kuvaamaan yhtä pelimaailman ruutua liikkumisaluetta etsittäessä.

```
class TPathNode
{
public:
    /* Constructor, parameters are:
     * current block position,
     * current block actionpoint cost,
     * parentblock actionpoint cost,
     * parentblock position
     * */
     TPathNode(TPoint aPosition, TInt aBaseCost,
           TInt aParentTotalCost,  TPoint aParentPosition);

     virtual ~TNode();           // Virtual Destructor

     TPoint iPosition;           // Current block position
     TInt iBaseCost;             // Current block actionpoint cost
     TInt iParentCost;           // Parentblock actionpoint cost
     TInt iTotalCost;            // Basecost + parentcost
     TPoint iParent;             // Parentblock position
};
```

Kuva 4.7 TPathNode-luokan esittely

TPathNode-luokkaan kuuluu seuraavat tiedot:

- iPosition: nykyisen ruudun sijainti
- iBaseCost: nykyisen ruudun toimintapisteiden kulutus
- iParentCost: ”isäntä”-ruudun, eli ruudun josta nykyiseen ruutuun tultiin, toimintapisteiden kokonaiskulutus
- iTotalCost: toimintapisteiden kulutus kokonaisuudessaan, eli nykyisen ruudun ja ”isäntä”-ruudun toimintapisteiden kulutuksen summa
- iParentPosition: ”isäntä”-ruudun sijainti

TPathNode-luokkaa tarvitaan, jotta voidaan pitää yllä tietoa siitä, kuinka monta toimintapistettä on kulunut tultaessa tarkasteltavaan ruutuun. Samalla saadaan myös rakennettua polku niistä ruuduista, josta tarkasteltavaan ruutuun päästiin. Tätä polkua käytetään myöhemmin pelihahmon liikuttamisessa.

```
class TPathfinder
{
public:
   TPathfinder();  // Constructor
   virtual ~TPathfinder(); // Virtual destructor

   /* Searches and returns movement area,
   * takes unitmember position and actionspoints as paramenter */
   RArray<TNode> GetMovementArea(TPoint aUnitMemberPosition,
                                 TInt aActionPoints);

   // Returns path of blocks for unitmember to move
   RArray<TPoint> GetPath(TPoint aTarget);

private:
   TMapBlockType iMap[KMapSizeX][KMapSizeY]; // Game map

   void DoPath(TPoint aTarget); // Does path for unitmember

   // Finds nodes(blocks) for unitmember
   void FindNodes(TPoint aUnitMemberPosition);

   // Searches nodes from array
   TBool CompareNodes(RArray<TNode> aArr,TInt aNodeX,Tint aNodeY);

   void InvertPath();            // Inverts unitmember path

   RArray<TNode> iOpenList;      // List that holds unchecked nodes
   RArray<TNode> iClosedList;    // List that holds checked nodes
   RArray<TPoint> iPath;         // Path for unitmember
   TInt iMovementPoints;         // Unitmembers actionpoints
};
```

Kuva 4.8 TPathFinder-luokan esittely

TPathFinder-luokassa, esitelty kuvassa 4.8, liikkumisalue etsitään kutsumalla julkista funktiota GetMovementArea(), jolle annetaan parametrina pelihahmon sijainti ja toimintapisteet. GetMovementArea()-funktio kutsuu sisäistä funktiota FindNodes(), joka toteuttaa liikkumisalueen etsimisen. Ruutujen merkitsemiseen käytetään kahta TPathNode-tyyppissä listaa. Toinen lista, iOpenNodes, sisältää tiedon ruuduista, joita ei ole vielä tarkistettu. Lista iClosedNodes taas sisältää tiedon jo tarkistetuista ruuduista, joihin on mahdollista liikkua.

Kun pelihahmon liikkumisalueen etsintä aloitetaan, tehdään uusi TPahtNode-tyyppinen olio, nimeltään startNode, jolle annetaan parametreina pelihahmon sijainti, toimintapisteiden kulutukset sekä ”isäntä-ruudun” sijainti:

```
TNode startNode(aUnitMemberPosition.iX, aUnitMemberPostion.iY,
        0, 0, aUnitMemberPosition.iX, aUnitMemberPosition.iY);
```

Lisätään startNode tarkastamattomien ruutujen listaan iOpenList, jota käydään läpi silmukassa:

```
iOpenList.Append(startNode);
```

Tehdään uusi TPathNode-tyyppinen olio listassa olevan kulloinkin tarkasteltavan ruudun perusteella:

```
for(TInt i = 0; i < iOpenList.Count(); i++)
{
   TNode currentNode = iOpenList[i];
```

Katsotaan, että tarkasteltavan ruudun toimintapisteiden kokonaiskulutus on pelihahmon toimintapisteiden sallimissa rajoissa sekä katsotaan pelimaailman ruututyypit sisältävästä iMap-listasta tarkasteltavan ruudun oikealla puolella olevan ruudun tyyppi (iMap[currentNode.iX + 1][currentNode.iY]). Tarkistetaan myös että se sijaitsee pelimaailman sisäpuolella:

```
if(currentNode.iTotalCost <= iMovementPoints )
{
 if((iMap[currentNode.iX + 1][currentNode.iY] == EEmpty
  || iMap[currentNode.iX + 1][currentNode.iY] == EObjectiveItem1
  || iMap[currentNode.iX + 1][currentNode.iY] == EObjectiveItem2
  || iMap[currentNode.iX + 1][currentNode.iY] == EEscape)
  && currentNode.iX + 1 <= KAmountOfXBlocks  )
```

CompareNodes()-metodilla tarkastetaan, että oikean puoleinen ruutu ei ole kummassakaan listassa. Tarkasteltavan ruudun oikealla puolella sijaitseva ruutu lisätään tarkastamattomien ruutujen listaan. Sille annetaan parametrina sen sijainti, toimintapisteiden kulutus, toimintapisteiden kokonaiskulutus tähän mennessä sekä tarkasteltavan ruudun sijainti, joka siis on ”isäntä-sijainti”:

```
if(!CompareNodes(iClosedList,currentNode.iX +1, currentNode.iY)
&& !CompareNodes(iOpenList,currentNode.iX +1, currentNode.iY))
{
  iOpenList.Append(
  TPathNode( TPoint(currentNode.iX + 1, currentNode.iY ), 1,
  currentNode.iTotalCost, TPoint(currentNode.iX, currentNode.iY))
  );
}
```

Tämä toimenpide tehdään myös tarkasteltavan ruudun vasemmalle sekä ylä- ja alapuolelle. Näin listaan saadaan uusia ruutuja, joiden ympäriltä katsotaan samalla tavalla muita mahdollisia ruutuja, joihin pelihahmo voi liikkua. Lopuksi tarkasteltava ruutu lisätään tarkistettujen ruutujen listaan ja silmukka jatkaa pyörimistään kunnes tarkastamattomia ruutuja ei ole enää jäljellä.

Tällä tavoin saadaan pelihahmon liikkumisalue, joka on siis iClosedList-listan sisältö. Lista kertoo kaikki mahdolliset ruudut, joihin hahmo voi liikkua, sekä myös kuinka monta toimintapistettä näihin ruutuihin liikkuminen maksaa.

4.5 Näkökenttä

Näkökentän etsimiseen käytin Bresenhamin viiva-algoritmia (engl. Breseham's line algorithm) /3/. Algoritmilla voi etsiä ruudukkoon piirretyn viivan kaikki ruudut, jotka viiva lävistää. Kun pelihahmon näkökenttää etsitään, niin algoritmi piirtää kuvitteellisen viivan hahmon ruudusta tarkasteltaviin kohteisiin, eli vihollisiin ja tarpeen mukaan kerättäviin esineisiin. Jos viiva törmää sellaiseen ruutuun jossa on jokin kiinteä objekti, niin pelihahmo ei näe tarkasteltavaa kohdetta. Kuvassa 4.9 on esitelty esimerkki.

Kuva 4.9 Esimerkki näkökentän toimintaperiaatteesta

4.6 Kamera

Koska näytölle voidaan kerralla piirtää vain 240 x 320 pikselin kokoinen alue, täytyi toteuttaa kamera. Sen avulla matkapuhelimen näytölle piirsin kuvat jotka sijaitset tuon alueen ulkopuolella. Käytännössä kamera on kaksi muuttujaa, jotka kertovat kuinka monta pikseliä x- ja y-suunnassa näytölle piirrettävä kuva poikkeaa origosta. Origolla tarkoitetaan matkapuhelimen näytön vasenta ylänurkkaa ja pelimaailmassa se myös on vasen ylänurkka. Aina bittikarttoja piirrettäessä täytyy niiden sijaintiin lisätä tämä poikkeama. Tällä tavalla saadaan vaikutelma, että pelimaailman yllä on kamera joka liikkuu, vaikka tosiasiassa vain bittikartat liikkuvat. Kamera on myös animoitu, jolloin se liikkuu automaattisesti omia pelihahmoja selattaessa tai kun vihollishahmoa tähdätään.

Kameran liike valmistellaan InitializeCamera()-metodissa. Siellä tarkistetaan, että kameran liike lopetetaan silloin, kun pelihahmo on näytön keskellä ja ettei pelimaailman ulkopuolisia alueita piirretä. Kun kamera on alustettu, niin ajastimen avulla kutsutaan kuvassa 4.10 näkyvää AnimateCamera()-metodia, joka toteuttaa kameran liikkeen. Metodia kutsutaan niin pitkään kunnes kamera on tullut haluttuun loppupisteeseen.

```
TBool CGraphicsManager::AnimateCamera()
{
// If x-coordinate of camera offset is near zero, stabilize it
if(Abs(iCameraAnimEnd.iX -  iCameraOffset.iX) < 1)
           iCameraOffset.iX = iCameraAnimEnd.iX;

// Else move camera along x-coordinate
else
iCameraOffset.iX -= (iCameraOffset.iX - iCameraAnimEnd.iX) / 4;

// If y-coordinate of camera offset is near zero, stabilize it
if(Abs(iCameraAnimEnd.iY -  iCameraOffset.iY) < 1)
           iCameraOffset.iY = iCameraAnimEnd.iY;

// Else move camera along y-coordinate
else
iCameraOffset.iY -= (iCameraOffset.iY - iCameraAnimEnd.iY) / 4;

// If both coordinates match, end camera animation
if( iCameraOffset.iX == iCameraAnimEnd.iX
     && iCameraOffset.iY == iCameraAnimEnd.iY)
{
return ETrue;
}

// Continue animation
return EFalse;
}
```

Kuva 4.10 AnimateCamera()-metodin toiminta

4.7 Grafiikka ja animaatiot

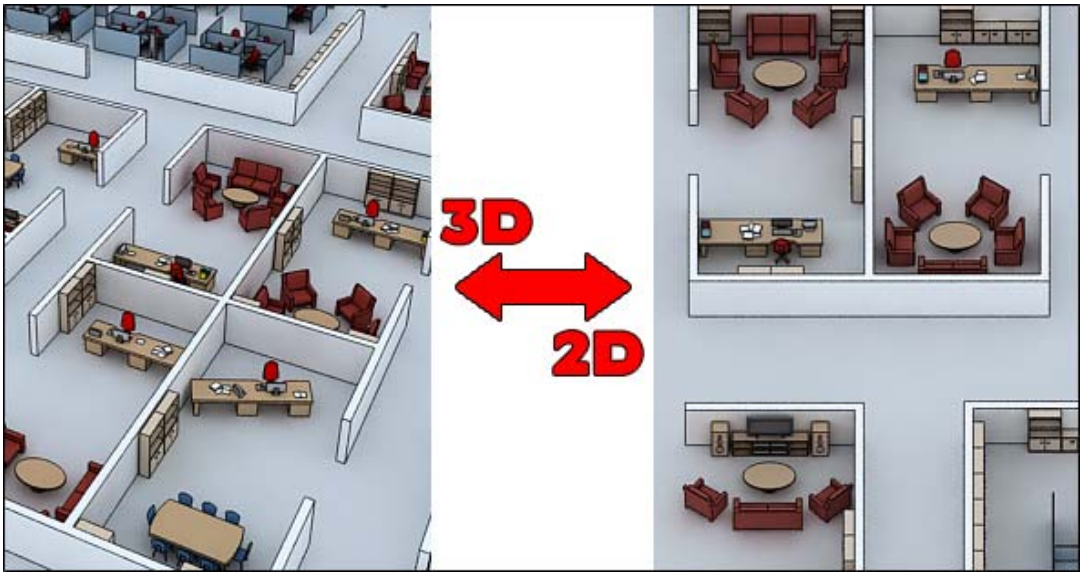
Symbianissa näytölle voidaan piirtää esimerkiksi käyttämällä S60-alustan ikkunapalvelinta (Window Server, WSERV). Tämä yleisesti käytetty tapa on suoraviivainen ja helppokäyttöinen, mutta mahdollisesti hidas. Jos useampi sovellus haluaa yhtä aikaa piirtää näytölle, niin ikkunapalvelimen tehtävänä on päättää, mikä sovellus saa piirtää mihinkin kohtaan näytölle. Tästä seuraa viivettä piirtämisessä. Peleissä on yleensä tärkeää, että liikkeet ja animaatiot pyörivät sujuvasti, joten ikkunapalvelimen käyttöä on syytä välttää mahdollisten hitauksien takia. Ikkunapalve-

lin voidaan kuitenkin ohittaa ja bittikartat piirtää suoraan näytölle käyttämällä Direct Screen Access (DSA) – luokkia. Jos käyttöjärjestelmän täytyy piirtää näytölle jokin yllättävä tapahtuma, kuten esimerkiksi saapuva puhelu, niin kehittäjän on huolehdittava siitä, että DSA pysäytetään ja annetaan ikkunapalvelimen hoitaa piirtäminen. Peli käyttää hyväksi myös tuplapuskurointia, jossa piirrettävä bittikartta piirretään ensiksi ns. taustapuskuriin josta se on nopeampi piirtää ruudulle. Tämä mahdollistaa sen, että nopeasti liikkuvat bittikartat eivät repeydy ruudulle piirrettäessä. /4/

Kuvasta 4.11 nähdään, että esimerkiksi pelihahmon bittikartta sisältää monta erilaista kuvaa hahmon liikkumisesta joista jokainen kuva on yksi animaation ruutu, eli frame. Kun pelihahmo animoidaan, niin ajastimen avulla pelihahmon bittikartasta piirretään haluttu osa ruudulle ja saadaan vaikutelma liikkuvasta pelihahmosta. Tietysti pelihahmon fyysistä sijaintia pitää myös päivittää.

Kuva 4.11 SWAT-pelihahmon kävelyanimaation bittikartta

Pelikenttä on 1280 x 1200 pikselin kokoinen alue, joka on jaettu 20:neen 240 x 320 pikselin kokoiseen bittikarttaan. Aluksi aloin piirtämään kenttää kaksiulotteisesti kuvankäsittelyohjelmalla, mutta huomasin että se on vaivalloista eikä siitä tullut kovin hyvännäköistä. Siksi päätin tehdä kentän 3D-mallinnuksena, joka onnistuu minulta huomattavasti paremmin, ja samalla sain myös helposti tehtyä perspektiivin, joka ei ole suoraan ylhäältä päin. Kun 3D-malli väreineen oli valmis, niin tein siihen värikorjauksia ja lisäsin yksityiskohtia kuvankäsittelyohjelmalla.



Kuva 4.12 Pelin kenttä 3d-grafiikasta 2d-grafiikkaan

Suurimmat haasteet grafiikan teossa oli matkapuhelimen näytön rajoitettu koko. Valikoita tehdessä jouduin monta kertaa aloittamaan grafiikan suunnittelun alusta, jotta kaikki tarpeellinen mahtuisi ruudulle. Kuitenkin sain mielestäni tehtyä sopivan selkeän käyttöliittymän sekä monipuoliset, vaikkakin ahtaat, valikot (kuva 4.13).

Kuva 4.13 Pelihahmon toimintavalikko

4.8 Valikot

En halunnut käyttää pelissäni Symbianin valmiita valikkokomponentteja, koska ne eivät visuaalisesti sopineet pelin muuhun grafiikkaan. Tein kaksi apuluokkaa, CMenu ja CMenuItem, jotta omien valikoiden tekeminen ja niiden käsittely olisi helppoa. CMenuItem-luokka kuvaa valikossa olevaa yhtä komponenttia, eli nappia, ja se sisältää tiedot napin bittikartoista ja sijainnista. CMenu-luokka on valikon rajapinta ja se omistaa CMenuItem-tyyppisen listan, joka sisältää tiedot kaikista valikossa olevista napeista.

```
class CMenu : public CBase
{
public:
            static CMenu* NewL(TMenuType aMenuType,
            MMenuObserver&aObserver);

           void SelectNextItem();
           void SelectPrevItem();
           TInt SelectedItem();
           .
           .
           .

private:
           TInt iItemCount;
            RPointerArray<CMenuItem> iMenuItems;

            .
            .
```

Kuva 4.14 CMenu-luokka

Valikko tehdään antamalla CMenu-luokan rakentajalle parametriksi valikon tyyppi. Tyypin perusteella katsotaan kuinka monta komponenttia valikkoon tulee, mitkä bittikartat niillä on ja mihin ne sijoitetaan ruudulla. Kuvasta 4.15 nähdään esimerkki siitä miten ”Game Type”-valikkoon luodaan komponentit.

```
case EMenu_GameTypeMenu: {
iMenuItems.Append(CMenuItem::NewL(
            EMbmMobtacticsGametypemenu_button_eliminate,
           EMbmMobtacticsGametypmenu_button_eliminate_selected,
           KEliminateButtonPosition ));

iMenuItems.Append( CMenuItem::NewL(
           EMbmMobtacticsGametypemenu_button_escape,
           EMbmMobtacticsGametypemenu_button_escape_selected,
           KEscapeButtonPosition ));

iMenuItems.Append( CMenuItem::NewL(
           EMbmMobtacticsGametypemenu_button_capture,
           EMbmMobtacticsGametypemenu_button_capture_selected,
           KCaptureButtonPosition ));
```

Kuva 4.15 Game Type –valikon nappien luonti

CMenuItem-luokan rakentajalle annetaan parametreinä napin kahden bittikartan nimet sekä sijainti. Kahta bittikarttaa käytetään kuvaamaan napin eri tiloja, eli onko se valittuna vai ei. Kun CMenu-luokalta pyydetään valitsemaan seuraava nappi, niin se vaihtaa napin bittikartat sopiviksi. Kuvasta 4.16 nähdään, että nappeja piirrettäessä käydään läpi kaikki valikon napit, joista saadaan suoraan napin sijainti ja bittikartta.

```
// Loop through all buttons in menu (Eliminate, Escape, Capture)
for(TInt i=0;i < iGameData->iCurrentMenu->GetMenuItemCount(); i++)
{
// Draw item on screen
aBc->BitBlt(iGameData->iCurrentMenu->GetMenuItemPosition(i),
            iGameData->iCurrentMenu->GetMenuItemBitmap(i));
  }
```

Kuva 4.16 Nappien piirto näytölle

Kuvassa 4.17 on valmis ”Game Type”-valikko johon on piirretty ”Eliminate”, ”Escape” ja ”Capture”-nappien lisäksi myös muuta grafiikkaa.

Kuva 4.17 Valmis Game Type -valikko

4.9 Äänet

MobTactics-pelissä äänet on toteutettu käyttämällä Symbianin Multimedia Framework:ia (MMF). MMF sisältää useita luokkia erilaisten äänitiedostojen soittamiseen ja käsittelyyn. Luokka mahdollistaa laitteen tukemien ääniformaattien soittamisen suoraan tiedostosta ja se on erittäin helppo toteuttaa. Äänien soittamisen totetutin CSoundManager-luokassa (kuva 4.18) ja se piti periyttää MMdaAudioPlayerCallback-luokasta. CSoundManager-luokan pitää toteuttaa MMdaAudioPlayerCallback-luokan metodit MapcInitComplete ja MapcPlayComplete jotka hoitavat äänen toistamiseen liittyvät virhekoodit.

```
class CSoundManager: public CBase, public MMdaAudioPlayerCallback
{
public:
    // from MMdaAudioPlayerCallback
   void MapcInitComplete(TInt aError,
                  const TTimeIntervalMicroSeconds& aDuration);

   // from MMdaAudioPlayerCallback
   void MapcPlayComplete(TInt aError);

   void PlaySound(TSoundIds aSound);

private:
   CMdaAudioPlayerUtility* iCurrentSoundPlayer;
   CMdaAudioPlayerUtility* iShootHitPlayer;
   CMdaAudioPlayerUtility* iShootBurstPlayer;
```

Kuva 4.18 CSoundManager-luokka

CSoundManager-luokan ConstructL-metodissa äänet alustetaan kuvassa 4.19 olevalla tavalla.

```
_LIT(KShootBurstSoundFile, "\\resource\\apps\\burst.wav");
_LIT(KShootHitSoundFile, "\\resource\\apps\\hit.wav");
TFileName ShootHitSoundFile(KShootHitSoundFile);
TFileName ShootBurstSoundFile(KShootBurstSoundFile);

iShootBurstPlayer =
    CMdaAudioPlayerUtility::NewFilePlayerL(ShootBurstSoundFile, *this);
iShootHitPlayer =
    CMdaAudioPlayerUtiity::NewFilePlayerL(ShootHitSoundFile, *this);
```

Kuva 4.19 CSoundManager-luokan alustus

Äänet soitetaan kutsumalla CSoundManager-luokan julkista metodia PlaySound() ja antamalla sille parametrinä soittevan äänen tunnus (kuva 4.20)

```
void CSoundManager::PlaySound(TSoundIds aSound)
{
    if (iState!=EReady)
       return;

    StopSound();

     switch(aSound)
     {
       case EShootBurst:
            iCurrentSoundPlayer = iShootBurstPlayer;
            break;

       case EShootHit:
            iCurrentSoundPlayer = iShootHitPlayer;
             break;
     };

     if(iCurrentSoundPlayer)
     {
        iState=EPlaying;
        iCurrentSoundPlayer->Play();
     }
}
```

Kuva 4.20 Äänen toistaminen

4.10 Tekoäly

Kuten aikaisemmin totesin, MobTactics suunniteltiin kaksinpeliksi. Kun peli oli muuten valmis, päätin kokeilla yksinkertaisen tekoälyn tekemistä, jotta peliä voisi pelata myös yksin. Koska en pystynyt käyttämään tekoälyn toteutukseen paljoa aikaa enkä myöskään aikaisemmin ollut tekoäly-ohjelmointia tehnyt, niin tekoäly-vastustaja on aika yksinkertainen eikä se kykene tekemään mitään erityisiä taktisia päätöksiä.

Pelivuoron ollessa tekoälyllä, se jokaisella pelihahmollaan toistaa seuraavanlaista rutiinia niin pitkään kunnes sen toimintapisteet ovat loppuneet:

- Jos energiaa on alle puolet jäljellä, niin käytä lääkepakkaus jos hahmo kantaa sellaista
- Jos aseesta on panokset loppumassa, niin lataa ase. Jos ei ole lippaita jäljellä, niin vaihda toiseen aseeseen.
- Jos vastustajan pelihahmo on näkyvissä, niin ammu.
- Liikuta pelihahmoa.

Tekoälyn pelihahmot navigoivat kentällä reittipisteiden perusteella. Jokaiselle tekoälyn ohjaamalle pelihahmolle arvotaan pelityypin ja joukkueen perusteella jokin etukäteen määritetyistä reittipiste-joukoista. Liikkuessaan tekoälyn pelihahmo etsii lyhimmän reitin ensimmäiselle reittipisteelle, johon saavuttuaan taas seuraavalle reittipisteelle ja niin edelleen. Lyhimmän reitin etsimiseen vaikuttavat muiden pelihahmojen sijainti kentällä, joten vaikka reittipiste-joukot on etukäteen määritelty, niin itse reitti on harvoin samanlainen kuin muilla pelikerroilla. Tekoälyn pelihahmojen reitit voivat myös muuttua pelin aikana. Esimerkiksi Capture-pelityypissä, tekoäly voi ohjata hahmonsa suoraan Escape-alueelle, jos kaikki kerättävät esineet ovat löytyneet.

Koska tekoälyn reitit, aseet ja varusteet arvotaan satunnaisesti, ei voida sanoa että tekoäly olisi kovin älykäs tai taktinen. Kuitenkin tekoäly osaa liikkua eri pelityypin mukaan, ampua vihollisia, käyttää varusteita ja toteuttaa pelin voittoon vaatimat tehtävät, joten päätin jättää sen mukaan peliin.

5 KEHITYSIDEAT

Vaikka sain peliin tehtyä alun perin suunnittelemani toiminnot sekä lisäksi vielä yksinkertaisen tekoälyn, niin jatkokehitysideoita löytyy aina. Mielestäni suurin puute tällä hetkellä on se, että pelissä on vain yksi kenttä. Tosin jos peliin tekisi lisää kenttiä, niin se kasvattaisi myös pelin tiedostokokoa aika lailla bittikarttojen koon takia. Tämä vaatisi sitten erilaisen toteutuksen pelikentän tekoon, esimerkiksi käyttämällä ruutupohjaista-grafiikkaa. Se taas tekisi mielestäni grafiikasta paljon yksitoikkoisemman ja itseään toistavan, mutta vaatisi vähemmän bittikarttoja. Animaatioihin pitäisi saada lisää ruutuja, jotta ne näyttäisivät paremmilta ja liikkuisivat sulavammin. Myös tekoälyn voisi tehdä järkevämmäksi. Kentässä ei myöskään ole huoneissa minkäänlaisia ovia, joiden aukaiseminen ja tiirikoiminen toisivat lisää haastavuutta peliin. Pelihahmoilla voisi olla myös jonkinlaisia valo- tai tainnutuskranaatteja. Myös kentällä satunnaisesti juoksevat siviilit olisi hauska lisä.

6 YHTEENVETO

Mielestäni pääsin haluamiini tavoitteisiin ja sain tehtyä sellaisen pelin kuin halusin. Suunniteluun olisin voinut käyttää enemmän aikaa, luokista tuli aika isoja ja ne voisi jakaa pienempiin kokonaisuuksiin. Suurimmat haasteet oli toteuttaa pelihahmon lyhyimmän reitin löytäminen ja näkökenttä. Sekin johtui siitä, että käyttämäni algoritmit vaikuttivat teoriassa aluksi erittäin monimutkaisilta, mutta pienen tutustumisen jälkeen ne tuntuivat loogisilta ja helppokäyttöisiltä. Myös grafiikka, tai sen saaminen hyvännäköiseksi, osoittautui välillä haasteelliseksi joten käytin siihen paljon aikaa. Tavoitteenani oli myös oppia lisää Symbian-ympäristöstä. Opin kuinka käytetään Direct Screen Access-luokkia Window Serverin ohittamiseen sekä opin kuinka sovelluksiin lisätään ääntä Multimedia Framework:ia käyttäen.

Vaikka peli on periaatteiltaan aika yksinkertainen, koodirivejä tuli silti tuhansia. Paremmalla suunnittelulla ja järkevämmällä olio-ajattelulla, rivejä saisi karsittua paljon pois. Pelin rakenne on kuitenkin sen verran yksinkertainen, että siihen on helppo lisätä erityyppisiä tehtäviä tai erilaisia aseita ja varusteita. Huomasin kuitenkin perusteellisen suunnittelun tärkeyden ja aion ottaa sen huomioon tulevissa projekteissa.

Vaikka peli valmistui suunnitelmieni mukaisesti, niin tarkoituksenani on tulevaisuudessa toteuttaa edellisessä kappaleessa mainitut jatkokehitysideat.

LÄHTEET

1. Symbian homepage [www-sivu] Saatavissa: http://www.symbian.com/about/fastfacts.html

2. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, ja Clifford Stein. Introduction to Algorithms. MIT Press and McGraw-Hill, 2001

3. Max K. Agoston. Computer Graphics and Geometric Modeling: Implementation and Algorithms. Springer, 2005

4. Jo Stichbury. Games On Symbian: A Handbook for Mobile Development. Wiley, 2008

5. Ben Morris. The Symbian OS Architecture Sourcebook: Design And Evolution of a Mobile Phone OS. Wiley, 2007

LIITE A: Pelin luokat ja niiden väliset riippuvuudet