Panu Korhonen

Moninpelimuodon kehittäminen kaupalliselle pelimoottorille

Opinnäytetyö

Syksy 2011

Tekniikan yksikkö

Tietojenkäsittelyn koulutusohjelma

Seinäjoen ammattikorkeakoulu
SEINÄJOKI UNIVERSITY OF APPLIED SCIENCES

SEINÄJOEN AMMATTIKORKEAKOULU

Opinnäytetyön tiivistelmä

Koulutusyksikkö:	Tekniikan yksikkö
Koulutusohjelma:	Tietojenkäsittelyn koulutusohjelma
Suuntautumisvaihtoehto:	Sovellustuotannon suuntautumisvaihtoehto
Tekijä:	Panu Korhonen
Työn nimi:	Moninpelimuodon kehittäminen kaupalliselle pelimoottorille
Ohjaaja:	Markku Lahti
Vuosi: 2011	Sivumäärä: 76 Liitteiden lukumäärä: 12

Opinnäytetyön tarkoituksena oli luoda uusi, alkuperäisistä poikkeava, moninpelimuoto ARMA 2: Combined Operations -tietokonepeliin. Pelimuodon kohderyhmänä olivat pienet kaveripiirit, jotka pelaavat yhdessä verkon välityksellä. Pelimuodossa pyrittiin hyödyntämään alkuperäisten pelimuotojen parhaita puolia ja siitä kehitettiin joka pelikerralla vaihtuva, mutta perusteiltaan yksinkertainen, kokonaisuus. Työn suunnittelu sekä toteutus jaettiin Jesse Schellin esittämiin pelin neljään osa-alueeseen, ja suunnittelussa otettiin huomioon useissa peleissä käytettyjä toimivia ratkaisuja, joita työssä sovellettiin.

Työn graafisessa suunnittelussa käytettiin hyödyksi Microsoft Excel -taulukkolaskentaohjelmaa ja toteutus suoritettiin pelin omalla tehtäväeditorilla sekä Notepad++-tekstinkäsittelyohjelmalla. Työn toteutus jaettiin vaiheisiin, jotka sisältävät eri osa-alueiden ohjelmoinnin sekä ne yhdistävän testauksen.

Avainsanat: tietokonepeli, muokkaus, moninpeli, pelimuoto

SEINÄJOKI UNIVERSITY OF APPLIED SCIENCES

Thesis abstract

Faculty:	School of Technology
Degree programme:	Business Information Technology
Specialisation:	Software Production
Author:	Panu Korhonen
Title of the thesis:	The development of a multiplayer game mode for a commercial game engine
Supervisor:	Markku Lahti
Year: 2011	Number of pages: 76 Number of appendices: 12

The goal of this thesis was to create a new multiplayer game mode for ARMA 2: Combined Operations computer game. The focus client groups of the game mode were small circles of friends who play together via the internet or local area network. An effort was made to utilize working elements from the original game modes. The game mode was also developed to offer different experiences on every game session but to be simple and easy to learn. The design and implementation were split into four basic game elements suggested by Jesse Schell. Proven game design concepts from different games were taken into consideration in the design and adapted into the new game mode accordingly.

The design portion of the thesis was accomplished by utilizing Microsoft Excel spreadsheet software. The implementation was done with the game's own mission editor and Notepad++ source code editor. The implementation was split into phases which include programming phases for different game elements and a testing phase where all the elements are merged.

Keywords: computer game, modification, multiplayer, game mode

SISÄLTÖ

Kuvio- ja taulukkoluettelo

Käytetyt termit ja lyhenteet

A2: CO	Työn pohjana käytetty FPS-peli, joka on rakennettu Real Virtuality 3 -pelimoottorin alustalle. A2: CO on lyhennelmä nimestä ARMA 2: Combined Operations.
Alustustiedosto	Alustustiedostossa määritellään kyseessä olevan pelin tai pelimuodon sisältö luokkarakenteena.
Ammus	Kts. Ase.
Ase	Aseilla yksiköt voivat vahingoittaa peliympäristössä olevia toisia yksiköitä sekä jopa itseään. Aseet käyttävät lippaita, jotka sisältävät ammuksia. A2: CO sisältää n. 150 asetta, joista jokainen käyttäytyy eri tavalla.
Capture The Flag	CTF, eli lipunryöstö, on moninpelimuoto, jossa pelaajat on jaettu yleensä kahteen ryhmään. Molemmilla ryhmillä on tukikohta, jossa ryhmän lippu sijaitsee. Pelaajien tehtävänä on ryöstää vastaryhmän lippu tuomalla se omaan tukikohtaansa.
Coop	Kts. Cooperation.
Cooperation	Moninpelimuoto, jossa pelaajat asetetaan samalle osapuolelle ja toimivat yhdessä päihittääkseen vastassa olevat, tekoälyn ohjaamat, viholliset. Cooperation -pelimuodosta käytetään lyhennettä coop.

Deathmatch	Moninpelimuoto, jossa jokainen pelaaja on asetettu eri osapuolelle ja pelaajien päämääränä on tuhota toiset pelaajat.
Ensimmäisen persoonan ammuntapeli	Kts. FPS-peli.
Esine	Esineitä ovat esimerkiksi kiikarit, kello, kartta ja radio. Jokaisella esineellä on oma pelimekaaninen käyttötarkoituksensa.
Estetiikka	Yksi pelin neljästä peruselementistä. Estetiikkaa on pelin kaikki ihmisen aisteihin sekä tunteisiin liittyvä sisältö, kuten kuvat, videot, ääni sekä tunnelma.
FPS-peli	FPS- eli ensimmäisen persoonan ammuntapelit ovat toimintapelien alagenre. FPS-peleissä pelaaja ohjaa hahmoaan kolmiulotteisessa peliympäristössä ja yrittää tuhota vastustajiaan hänelle tarjolla olevilla aseilla. Pelaaja näkee ympäristön hahmonsa silmin ja ohjaa hahmoaan näppäimistöllä sekä hiirellä.
Graafinen käyttöliittymä	Pelaaja vuorovaikuttaa pelin kanssa käyttäen graafista käyttöliittymää. Graafiseen käyttöliittymään luetaan kaikki sisältö, jonka pelaaja näkee, mutta joka ei ole peliympäristössä.
Hahmo	Työn pelimuodossa pelaajan valitsema yksikkötyyppi. Eri hahmoilla on pelimuodossa erilaisia ominaisuuksia.
Istunto	Kts. Peli-istunto.

Kampanja Yksinpelimuoto, jossa pelaaja kulkee pelin teemaa tukevaa ja ennalta käsikirjoitettua tarinaa kohtauksesta toiseen.

Kenttä Määrää pelimuodon tapahtumapaikan sekä alkutilanteen. Yksi pelimuoto voi sisältää useita kenttiä.

Kokemus Kts. Kokemuspiste.

Kokemuspiste Työn pelimuodossa hahmot saavat kokemuspisteitä mm. tuhoamalla vihollisia. Kokemuspisteillä saavutetaan tasoja.

Komentosarjakieli Kts. SQF.

Komentotiedosto SQF-kieltä sisältävä tiedosto, jonka tulkkaamisen pelimoottori aloittaa pelimuodon käynnistyksessä.

Lipas Kts. Ase.

Lokaali Muuttuja tai tapahtuma, joka esiintyy vain yhdellä asiakaskoneella, eikä sen tilaa lähetetä verkon kautta palvelimelle tai muille asiakaskoneille.

Moninpelimuoto Pelimuoto, jonka pelaamiseen vaaditaan enemmän kuin yksi pelaaja. Useimmiten tämä tapahtuu verkon välityksellä jokaisen pelaajan käyttäessa omaa tietokonettaan.

Näkymä RV3:ssa kaksiulotteinen graafinen käyttöliittymä, joka voi olla pysyvä tai väliaikainen.

Objekti	Objekteja ovat kaikki pelaajien näkemät kolmiulotteiset elementit, jotka erottuvat peliympäristön maastosta.
Pelaaja	Henkilö, joka ohjaa yhtä hahmoa yhdeltä tietokoneelta.
Peli-istunto	Istunto on yhden kentän pelaaminen alusta loppuun tietyssä pelimuodossa.
Pelimekaniikka	Yksi pelin neljästä peruselementistä. Pelimekaniikka määrää pelin säännöt sekä tavoitteet. Sen avulla pelaajat ovat vuorovaikutuksessa pelin kanssa.
Pelimoottori	Pelimoottori on osa pelin teknologiaa ja silta siitä pelimekaniikkaan, estetiikkaan sekä tarinaan. Se on valmiiksi ohjelmoitu ohjelmarunko, joka määrittää miten pelin osa-alueiden sisällön esittäminen mahdollistetaan pelaajille. Pelimoottori sisältää komponentteja, jotka helpottavat esimerkiksi kaksi- tai kolmiulotteisen grafiikan piirtämistä näytölle, äänen toistamista monilta kanavilta ja fysiikan sekä tekoälyn mallinnusta.
Pelimuoto	Pelimuoto on pelin pelimoottoria sekä muita ominaisuuksia hyödyntävä kokonaisuus. Jokaisessa pelimuodossa on omat tavoitteensa sekä sääntönsä.
Peliympäristö	Pelin tapahtumia rajaava alue. A2: CO:n peliympäristöt ovat korkeuskartoista mallinnettuja ulkoilmaympäristöjä. Pelin kolmiulotteinen sisältö, kuten yksiköt sekä muut objektit sijoitetaan peliympäristöön.

Real Virtuality	Bohemia Interactiven kehittämä pelimoottorisarja. Tässä työssä käytettiin alustana A2: CO-peliä, joka hyödyntää Real Virtuality 3 -pelimoottoria.
RV3	Real Virtuality 3. Kts. Real Virtuality.
Simulaatio	Kts. Taistelusimulaatio.
SQF	A2: CO:ssa käytettävä komentosarjakieli. Pelimuodon ohjelmointi suoritetaan käyttäen SQF-kieltä.
Taistelusimulaatio	FPS-peli, joka sisältää muihin peleihin verrattuna suuren määrän realistista sisältöä sekä toimintoja. Simulaatio pyrkii mallintamaan oikean maailman olosuhteita mahdollisimman tarkasti.
Taito	Työn pelimuodossa taidon aktivointi antaa hahmolle uuden kyvyn tai koko pelaajaryhmälle mahdollisuuden ostaa uusia varusteita.
Taitopiste	Työn pelimuodossa tason saavutettuaan hahmo saa yhden taitopisteen. Pelaaja voi aktivoida taitopisteellä hahmolleen taidon.
Tarina	Yksi pelin neljästä peruselementistä. Tarina kattaa pelin juonikuviot sekä tapahtumat. Se voi olla ennalta käsikirjoitettu tai pelin itsensä luoma.
Taso	Työn pelimuodossa tietyn kokemuspistemäärän saavutettuaan hahmo ylenee tason. Hahmo saa tästä yhden taitopisteen.

Team deathmatch Deathmatch-pelimuodon variaatio, jossa pelaajat jaetaan ryhmiin ja ryhmien tarkoituksena on tuhota toiset ryhmät.

Teema Teema yhdistää pelin elementit kokonaisuudeksi ja ilmaisee, mistä peli kertoo ja mikä on sen tarkoitus.

Tehtävä Pelimuodon ilmentymä. Pelimuoto sanelee säännöt ja tavoitteet. Tehtävä antaa sille tapahtumapaikan.

Tehtäväeditori Pelin sisältämä työkalu, jolla objekteja asetellaan peliympäristöön.

Teknologia Yksi pelin neljästä peruselementistä. Teknologia sisältää ne välittäjämateriaalit sekä -keinot, joilla peli esitetään pelaajille. Se sallii sekä rajoittaa muiden elementtien käyttöä.

Tekoäly Joukko funktioita, jotka määrittävät tietokoneen hallussa olevien yksiköiden toiminnot. Käytetään myös viittaamaan em. yksiköihin.

Toimeksianto Työn pelimuodon osa, jossa pelaajat keräävät hahmoilleen kokemusta sekä rahaa.

Tukikohta Työn pelimuodossa pelaajien aloituspaikka. Tukikohdassa pelaajat valitsevat hahmonsa, varusteensa sekä seuraavaksi suoritettavan toimeksiannon.

Varuste Varusteisiin kuuluvat aseet, lippaat sekä esineet. Jokainen yksikkö voi kantaa saman määrän varusteita.

Yksikkö	Yksiköt ovat ympäristössä liikkuvia objekteja, jotka mallintavat ihmishahmoja. Tehtävää luodessa valitut yksiköt määritellään pelaajien ohjattavaksi. Muita yksiköitä ohjaa pelimoottorin tekoäly.
Yksinpelimuoto	Pelimuoto, jota voi pelata vain yksi pelaaja.

(Schell 2008 & Rabin ym. 2005.)

1 JOHDANTO

Tietokonepelit sisältävät monesti useita pelimuotoja, joista jokaisessa on omat tavoitteensa sekä sääntönsä. Ensimmäisen persoonan ammuntapeleissä pelimuodot ovat hyvin vakiintuneita ja tarjoavat vähän vaihtelua peli-istuntojen välillä. Työn tarkoituksena oli luoda uusi pelimuoto, joka tarjoaa jokaisella pelikerralla uusia kokemuksia sekä vaihtoehtoja. Ideoita haettiin suosituista FPS-peleistä, muista peligenreistä sekä pelialan kirjallisuudesta ja niitä pyrittiin soveltamaan tehokkaasti valitulle alustalle.

1.1 Työn tausta

Työn taustana oli halu kehittää monipuolinen pelimuoto, joka hyödyntäisi alustapelin sisältöä, mutta tarjoaisi pelaajille enemmän vapautta sekä vaihtoehtoja kuin yleisesti FPS-peleistä löytyvät pelimuodot.

1.2 Työn tavoitteet

Työn tavoitteena oli luoda uusi pelimuoto ARMA 2: Combined Operations -tietokonepeliin. Pelimuodon suunnittelussa otettiin huomioon kyseisen pelin alkuperäinen kohderyhmä sekä sitä tarkennettiin pieniin kaveripiireihin, jotka pelaavat yhdessä verkon välityksellä. Pelimuodosta pyrittiin kehittämään alkuperäisten pelimuotojen parhaita puolia soveltava, mutta joka pelikerralla uusia kokemuksia tarjoava, kokonaisuus. Työssä käytettiin hyväksi pelialan kirjallisuutta sekä useissa muissa peleissä hyväksi havaittuja ominaisuuksia.

1.3 Työn rakenne

Työn alussa esitetään teoriaa tietokonepeleistä, ensimmäisen persoonan ammuntapeleistä, niiden muokkaamisesta sekä muokkauksen vaikutuksista pelien kehittäjiin, pelien muokkaajiin sekä pelaajiin. Seuraavassa osassa esitetään työhön valitun pelin sekä sen pelimoottorin historiaa, pelin sisältämiä pelimuotoja

sekä uuden pelimuodon suunnitteluun vaikuttavia tekijöitä. Tämän jälkeen käydään läpi työn sunnitteluosa, joka on jaettu neljään osa-alueeseen: teknologia, pelimekaniikka, estetiikka sekä tarina. Toteutusosa on jaettu vaiheisiin, joiden lopussa suoritetaan testaus.

Työssä esitetään useita kuvioita sekä esimerkkipeleistä että työn edistymisen vaiheista. Työn ohjelmointi on suoritettu alusta loppuun ilman kolmannen osapuolen koodisisältöä, ja työn koodimäärä on noin 7000 riviä kommentteineen. Toteutusosassa mainitut funktiot ja skriptit löytyvät liitteinä työn lopusta.

2 TIETOKONEPELIT

Tässä osassa esitetään teoriaa tietokonepeleistä, ensimmäisen persoonan ammuntapeleistä, niiden muokkaamisesta sekä muokkauksen vaikutuksista pelien kehittäjiin, pelien muokkaajiin sekä pelaajiin.

2.1 Tietokonepelien rakenne

Tietokonepelit ovat teoriassa vain tietokoneohjelmia, jotka antavat käyttäjälleen pelien tuomia kokemuksia, kuten hauskuutta, ongelmanratkaisua, vallan tunnetta sekä sosiaalisuutta (Rabin ym. 2005, 71–82). Tietokonepelit eroavat muusta viihteestä, kuten kirjallisuudesta, musiikista sekä liikkuvasta kuvasta siten, että pelit voivat sisältää kaikkia edellä mainittuja. Nämä viihdemuodot yhdistettynä pelaajalle annettuun kontrolliin tietokoneohjelman ympäristössä määrittävät tietokonepelin. (Schell 2008, 10–38.) Tietokonepelit jaetaankin kehityksessä usein osa-alueisiin, jotka vastaavat muun viihteen sekä ohjelmistokehityksen asiantuntijoiden osaamisalueita, kuten tarinankerrontaa, äänimaailmaa, visuaalisuutta sekä ohjelmointia (Rabin ym. 2005, 856).

Schell (2008, 41–43) jakaa pelit neljään peruselementtiin, jotka ovat **teknologia**, **pelimekaniikka**, **estetiikka** sekä **tarina**. Näitä elementtejä tulee pitää yhtä tärkeinä ja niiden täytyy vahvistaa toisiaan sekä pelin teemaa. **Teknologia** sisältää ne välittäjämateriaalit sekä -keinot, joilla peli esitetään pelaajille. Se sallii sekä rajoittaa muiden elementtien käyttöä. **Pelimekaniikka** määrää pelin säännöt sekä tavoitteet. Sen avulla pelaajat ovat vuorovaikutuksessa pelin kanssa. Pelimekaniikka erottaa pelin muista viihdemuodoista, kuten kirjoista ja elokuvista, jotka eivät sellaista sisällä. **Estetiikkaa** on pelin kaikki ihmisen aisteihin sekä tunteisiin liittyvä sisältö, kuten kuvat, videot, ääni sekä tunnelma. Estetiikalla on suorin yhteys pelaajan pelikokemukseen. **Tarina** käsittää pelin juonikuviot sekä tapahtumat. Tarina voi olla ennalta käsikirjoitettu tai haarautuva kokonaisuus. Työn suunnitteluosa sekä toteutusosan vaiheet jaettiin neljään lukuun, joista jokainen käsittelee yhtä Schellin peruselementtiä.

Jokaisen tietokonepelin taustalla on pelimoottori, joka on osa pelin teknologiaa ja silta siitä pelimekaniikkaan, estetiikkaan sekä tarinaan. Pelimoottori on valmiiksi ohjelmoitu ohjelmarunko, joka määrittää miten muiden osa-alueiden esittäminen mahdollistetaan pelaajille. Pelimoottori sisältää komponentteja, jotka helpottavat esimerkiksi kaksi- tai kolmiulotteisen grafiikan piirtämistä näytölle, äänen toistamista monilta kanavilta, fysiikan sekä tekoälyn mallinnusta sekä verkkoliikennettä. Pelimoottori on usein testattu useilla laitteistokokoonpanoilla, jotta sen toiminta on käyttäjän laitteistosta riippumatta samanlainen. (Rabin ym. 2005, 267–271.) Yhden moottorin alustalle voidaan näin luoda useita erilaisia pelejä (Kuvio 1). Pelimoottori on siis jykevä kivijalka, joka mahdollistaa useiden talomallien rakentamisen sen päälle.

Kuvio 1. Kaksi erilaista peliä, jotka on luotu Source-pelimoottorin alustalle.

2.2 Ensimmäisen persoonan ammuntapelit

Ensimmäisen persoonan ammuntapelit, englanniksi first person shooters tai FPS, ovat toimintapelien alagenre. FPS-peleissä pelaaja ohjaa hahmoaan kolmiulotteisessa peliympäristössä ja yrittää tuhota vastustajiaan hänelle tarjolla olevilla aseilla. Pelaaja näkee ympäristön hahmonsa silmin ja ohjaa hahmoaan näppäimistöllä sekä hiirellä. (Rabin ym. 2005, 32.) Peleissä on useita pelimuotoja, jotka jaetaan yksinpelimuotoihin sekä internetissä tai lähiverkossa pelattaviin moninpelimuotoihin. Alla niistä on esitetty yleisimmät.

Kampanja. Useimmat nykyaikaiset FPS-pelit eivät sisällä kuin yhden yksinpelimuodon. Roolipeleistä johdettu kampanja-pelimuoto on suunniteltu nimenomaan yksinpeliä varten ja siinä on pelin teemaa tukeva tarina. Kampanja on yleisimmin jaettu elokuvamaisesti kohtauksiin, eli kenttiin, joissa pelaaja ohjaa päähahmoa. Kentän läpäistäkseen on pelaajan suoritettava siinä vaaditut tehtävät. Monet kampanjat ovat yksisäikeisiä, joten pelaaja ei toiminnoillaan voi vaikuttaa tarinan lopputulokseen.

Deathmatch. Tässä moninpelimuodossa jokainen pelaaja on asetettu eri osapuolelle ja pelaajien päämääränä on tuhota toiset pelaajat. Viimeiseksi jäänyt pelaaja on voittaja. Toisessa variaatiossa pelimuoto on jaettu eriin, joissa on aikaraja. Pelaaja saa pisteen tuhottuaan toisen pelaajan. Tuhouduttuaan pelaaja syntyy uudelleen peliympäristöön. Erän loputtua eniten pisteitä saanut pelaaja on voittaja. Tästä pelimuodosta on myös olemassa ryhmävariaatio, Team deathmatch, jossa pelaajat jaetaan ryhmiin ja ryhmien tarkoituksena on tuhota toiset ryhmät.

Capture The Flag. CTF, eli lipunryöstö, on moninpelimuoto, jossa pelaajat on jaettu yleensä kahteen ryhmään. Molemmilla ryhmillä on tukikohta, jossa ryhmän lippu sijaitsee. Pelaajien tehtävänä on ryöstää vastaryhmän lippu tuomalla se omaan tukikohtaansa. Ryöstön jälkeen lippu palautuu alkuperäiseen tukikohtaan ja pelin voittaja määritellään tietyn ryöstömäärän jälkeen.

Cooperation. Coop- eli cooperation-moninpelimuodossa pelaajat asetetaan samalle osapuolelle ja toimivat yhdessä päihittääkseen vastassa olevat, tekoälyn

ohjaamat, viholliset. Coop-pelimuodossa jokaisen kentän asetelma sekä päämäärät ovat joko pelin tai kentän tekijän määrittämiä. Tässä pelimuodossa saattaa olla siis erittäin paljon vaihtelua kenttien välillä, mutta yksittäinen kenttä tarjoaa hyvin vähän uudelleenpelattavuusarvoa, sillä sen alkutilanne sekä päämäärät ovat joka pelikerralla samat.

2.3 Tietokonepelien muokkaus

PC-pelit on usein luotu siten, että niiden muokkaaminen myöhemmin on mahdollista kehittäjien lisäksi myös pelaajien taholta. Kehittäjät luovat pelin kehitysvaiheessa työkaluja, joilla uuden sisällön luominen peliin on helppoa ja nopeaa. Pelin valmistuttua näitä työkaluja ei kehittäjien kannalta usein enää tarvita. Monet kehittäjät ovat kuitenkin huomanneet, että julkaisemalla käyttämänsä työkalut pelin ohella, lisäävät he pelin käyttöikää huomattavasti, sillä näin pelaajille annetaan mahdollisuus lisätä peliin sisältöä ja jakaa sitä kanssapelaajilleen. Tämän seurauksena pelien ympärille kehittyy niin sanottuja muokkausyhteisöjä, joissa pelaajat jakavat tekemäänsä uutta sisältöä ja keskustelevat siitä (Rabin ym. 2005, 858.) Keskustelu tapahtuu usein pelin virallisella keskustelulaudalla.

Kaupallisen tietokonepelin muokkaus luo erittäin hyvät mahdollisuudet aloittelevalle pelin kehittäjälle tutustua pelinkehitysprosessiin. Koska peli on jo toimiva kokonaisuus, voi aloittelija kokeilla haluamansa osa-alueen muokkausta ja tarkastella sen vaikutusta kokonaisuuteen. Yksittäinen pelin muokkaaja voikin käydä läpi niin pelin teknologiaan sekä pelimekaniikkaan kuin estetiikkaan ja tarinaankiin liittyvät toiminnot yksitellen.

3 ARMA 2: COMBINED OPERATIONS

Tässä osassa käydään läpi työhön valitun pelin sekä sen pelimoottorin historiaa, pelin sisältämiä pelimuotoja sekä uuden pelimuodon suunnitteluun vaikuttavia tekijöitä.

3.1 Pelimoottori

Vuonna 2001 Tšekkiläinen riippumaton peliyhtiö, Bohemia Interactive Studio, myöhemmin BIS, julkaisi debyytti-taistelusimulaatiopelinsä nimeltään Operation Flashpoint: Cold War Crisis. Tämä Microsoft Windows -alustainen PC-peli oli rakennettu yhtiön oman pelimoottorin alustalle, jota kutsuttiin nimellä **Real Virtuality 1**. Pelimoottori sisälsi innovatiivisia ominaisuuksia, kuten

- vapaan hiekkalaatikkoympäristön
- reaaliaikaisen varjojen laskennan
- fotorealistiset pintatekstuurit
- muuttuvat vuorokausi- sekä sääefektit
- muuttuvat ääniefektit esim. äänen nopeus
- osittain itsehallinnollisen tekoälyn
- helppokäyttöisen tehtäväeditorin sekä
- sisäänrakennetun, n. 350 komentoa sisältävän, komentosarjakielen nimeltään SQS. (Evolution of the Real Virtuality Engine 2008.)

Vuonna 2007 BIS julkaisi seuraavan PC-taistelusimulaationsa nimikkeellä ARMA: Armed Assault. Peli käytti yhtiön pelimoottorin seuraavaa versiota nimeltään **Real Virtuality 2**. Tämä versio sisälsi useita grafiikkaparannuksia sekä n. 400 uutta SQS-komentoa. (Evolution of the Real Virtuality Engine 2008.) Myös uusi komentokieli, nimeltään SQF, otettiin käyttöön vanhan rinnalle. SQF oli tehokkaampi sekä monipuolisempi, sillä se mahdollisti mm. funktioiden luomisen sekä arvojen palauttamisen kutsuvalle komentosarjalle. (SQS to SQF conversion, 2011.)

Real Virtuality 3, myöhemmin **RV3**, julkaistiin vuonna 2009 PC-pelin ARMA 2 -muodossa, jota vuonna 2010 täydensi lisäosa nimeltään ARMA 2: Operation Arrowhead. Samana vuonna julkaistiin nämä kaksi yhteen sitova pakettikokonaisuus ARMA 2: Combined Operations, myöhemmin A2: CO, joka mahdollisti molempien pelien sisällön hyödyntämisen samoissa ympäristöissä. Grafiikkapäivitysten lisäksi **RV3** mahdollisti edellistä suurempia ympäristöjä, sisälsi useita pelimekaanisia päivityksiä sekä uusia SQF-komentoja. (Interesting Facts And Figures 2008.) **RV3**:ssa SQF-komentosarjakieli syrjäytti vanhan SQS-kielen.

3.2 Pelimuodot

A2: CO sisältää neljä perus moninpelimuotoa, jotka ovat: Deathmatch, CTF, Warfare 2 sekä Cooperation. Pelimuodon ilmentymää kutsutaan tehtäväksi.

Warfare 2. ARMA 2:ssa debytoinut Warfare 2 -pelimuoto yhdistää team deathmatchin sekä strategiapelin elementtejä. Pelaajat on jaettu kahdelle osapuolelle, ja yksi pelaaja molemmilta osapuolilta toimii komentajana. Koko peliympäristö on jaettu asutuskeskuksiin, jotka tuottavat resursseja. Pelaajien päämääränä on vallata jokainen peliympäristössä sijaitseva asutuskeskus ja saattaa näin sen tuottamat resurssit oman komentajansa käyttöön. Komentajalla ei ole ohjattavaa yksikköä, vaan hän pelaa tehtävää peliympäristöä ylhäältä päin kuvaavasta näkymästä. Molemmilla osapuolilla on oma tukikohtansa, minne komentaja rakentaa osapuolensa resursseilla rakennuksia, jotka lisäävät pelaajien taistelukykyä. (Warfare 2 Manual 2010.) Warfare 2 -tehtävät tarjoavat joka pelikerralla jotain uutta, sillä tehtävän kulku on pelaajien päätettävissä. Pelimuodossa on kuitenkin monimutkaisuutensa vuoksi jyrkkä oppimiskäyrä, mikä vieraannuttaa aloittelevia pelaajia.

Cooperation. A2: CO:n coop-pelimuodossa jokaisen tehtävän asetelma sekä päämäärät ovat tehtävän tekijän erikseen laatimia ja ne usein mallintavat oikean maailman sotilastehtäviä, kuten aseman puolustamista, asemiin hyökkäystä, väijytystä tai kohteen tuhoamista. Tämä vetää pelaajia puoleensa, sillä A2: CO -peliä mainostetaan nimenomaan taistelusimulaationa (Arma 2: Combined Operations Features 2011). Coop on myös mainituista pelimuodoista selvästi

monipuolisin, sillä tehtäviä on saatavilla paljon ja ne eroavat toisistaan merkittävästi.

3.3 Uusi pelimuoto

Opinnäytetyön tavoitteena oli kehittää uusi, olemassa olevista poikkeava, moninpelimuoto A2: CO:lle. Pelimuodossa pyrittiin hyödyntämään muiden pelimuotojen parhaita puolia ja siitä suunniteltiin vaihteleva, mutta perusteiltaan yksinkertainen, kokonaisuus. Pelimuodon tyyliksi valittiin Coop-pelimuodon tapaan pelaajat vastaan tekoäly, sillä se korostaa joukkuepeliä ja soveltuu pienille pelaajamäärille. Pelimuodon haluttiin sisältävän

- vaihteleva vaikeustaso
- suuri uudelleenpelattavuusarvo
- pieni pelaajamäärä-vaatimus sekä
- alkuperäisen pelin simulaatio-henkisyys.

3.3.1 Vaikeustaso

Vaikeustaso on monissa Coop-tehtävissä vakio. Tehtävä saattaa olla tehty 1–10 pelaajalle, mutta vastustajien määrä pysyy samana vaikka pelaajia on yksi tai kymmenen. Tämä tylsistyttää tai turhauttaa pelaajia. Rabinin ym. (2005, 89) mukaan hyvä peli on suunniteltu niin, että sen vaikeustaso nousee ajan kuluessa, jotta pelaajan taitojen karttuessa hän ei ajaudu tylsyyteen tai turhautumiseen. Pelimuoto haluttiin luoda siten, että sen vaikeustaso määräytyy suhteessa pelaajien taitoon ja kokemukseen sekä pelaajamäärään.

3.3.2 Uudelleenpelattavuus

Coop-tehtävät johdattavat usein samaa kaavaa tehtävän alusta loppuun. Päämäärät sekä vastustajien koostumus ja sijoitus pysyvät samoina, mikä vaikuttaa negatiivisesti uudelleenpelattavuuteen. Schell (2008, 265) kirjoittaa, että

hyvä peli luo tapahtumasarjoja, jotka ovat niin mielenkiintoisia, että ihmiset haluavat kertoa niistä muille. Pelimuodosta päätettiin luoda vaihtuva kokonaisuus, joka tarjoaa jokaisella pelikerralla erilaisia tapahtumia sekä päämääriä.

3.3.3 Pelaajamäärä

Taistelusimulaationa A2: CO tarjoaa suuria peliympäristöjä sekä mahdollisuuden isoihin pelaajamääriin. Tämä toimii julkisilla palvelimilla, joissa pelaaminen tuntemattomien ihmisten kanssa ei ole ongelma. Monet ystävyyssuhteet pidetään koossa viikottaisilla kortti- tai pallopeleillä, mutta tietokone- sekä konsolipelien roolia ystäväpiireissä ei voida nykyään väheksyä (Schell 2008, 355). Useat pelaajat haluavat pelata ystäviensä kanssa kotipalvelimillaan. Isojen pelaajamäärien kerääminen ystäväpiirin keskuudessa voi kuitenkin olla ongelma. Useat uudet moninpelit ovat ilmeisesti jo huomanneet tämän kohderyhmän:

- Left 4 Dead (2008) on luotu 1 - 8 pelaajalle (Left 4 Dead: Overview 2009).
- Killing Floor (2009) on luotu 1 - 6 pelaajalle (Killing Floor: Overview 2010).
- Alien Swarm (2010) on luotu 1 - 4 pelaajalle (Alien Swarm on Steam 2011).

Pelimuodossa päätettiin pyrkiä pieneen pelaajamäärään, mutta tarjota jokaiselle pelaajalle mahdollisuus valita omanlaisensa pelityyli.

3.3.4 Simulaatio

A2: CO on tunnettu realistisuudestaan verrattuna muihin ensimmäisen persoonan ammuntapeleihin, ja Bohemia Interactive Studio markkinoi peliä nimenomaan taistelusimulaationa (ARMA 2 & ARMA 2: operation Arrowhead 2010). Tämä tarkoittaa, että pelin pelaajakunta koostuu suurimmaksi osaksi henkilöistä, joita kiinnostaa taistelun realistisuus. Schell (2008, 106) mainitsee, että pelin suunnittelijan tulee aina pitää mielessä pelaajien mielenkiinnon kohteet, jotta pelaajille voidaan tarjota, mitä he haluavat. Pelimuodossa pyrittiin säilyttämään

kaikki alkuperäisen pelin realistiset elementit, mutta lisäämään niiden syvyyttä pelaaja- sekä hahmokohtaisella kehityksellä.

3.3.5 Teema

> ”Teema yhdistää pelin elementit kokonaisuudeksi ja ilmaisee, mistä peli kertoo ja mikä on sen tarkoitus” (Schell 2008, 49).

Pelin teeman valintaan vaikuttivat useat asiat, kuten pelimoottorin tekniset ominaisuudet, alkuperäisen pelin sisältö, pelaajakunta, pelaajien määrä sekä pelityyli. Pelimuodon oli siis toimittava suuressa peliympäristössä, hyödynnettävä alkuperäisen pelin suurta varustevalikoimaa, vedottava realistista taistelutoimintaa hakeville pelaajille sekä oltava pelikelpoinen myös pienellä pelaajamäärällä.

Pelimuodon teemaksi valittiin palkkasotilaana toimiminen. Tämä teema toteutti kaikkia edellä mainittuja vaatimuksia:

Pelimoottori sisältää suuria peliympäristöjä, joihin pelaajille voidaan luoda vaihtelevia toimeksiantoja. Myös palkkasotilaat suorittavat sekalaisia tehtäviä toiminta-alueellaan riippuen heidän työnantajastaan.

Peli sisältää useita osapuolia sekä yli sata varustetta. Pelimuotoon voitiin luoda toimintaan sekä varusteisiin kiinnitetty talous, joka on palkkasotilaiden työssä hyvin tärkeää.

A2: CO:n pelaajakunta suosii realistista taistelutoimintaa (Flick 2010). Palkkasotilaana toimiminen edellyttää sotilasoperaatioihin osallistumista.

Pelaajamäärä haluttiin pitää pienenä ja pelityyli ryhmähenkisenä, jotta pelimuoto olisi pienten ystäväpiirien pelattavissa. Palkkasotilaat toimivat usein pienissä ryhmissä kohti yhteistä päämäärää.

Alkuperäisen pelin tavoin päätettiin pelimuodon kielenä säilyttää englanti, jotta se olisi saatavilla koko alkuperäisen pelin pelaajakunnalle. Pelimuodolle annettiin nimeksi **Assignment**, eli toimeksianto, joka kuvaa palkkasotilaan päivittäistä toimintaa ytimekkäästi.

4 PELIMUODON SUUNNITTELU

Suunnitteluosassa hahmoteltiin pelimuodon toiminnot sekä ulkonäkö. Pelimuodosta pyrittiin suunnittelemaan kokonaisuus, jonka ominaisuudet tukevat toisiaan niin toiminnallisesti kuin graafisesti. Pelattavuus sekä perustoiminnot pyrittiin pitämään yhtenäisenä alkuperäisen pelin kanssa, jotta pelimuodon oppimiskynnys olisi mahdollisimman matala.

4.1 Teknologia

> ”Teknologia on pohjimmiltaan se ilmaisumuoto, jossa estetiikka esitetään, mekaniikka tapahtuu, ja jonka kautta tarina kerrotaan” (Schell 2008, 42).

Tässä osassa käsitellään pelimuodon alustaksi valittua teknologiaa, sen ominaisuuksia sekä rajoitteita ja työssä käytettyjä työkaluja. Osa etenee alhaisen tason teknologiasta, kuten pelimoottorin perustoiminnasta, ylhäisen tason teknologiaan, kuten yksiköihin ja graafiseen käyttöliittymään.

4.1.1 Alustustiedosto

Käynnistettäessä **RV3** ajaa konekieleksi käännetyn alustustiedoston nimeltään config.bin, jossa määritellään kyseessä olevan pelin, tässä tapauksessa A2: CO:n, sisältö luokkarakenteena. Jokainen luokka voi sisältää loputtoman määrän ominaisuuksia sekä alaluokkia. Pääluokkiin kuuluvat mm.

- **cfgWorlds**, joka sisältää peliympäristöt
- **cfgVehicles**, joka sisältää objektit
- **cfgWeapons**, joka sisältää aseet ja esineet
- **cfgMagazines**, joka sisältää lippaat sekä
- **cfgAmmo**, joka sisältää ammukset.

Pelin alustustiedoston juuressa on myös määritelty graafisen käyttöliittymän perusluokat. Alustustiedoston luokkiin sekä niiden ominaisuuksiin voidaan viitata

suoritusvaiheessa, mutta sen sisältöä ei voida muokata. Käynnistyksen jälkeen uusien peliympäristöjen, objektien sekä varusteiden määrittely on siis mahdotonta. Tämän vuoksi pelimuodossa on käytettävä pelin omaa sisältöä, mutta sen luomiseen ja käyttäytymiseen peliympäristössä voidaan vaikuttaa.

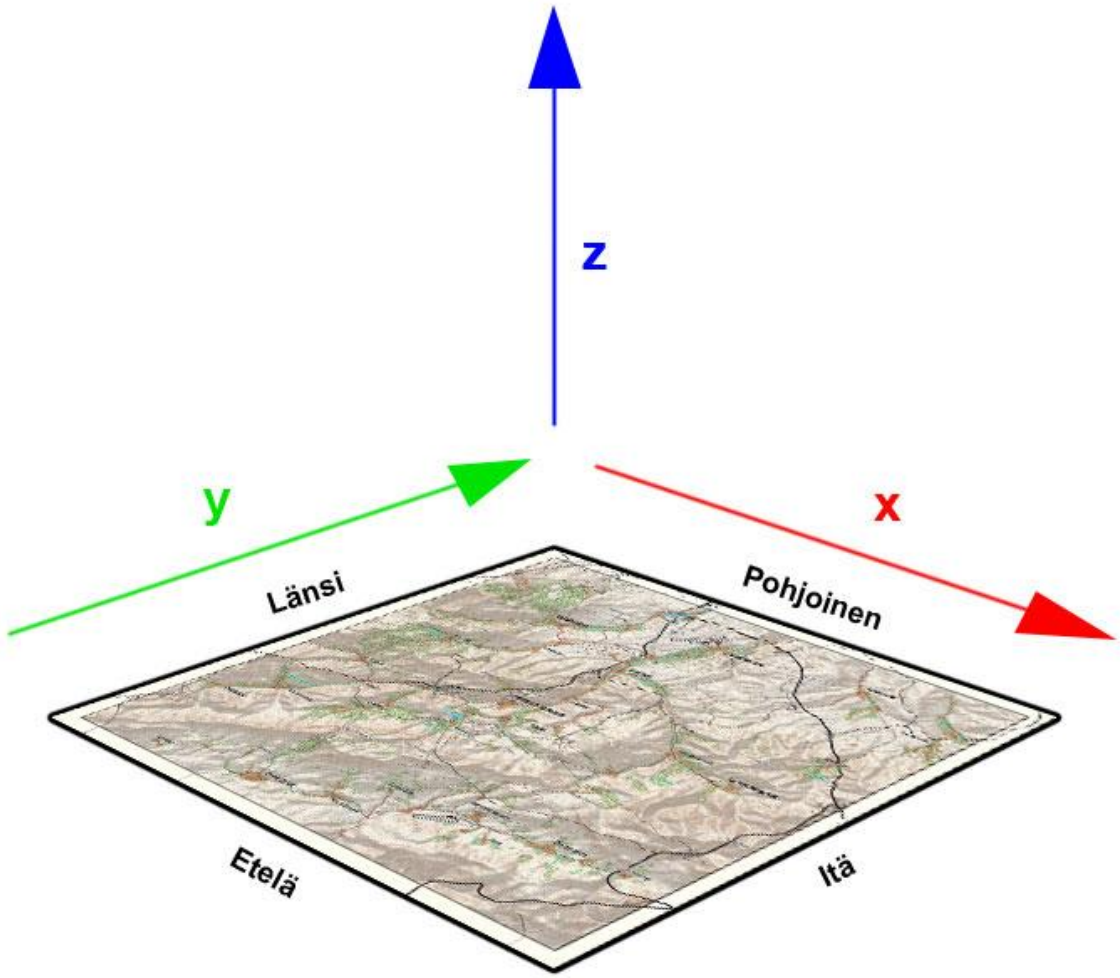
4.1.2 Komentosarjakieli

SQF on BIS:n kehittämä **RV3**-pelimoottorissa käytettävä komentosarjakieli. Komentosarjoja voidaan ajaa suoritusvaiheen aikana, ja pelimoottori tulkkaa komennot yksi kerrallaan. SQF tukee mm.

- neljää päämuuttujatyyppiä (boolean, number, string ja array)
- aritmeettisia operaatioita (+, -, *, /, % ja ^)
- loogisia operaatioita (!, && ja ||)
- vertailuoperaatioita (==, !=, <, >, <= ja >=)
- ehdollisia suorituksia (if - else sekä switch - case)
- perinteisiä toistoja (for, while - do sekä foreach)
- parametreja hyväksyviä sekä arvoja palauttavia funktioita
- yhden sekä monen rivin kommentointia
- monen komentosarjan yhtäaikaista ajamista sekä
- yli tuhatta peliympäristöön, objekteihin sekä graafiseen käyttöliittymään vaikuttavaa komentoa. (SQF syntax 2011.)

4.1.3 Peliympäristöt

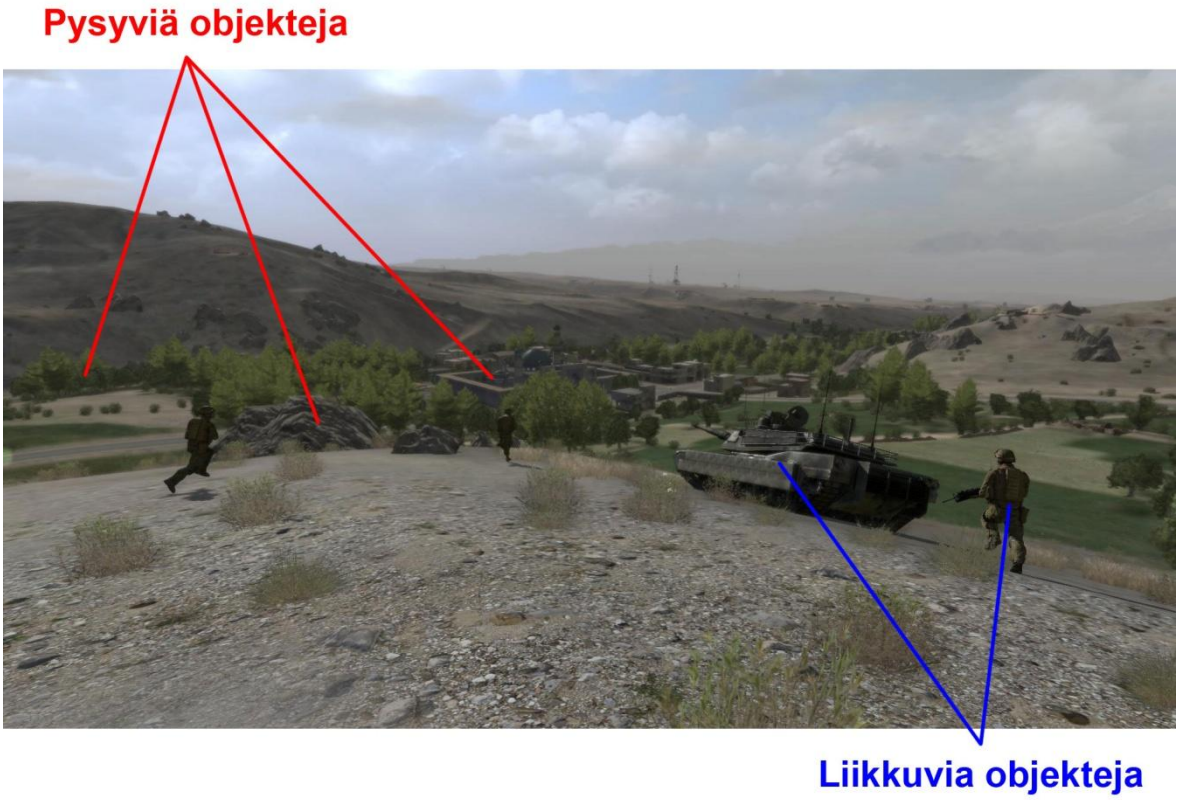
Peliympäristöt ovat korkeuskartoista mallinnettuja ulkoilmaympäristöjä, jotka toteuttavat kolmiulotteista koordinaatistoa. Koordinaatistossa x-akseli on länsi–itä suunnassa, y-akseli etelä–pohjois suunnassa ja z-akseli ala–ylä suunnassa (Kuvio 2). Koordinaatiston yksikkö vastaa pelimaastossa yhtä metriä. A2: CO sisältää valmiiksi mallinnetun vuoristomaastoisen ympäristön nimeltään Takistan, joka on kooltaan 164 km². Takistan sisältää 450 000 esiasetettua pysyvää objektia, kuten kiviä, puita sekä rakennuksia ja aitoja, jotka muodostavat peliympäristön 28 asutuskeskusta. (Takistan 2011.)



Kuvio 2. Peliympäristön koordinaatisto.

4.1.4 Objektit

Peliympäristön objektit ovat ilmentymiä **cfgVehicles**-luokassa määritellyistä alaluokista. Objekteja ovat kaikki pelaajien näkemät kolmiulotteiset elementit, jotka erottuvat ympäristön maastosta. Jokaisella objektilla on perusominaisuuksia, kuten polku objektia kuvaavaan kolmiulotteiseen malliin sekä objektin fyysiset ominaisuudet, kuten paino sekä koostumus. Objektit jakautuvat kahteen pääryhmään: pysyvät objektit, joita ovat (kivet, puut ja rakennukset) sekä liikkuvat objektit (yksiköt, ajoneuvot ja ammukset) (Kuvio 3). Liikkuviin objekteihin sovelletaan fysiikkamallinnusta, kun taas pysyviin ei.



Kuvio 3. Pysyviä ja liikkuvia objekteja peliympäristössä.

4.1.5 Tehtävät

Objekteja asetellaan peliympäristöön pelin sisältämällä tehtäväeditorilla. Editorissa on kaksiulotteinen päänäkymä, jossa valittu ympäristö esitetään x- ja y-akseleilla. Päänäkymä havainnollistaa ympäristön korkeuskäyrät sekä esiasetetut ja käyttäjän asettamat objektit. Tehtävä voidaan tallentaa joko avonaiseen projektimuotoon tai pakattuun testimuotoon. Projektimuodossa tehtävällä on oma hakemistonsa, joka sisältää ensisijaisesti vain tehtävätiedoston nimeltään **mission.sqm**. Tehtävätiedostossa on editorissa asetettujen objektien sekä tehtävän alkuparametrit, kuten tehtävän nimi ja vuodenaika.

Projektihakemistoon voidaan lisätä myös kaksi muuta tärkeää tiedostoa: **description.ext** sekä **init.sqf**. **Description.ext** on tehtävän oma, kääntämätön, alustustiedosto ja **init.sqf** tehtävän komentotiedosto. Tehtävän käynnistyksessä **RV3** kääntää ja ajaa **mission.sqm** sekä **description.ext**-tiedostot, ja aloittaa **init.sqf**-tiedoston tulkkaamisen. Projektihakemistoon voidaan luoda myös loputon

määrä alihakemistoja sekä muita tiedostoja ja niihin viitata alustus- sekä komentotiedostoissa. Koska tehtävän alustustiedostoon voidaan luoda uusia luokkia, ja komennoilla voidaan suoritusvaiheessa vaikuttaa objektien ominaisuuksiin, peliympäristön tilaan sekä graafiseen käyttöliittymään, on erittäin monipuolisten tehtävien luominen mahdollista. Työn aiheena oleva pelimuoto onkin siis käytännössä monimuotoinen tehtävä. Editorin esikatselu-ominaisuus mahdollistaa tehtävän ajamisen missä tahansa muokkauksen vaiheessa.

4.1.6 Yksiköt

Yksiköt ovat ympäristössä liikkuvia objekteja, jotka mallintavat ihmishahmoja. A2: CO sisältää n. 300 erilaista yksikköluokkaa, jotka on jaettu neljäksi osapuoleksi: länsi (West), itä (East), vastarintajoukot (Resistance) sekä siviilit (Civilian).

Jokainen osapuoli on jaettu ryhmittymiin, joilla on erilaiset univormut tai vaatetus (Kuvio 4). Ryhmittymillä on myös erilaisia kulttuuriasetuksia, kuten kasvojen piirteet sekä kieli. länsi-osapuolen saksalainen sotilas puhuu siis saksaa, kun taas itä-osapuolen venäläinen sotilas venäjää. Saman osapuolen yksiköitä voidaan sitoa ryhmiksi, jotka toimivat yhtenäisesti kohti annettua päämäärää. Yksikölle annetaan sotilasarvo, joka vaikuttaa sen sijoitukseen ryhmässä. Ryhmän korkea-arvoisin henkilö toimii sen johtajana. Ryhmän johtajan menehdyttyä ottaa seuraavaksi korkea-arvoisin ryhmän jäsen johdon ja jatkaa tehtävää. Yksiköt voivat myös nousta ajoneuvoihin ja ohjata niitä.

Kuvio 4. Erilaisia yksiköitä A2: CO-pelissä.

Tehtävää luodessa valitut yksiköt määritellään pelaajien ohjattavaksi. Tehtävän käynnistyttyä pelaajat ohjaavat yksikköään peliympäristössä ensimmäisen persoonan näkymästä. Pelaaja voidaan asettaa ohjaamaan minkä tahansa osapuolen yksikköä. Muita yksiköitä ohjaa pelimoottorin tekoäly. Osapuolilla on esimäärätyt suhteet, jotka vaikuttavat tekoälyn ohjaaman yksikön käyttäytymiseen sen kohdatessa toisia osapuolia. Esimerkiksi länsi- sekä itä-osapuolet ovat toisilleen vihamielisiä, kun taas siviilit ovat kaikille hyväntahtoisia.

4.1.7 Varusteet

Varusteisiin kuuluvat aseet, lippaat sekä esineet. Aseet on määritelty **cfgWeapons**-luokassa. A2: CO sisältää n. 150 asetta, joista jokaisella on eri ominaisuuksia, kuten ulkonäkö, yhteensopivat lippaat, mahdolliset tähtäinlaitteet sekä toissijaiset ampumalaitteet. Aseilla yksiköt voivat vahingoittaa peliympäristössä olevia toisia yksiköitä sekä jopa itseään. Aseet käyttävät lippaita, jotka sisältävät ammuksia. Lippaat ja ammukset on määritelty **cfgMagazines**- ja **cfgAmmo**-luokissa. **cfgWeapons**-luokka sisältää myös esineitä, joita ovat esimerkiksi kiikarit, kello, kartta ja radio. Jokaisella esineellä on oma

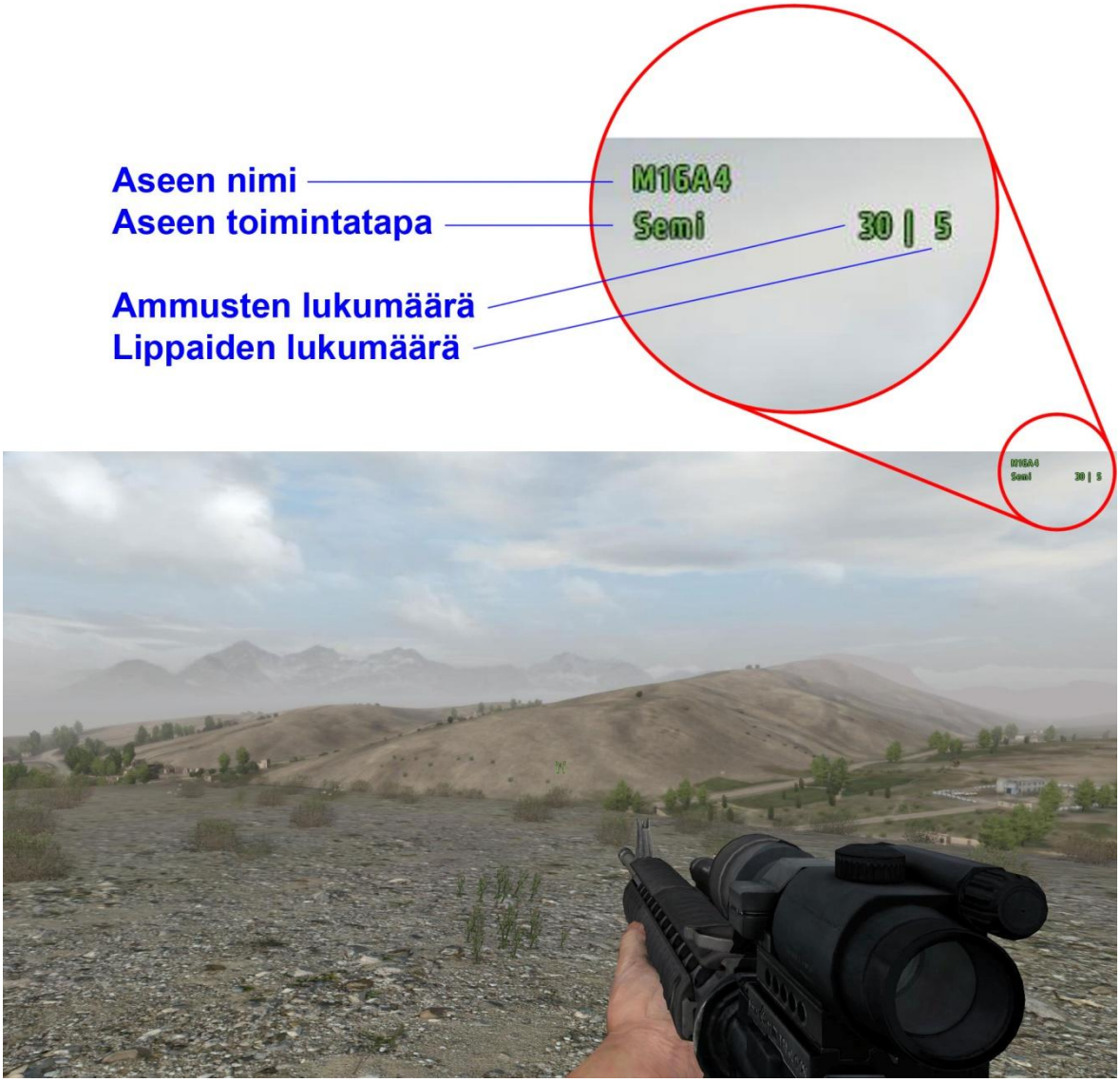
pelimekaaninen käyttötarkoituksensa. Jokainen yksikkö voi kantaa saman määrän varusteita. Yksiköt voivat halutessaan pudottaa varusteita maahan tai asettaa niitä laatikoihin toisten yksiköiden poimittavaksi. Yksiköt voivat myös poimia menehtyneiden yksiköiden varusteita. Pelaajat näkevät yksikkönsä varusteet varustenäkymästä, jonka he voivat avata missä pelin vaiheessa tahansa (Kuvio 5).

Kuvio 5. A2: CO:n varustenäkymä.

4.1.8 Graafinen käyttöliittymä

Pelimoottori tukee kolmiulotteisen ympäristön eteen piirrettäviä kaksiulotteisia näkymiä, jotka määritellään luokkina tehtävän **description.ext**-alustustiedoston juuressa. Näkymien elementit piirretään näytölle suhteessa käyttäjän valitsemaan resoluutioon sekä näkymäkokoon. Näkymälle varattu tila voi siis olla minkä tahansa kokoinen pikselialue näytöltä. Elementit jakautuvat tausta- sekä aktiivielementteihin, joita voidaan muokata SQF-komennoilla. Taustaelementtejä ovat mm. taustavärit, tekstit sekä kuvat. Kuvia voidaan asettaa tehtävän projektihakemistoon ja niihin viitata alustus- sekä komentotiedostossa. Aktiivielementtejä ovat mm. painikkeet, tekstikentät sekä listat. Aktiivielementteihin voidaan myös lisätä tapahtuman käsittelijöitä (event handlers), jotka suorittavat komentosarjan esimäärätystä tapahtumasta. Näkymät voivat olla pysyviä, kuten

pelaajan käytössä olevan aseen tiedot –näkymä (Kuvio 6) tai väliaikaisia, kuten pelaajan varustenäkymä (Kuvio 5). Pysyvät näkymät eivät vaikuta pelaajan hallitseman yksikön ohjaamiseen, kun taas väliaikaiset näkymät tuovat ruudulle kursorin ja siirtävät hiiri- sekä näppäimistökomennot yksikköä ohjaamasta käsiteltäväksi avatussa näkymässä. Pelaajan hallinta yksikköönsä palaa, kun väliaikainen näkymä suljetaan.



Kuvio 6. Pelaajan käytössä olevan aseen tiedot -näkymä.

4.1.9 Työkalut

Työssä käytettiin kolmea työkalua: Microsoft Office Exceliä, Notepad++ sekä jo ylempänä mainittua A2: CO:n sisältämää tehtäväeditoria. Microsoft Office Excel -taulukkolaskentaohjelmaa käytettiin sen ruutupohjaisuuden ja graafisuuden vuoksi näkymien ulkoasun suunnitteluun. Notepad++-ohjelmalla luotiin tehtävän alustus- sekä komentotiedostot ja tehtäväeditorilla esikatseltiin tehtävää sekä pakattiin se testimuotoon.

Notepad++ on GNU-hankkeen yleisellä lisenssillä julkaistu tekstinkäsittelyohjelma, jonka painopisteenä on lähdekoodin muokkaus (Notepad++ Home 2011). Se tukee mm.

- syntaksin korostusta
- useiden tiedostojen yhtäaikaista katselua välilehtinä
- merkkijonojen hakua ja korvausta sekä
- koodin osien piilottamista.

4.2 Pelimekaniikka

> ”Pelimekaniikka on pelin todellinen ydin. Se kattaa ne vuorovaikutukset sekä suhteet, jotka jäävät jäljelle, kun kaikki estetiikka, teknologia sekä tarina karsitaan pois.” (Schell 2008, 130.)

Tässä osassa käydään läpi pelimuodon pelimekaniikan kannalta tärkeät toiminnot sekä niiden vuorovaikutukset ja rajoitukset, jotka toimivat pelaajaa peliympäristössä ohjaavina sääntöinä.

Pelimuodossa päivän pituus lasketaan pelaajien toimintojen mukaan. Yhden päivän kulku on seuraavanlainen:

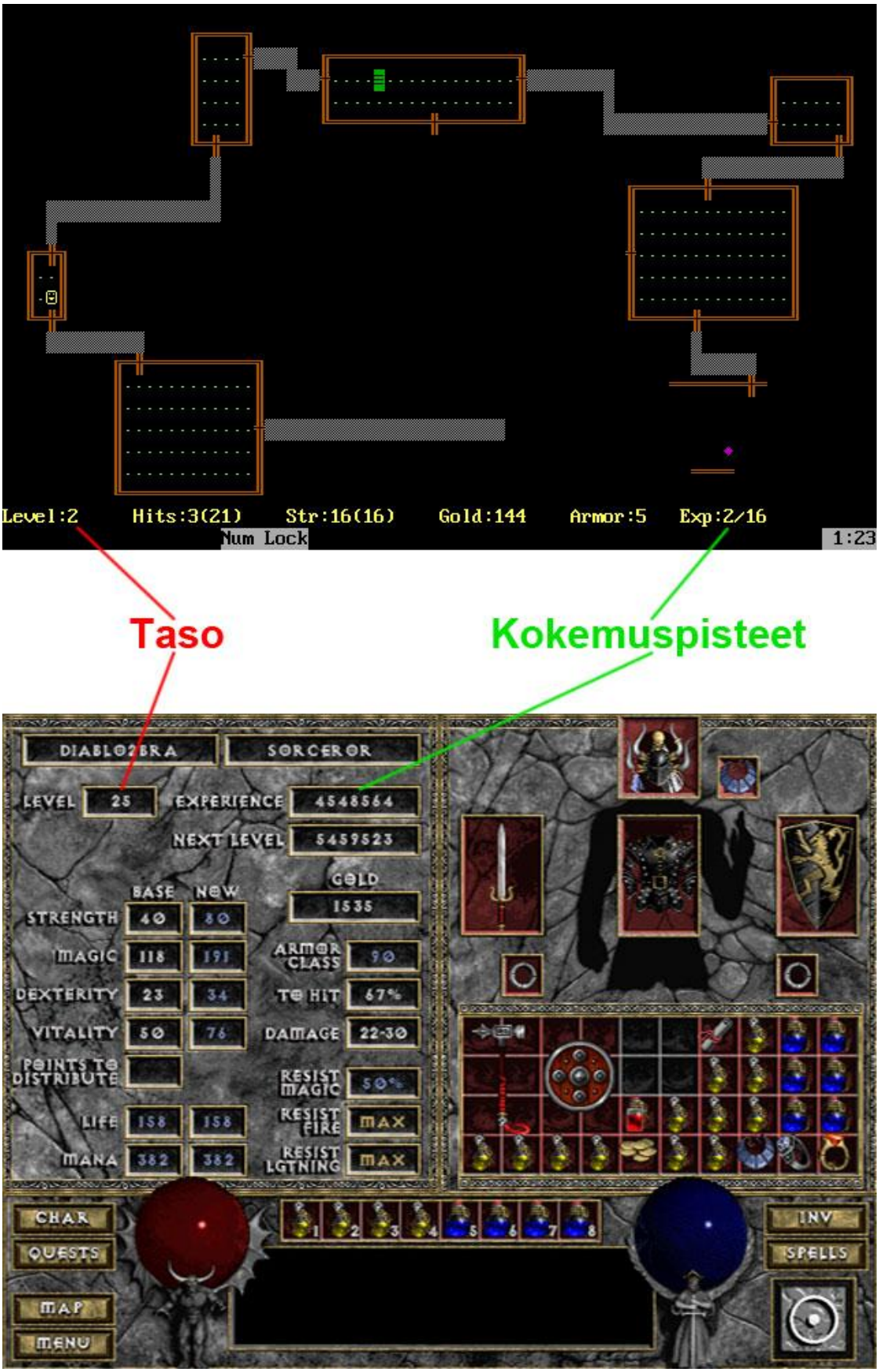
1. Ensimmäinen päivä alkaa. Pelaajat aloittavat tukikohdasta.
2. Pelaajat valitsevat itselleen hahmon.
3. Pelaajat ostavat rahalla varusteita ja siirtävät ne hahmonsa käyttöön.
4. Pelaajat äänestävät seuraavaksi pelattavaa toimeksiantoa.
5. Eniten ääniä saanut toimeksianto aloitetaan.

6. Pelaajat keräävät toimeksiannossa hahmolleen kokemusta esim. tuhoamalla vihollisia sekä suorittamalla työtehtäviä. Rahaa pelaajat saavat jokaisesta suoritetusta työtehtävästä.
7. Kun toimeksiannon jokainen työtehtävä on suoritettu, pelaajien äänestäessä lopettavansa toimeksiannon tai kun kaikkien pelaajien hahmo on menehtynyt, herätetään menehtyneet hahmot henkiin ja kaikki hahmot palautetaan tukikohtaan.
8. Uusi päivä alkaa. Tukikohdassa pelaajat aktivoivat hahmolleen taitoja hänelle kertyneillä taitopisteillä.
9. Palataan kohtaan 2. tai 3.

4.2.1 Kokemus ja taidot

> ”Peleissä resursseja ovat ne asiat, joita pelaajat käyttävät hyväkseen saavuttaakseen päämääriä” (Rabin ym. 2005, 116).

Kokemuspisteet (experience points) sekä tasot (levels) ovat pelialalla kauan käytettyjä käsitteitä (Kuvio 7). Kokemuspisteitä saa usein tuhoamalla vihollisia tai suorittamalla pelin tehtäviä. Saavuttaakseen seuraavan tason on pelaajan kerättävä ennalta määrätty kokemuspistemäärä. Tason saavuttamisesta palkitaan esimerkiksi pelaajahahmon käyttöön tulevalla uudella taidolla. Taidot ovat erinomainen tapa antaa pelaajalle vallan tuntu (Schell 2008, 151).



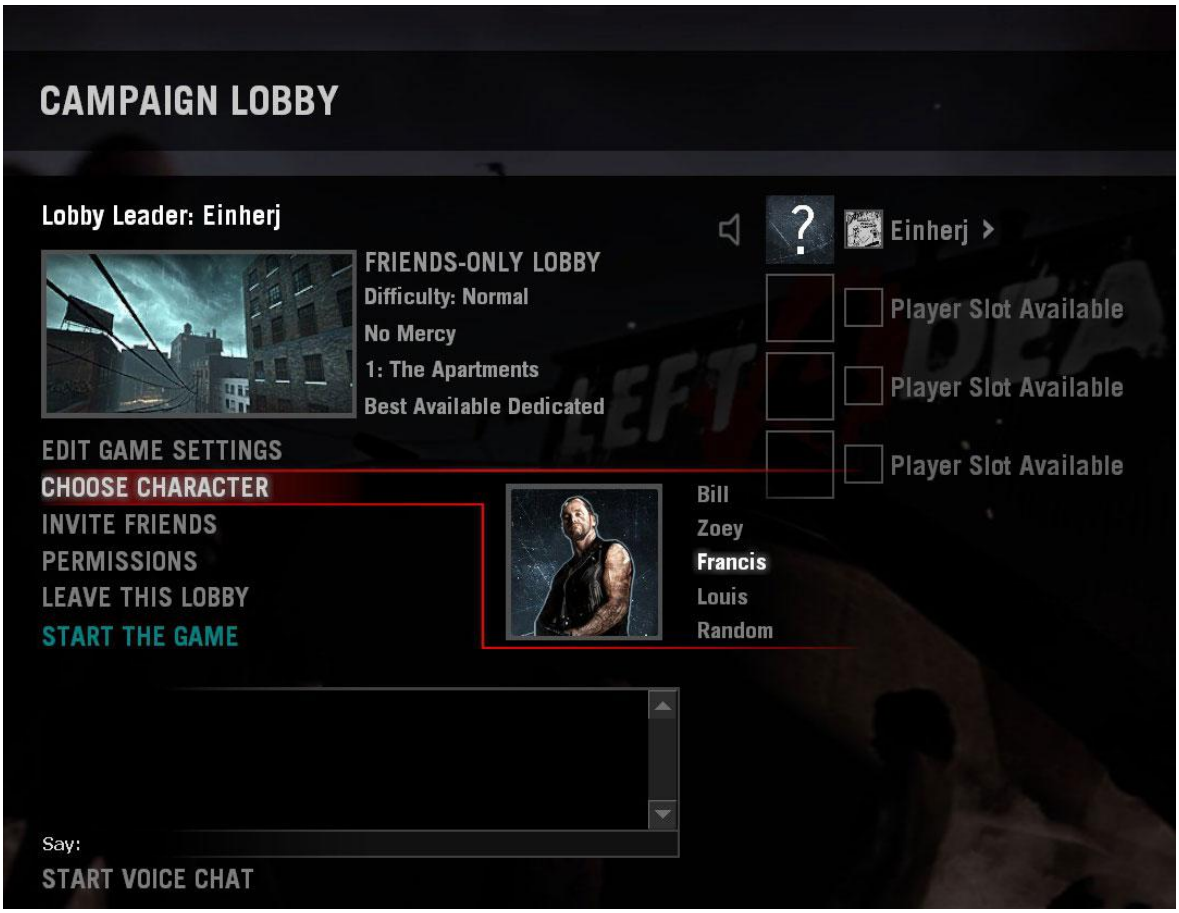
Kuvio 7. Tasot ja kokemuspisteet peleistä Rogue (1983) ja Diablo (1996).

A2: CO ei sisällä minkäänlaista kokemus- tai taitojärjestelmää, joten sellainen oli luotava. Pelimuotoon luotiin seuraavanlainen järjestelmä: Jokainen hahmo aloittaa tasolta yksi. Alussa vain yksi taito on aktiivisena. Hahmot saavat kokemusta tekemällä vahinkoa viholliysyksiköihin sekä suorittamalla toimeksiantojen työtehtäviä. Pelimuodossa päätettiin käyttää niin sanottua taitopuu-järjestelmää, jossa taitoja aktivoidaan järjestyksessä eräänlaisen puukuvion mukaisesti (Kuvio 8).Tiettyjen taitojen aktivointi vaatii yhden tai useamman taidon olevan aktiivisena. Saavutettuaan uuden tason hahmo saa yhden taitopisteen. Taitopisteellä pelaaja voi aktivoida uuden taidon hahmonsa taitopuusta. Kun taito on aktivoitu, taitopiste katoaa ja taito pysyy aktiivisena peli-istunnon loppuun asti. Taidon aktivointi antaa hahmolle uuden kyvyn tai koko pelaajaryhmälle mahdollisuuden ostaa uusia varusteita. Kyky voi olla esimerkiksi toisten pelaajahahmojen parantaminen tai mahdollisuus kutsua pelialueelle tykistötulta.

Kuvio 8. Esimerkki taitopuusta.

4.2.2 Hahmot

Useat pelit antavat pelaajalle vaihtoehtoja pelattavan hahmon valinnassa. Joskus valinta vaikuttaa vain hahmon ulkonäköön (Kuvio 9), mutta useimmin myös hahmon ominaisuuksiin ja pelin sisältöön (Kuvio 10 & Kuvio 11).



Kuvio 9. Hahmon valinta Left 4 Dead (2008) -pelissä.

Kuvio 10. Hahmon valinta Killing Floor (2009) -pelissä.

Kuvio 11. Hahmon valinta Alien Swarm (2010) –pelissä.

Dille ja Platten (2007, 65) esittävät, että ihanteellisesti pelaajilla tulee olla mahdollisuus valita, millä hahmolla he haluavat pelata, ja hahmovalinta tulee tehdä osaksi pelikokemusta. Pelaajille haluttiin antaa vaihtoehtoja hahmon valinnassa, ja jokaisen hahmon tuli olla selkeästi erottuva yksilö. Useissa taistelupeleissä hahmot eritellään siten, että jokaisella hahmolla on käytössään erilaisia aseita (Kuvio 12) tai hahmot käyttäytyvät peliympäristössä eri tavoin (Kuvio 13). Usein pelaajat voivat myös valita hahmokseen peliympäristössä jo olevia hahmoja.

Kuvio 12. Hahmon valinta Day of Defeat: Source (2005) -pelissä.

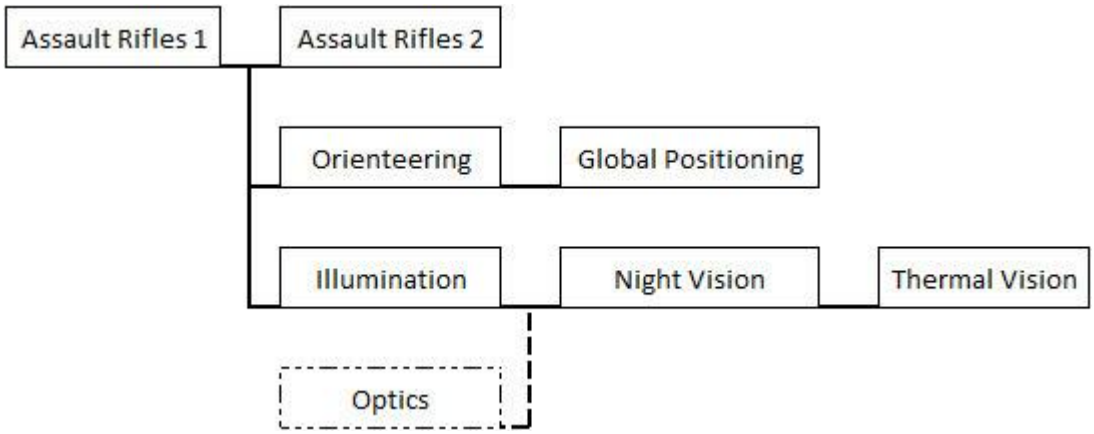
Kuvio 13. Hahmon valinta Team Fortress 2 (2007) -pelissä.

Pelaajan halua ilmaisuun ei saa rajoittaa pelin suunnittelijan visiolla, kuinka peliä kuuluisi pelata (Rabin ym. 2005, 116). A2: CO:ssa jokainen yksikkö omaa ulkonäköä lukuun ottamatta samat ominaisuudet sekä pystyy käyttämään mitä tahansa peliympäristöstä löytyvää varustetta. Peli sisältää myös toimintoja, joilla pelaajat voivat vaihdella aseita keskenään. Pelaajat ovat tottuneet näihin toimintoihin, joten heidän vapauttaan ei haluttu vähentää karsimalla niitä pois. Hahmojen yksilöintiin oli siis keksittävä omaperäinen ratkaisu: pelaajat voivat vaihtaa hahmoansa tukikohdassa, mutta jokaista hahmoa voi pelissä olla vain yksi. Jokainen hahmo kartuttaa kokemusta eri tavoin, ja kokemuksellaan sallii uusien varusteiden käytön koko ryhmälle. Jokainen hahmo voi käyttää samoja varusteita, mutta hahmot saavat enemmän kokemuspisteitä käyttäessään hahmonsa rooliin sopivia varusteita sekä toimiessaan roolinsa edellyttämällä tavalla. Näin jokainen pelaaja voi valita omaan pelityyliinsä sopivan hahmon, mutta ei sitoudu käyttäytymään hahmonsa roolin tavoin, vaikka se onkin pelimuodossa edistymisen kannalta hyödyllisempää. Suunnittelussa päädyttiin viiteen toisistaan pelimekaanisesti eroavaan hahmoon, jotka ovat

- ryhmänjohtaja (Team leader)
- konekiväärimies (Machinegunner)
- tukimies (Support)
- tarkka-ampuja sekä (Sniper)
- räjähde-asiantuntija (Demolitions).

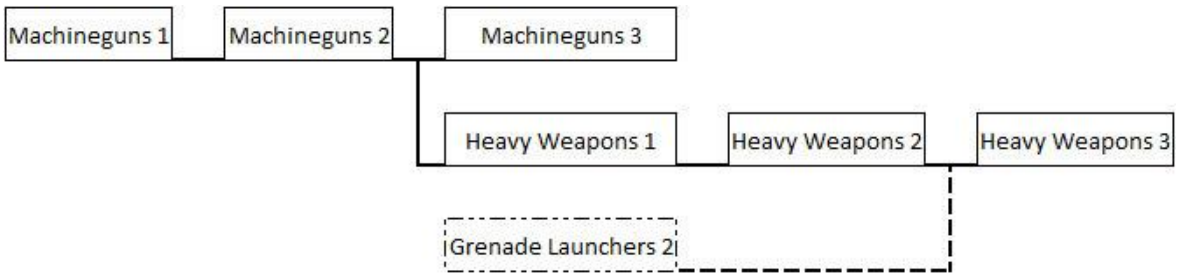
Ryhmänjohtaja. Ryhmänjohtajan roolina on antaa ryhmälle käskyjä tehtävän suorittamisesta. A2: CO sisältää toiminnon, jolla ryhmän korkea-arvoisin henkilö pystyy luomaan ryhmänsä jäsenille käskyjä, kuten käskyn liikkua tiettyyn karttapisteeseen tai käskyn avata tuli tiettyä vihollista kohti. Annetut käskyt näkyvät alaisten ruudulla. Koska pelaajat ovat vapaita liikkumaan peliympäristössä haluamallaan tavalla, käskyjen noudattaminen ei ole pakollista. Pelimuodon tarkoituksena on kuitenkin korostaa joukkuepeliä, joten käskyjen noudattamisesta päätettiin palkita sekä ryhmänjohtajaa, että käskyä noudattavaa ryhmän jäsentä kokemuspisteillä. Näin ryhmänjohtajan on hyödyllistä antaa käskyjä, jotka vaikuttavat alaisille suorituskelpoisilta ja alaisten on hyödyllistä noudattaa niitä. Ryhmänjohtaja voi aktivoida taitopuustaan ryhmänsä käyttöön eritasoisia

rynnäkkökiväärejä, aseiden lisälaitteita sekä muita varusteita, kuten kellon, kartan, kompassin, GPS:n ja pimeänäkölaitteen (Kuvio 14).



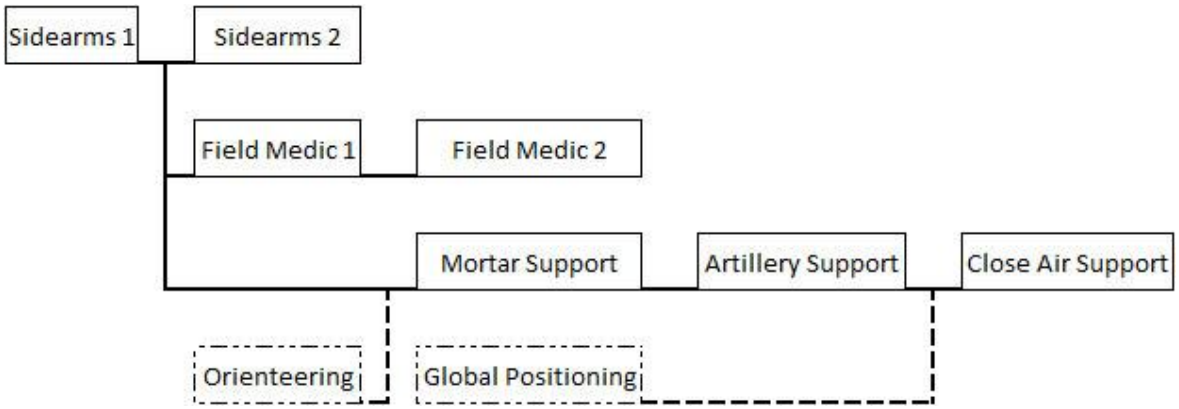
Kuvio 14. Ryhmänjohtajan taitopuu.

Konekiväärimies. Konekiväärimiehen roolina on pakottaa vihollinen suojautumaan ampumalla, sekä vetää huomiota itseensä. A2: CO:ssa tekoälyn ohjaamien yksikköjen reaktio ympärille osuviin ammuksiin on heittäytyä maahan makaamaan. Jos tekoäly tulkitsee tilanteen olevan erittäin vakava, se asettaa yksikön pakenemistilaan, jossa yksikkö ryömii tai juoksee uhkaa vastakkaiseen suuntaan. Konekiväärimiehelle päätettiin antaa kokemuspisteitä jokaisesta hänen tulen allaan maahan heittäytyneestä sekä pakenevasta vihollisesta. Näin Konekiväärimiehen hahmoa ohjaavan pelaajan on hyödyllistä ampua vihollisia kohti vetäen huomiota itseensä ja tukien ryhmäänsä. Konekiväärimies voi aktivoida taitopuustaan uusia konekiväärejä ryhmänsä käyttöön (Kuvio 15) sekä saa lisäpisteitä käyttäessään konekiväärejä muiden aseiden sijaan.



Kuvio 15. Konekiväärimiehen taitopuu.

Tukimies. Tukimiehen roolina on sijoittaa itsensä toiminnan taka-alalle ja tarpeen tullessa tukea muita ryhmän jäseniä. A2: CO sisältää parannustoiminnon, jolla määrätyt yksiköt voivat parantaa haavoittuneita toisia yksiköitä. Tukimies voi aktivoida itselleen parannuskyvyn ja saada kokemuspisteitä jokaisesta parantamastaan pelaajahahmosta. A2: CO sisältää myös toiminnon, jolla luodaan virtuaalinen tulituki-patteristo ja käsketään se ampumaan pelikentälle iskuja. Tätä toimintoa hyväksikäyttäen päätettiin tukimiehelle luoda myös toinen aktivoitava kyky, jolla hän voi kutsua peliympäristöön heittimistö- sekä tykistöiskuja. Näin Tukimies sopii sekä puolustavaan, että hyökkäävään pelityyliin. Tukimies voi taitopuustaan aktivoida näiden kykyjen lisäksi myös lähipuolustusaseita, kuten pistooleja ja konepistooleja ryhmänsä käyttöön (Kuvio 16).



Kuvio 16. Tukimiehen taitopuu.

Tarkka-ampuja. Tarkka-ampujan roolina on vahingoittaa yksittäisiä vihollisia pitkän matkan päästä. A2: CO sisältää useita tarkkuusammuntaan erikoistuvia aseita, joiden käyttöä tällä hahmolla suositaan. Tarkka-ampujan päätettiin saavan lisäpisteitä käyttämällä tarkkuuskiväärejä. Koska vihollisyksiköihin osuminen vaikeutuu niiden ollessa kaukana tai liikkeessä, päätettiin ampumismatkasta sekä vihollisen nopeudesta antaa myös lisäpisteitä. Tarkka-ampuja voi aktivoida taitopuustaan pelaajaryhmälle aseoptiikat, savuheitteet, äänenvaimentimet aseisiin sekä monenlaisia tarkkuuskivääreitä (Kuvio 17).

Kuvio 17. Tarkka-ampujan taitopuu.

Räjähde-asiantuntija. Räjähde-asiantuntijan roolina on käyttää hyväkseen räjähteiden tuomaa tulivoimaa vahingoittaakseen vihollisyksiköitä sekä -ajoneuvoja. Tässä tapauksessa räjähteiksi lasketaan käsikranaatit, kranaatinheittimet, singot sekä miinat, joita hahmo voi aktivoida taitopuustaan (Kuvio 18). Räjähde-asiantuntijalle päätettiin soveltaa samaa vihollisen suojautumisesta ja pakenemisesta annettavaa pisteytystä kuin konekiväärimiehelle, mutta antaa pisteitä myös ampumismatkasta. Lisäpisteitä päätettiin antaa myös objektien tuhoamisesta sellaisessa toimeksiannossa, jossa kyseisten objektien tuhoaminen on päämääränä.

Kuvio 18. Räjähde-asiantuntijan taitopuu.

4.2.3 Tukikohta, toimeksiannot ja viholliset

Tukikohta. Pelimuodon valmisteluosuudessa, eli tukikohdassa, pelaajat voivat rauhassa tutkia hahmojen, varusteiden sekä toimeksiantojen ominaisuuksia ja tehdä niihin liittyviä päätöksiä. Siellä pelaajat voivat liikkua vapaasti vahingoittumatta. Tukikohta sisältää kolme kohdetta, joiden edessä on keltaisella suorakaiteella merkattu alue. Alueeseen astuttuaan pelaajalle aukeaa siihen sidottu näkymä. kohteet ja niiden näkymät ovat

- parakki, jonka edestä aukeaa ryhmänäkymä
- metallikontti, jonka edestä aukeaa kauppanäkymä sekä
- karttataulu, jonka edestä aukeaa toimeksiantonäkymä.

Kohteet on sijoitettu kolmioon siten, että jokaisesta kohteesta on lyhyt matka toisiin kohteisiin ja ettei kohteiden välissä ole esteitä. Näin pelaajien liikkuminen tukikohdassa on mahdollisimman helppoa.

Toimeksiannot. Schell (2008, 153) esittää, että hyvän pelisuunnittelijan on käytettävä hyväkseen sattuman ja todennäköisyyden elementtejä, jotta pelikokemus olisi aina täynnä haastavia päätöksiä sekä mielenkiintoisia yllätyksiä. pelimuodon toimintaosuuksissa, eli toimeksiannoissa, pelaajat keräävät hahmolleen kokemuspisteitä sekä itselleen rahaa. Jokaisen päivän alussa generoidaan 1–4 uutta toimeksiantoa, jotka jaetaan satunnaisesti pelimuodon kolmelle osapuolelle: sininen, vihreä ja punainen. Pelaajat äänestävät seuraavaksi pelattavan toimeksiannon ja se aloitetaan. Yhden toimeksiannon pelaaminen kuluttaa yhden päivän pelimuodon sisäistä aikaa. Jokaisella toimeksiannolla on osittain satunnaisesti generoituja ominaisuuksia, kuten tapahtumapaikka, vaikeustaso, työtehtävien määrä ja tavoitteet, vihollisyksiköiden määrä ja laatu sekä kuinka kauan toimeksianto on tarjolla. Toimeksianto on tarjolla 1–5 päivää, jonka jälkeen pelaamaton toimeksianto katoaa. Toimeksianto sisältää yhden päätyötehtävän sekä 1–3 sivutyötehtävää. Työtehtäviä on seitsemää mahdollista: tietyn yksikkömäärän tuhoaminen, tietyn ajoneuvomäärän tuhoaminen, kohteen tuhoaminen, yksikön tuhoaminen, yksikön vangitseminen, yksikön vapauttaminen sekä materiaalin kaappaaminen.

Viholliset. Toimeksiantoon luodaan vihollisyksiköitä, jotka ovat vihamielisiä pelaajaryhmää kohtaan. Luotavien vihollisyksiköiden ulkonäkö määräytyy toimeksiannon antaneen osapuolen mukaan: sinisen osapuolen toimeksiannoissa vihollisyksikköinä esiintyy vihreitä sekä punaisia, vihreän osapuolen toimeksiannoissa esiintyy sinisiä sekä punaisia, ja punaisen osapuolen toimeksiannoissa vihollisina ovat siniset ja vihreät yksiköt. Yksiköiden määrä riippuu pelaajien määrästä, pelaajien tasosta sekä toimeksiannon vaikeustasosta. Vihollismäärässä on myös pieni satunnainen tekijä, jolla lisätään pelimuodon jälleenpelattavuusarvoa. Vihollisyksiköt järjestetään 1–10 yksikön ryhmiksi, joille

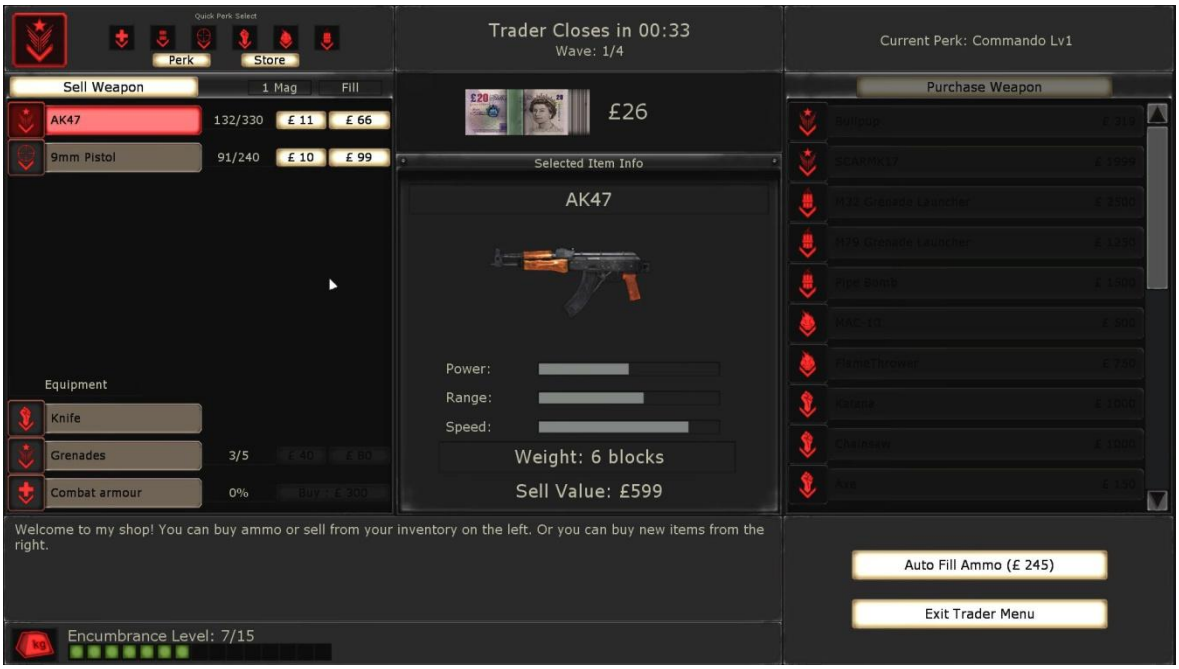
annetaan oma tehtävänsä, kuten kohteen vartiointi tai alueen partiointi. Vihollisyksiköillä on kymmenen eri roolia, jotka määräävät heistä saadun kokemuksen määrän. Vihollisyksiköiden roolit tärkeysjärjestyksessä pienimmästä suurimpaan ovat: kuljettaja, lääkintämies, pioneeri, kiväärimies, kranaatinampuja, panssarintorjuja, konekiväärimies, tarkka-ampuja, ryhmänjohtaja sekä upseeri.

Vihollisyksiköt varustetaan pelaajien varustuksen sekä toimeksiannon vaikeustason mukaan: jos toimeksiannolla on iso vaikeustaso, on vihollisyksiköiden varustus parempi kuin pelaajien. Jos taas vaikeustaso on pieni, on yksiköiden varustus pelaajia huonompi. Myös vihollisyksiköiden tekoälyn taso on suoraan verrannollinen toimeksiannon vaikeustasoon. Työtehtävästä saadun rahallisen palkkion määrä riippuu sen vaikeustasosta sekä toimeksiannon vihollisyksiköiden määrästä.

4.2.4 Raha ja hankkijat

> ”Pelin sisäinen talous voi antaa sille yllättävää syvyyttä ja oman elämänsä” (Schell 2008, 204).

Useat pelit sisältävät jonkinlaisen talouden mallintamisen (Kuvio 19 & Kuvio 20). Peleissä rahaa hankitaan esimerkiksi tuhoamalla vihollisia, löytämällä sitä peliympäristöstä tai suorittamalla tehtäviä. Rahaa käytetään usein uusien varusteiden tai muiden virtuaalisten esineiden hankkimiseen, jotka parantavat pelaajan mahdollisuuksia edistyä pelissä.



Kuvio 19. Kauppanäkymä Killing Floor (2009) -pelissä.

Kuvio 20. Kauppanäkymä Mount & Blade: Warband (2010) -pelissä.

Pelin resursseilla on merkitystä vain, jos niiden tarjonta on sekä hyödyllistä että rajoitettua (Rabin ym. 2005, 117). Pelimuotoa varten luotiin arvo- sekä kauppajärjestelmä. Käynnistyksessä jokaisen varusteen hinta lasketaan sen

ominaisuuksista tai sille määrätään kiinteä hinta. Esimerkiksi tarkoilla aseilla on epätarkkoja suurempi hinta, ja isompi reppu maksaa enemmän kuin pienempi. Pelimuoto sisältää kaupan, josta pelaajat voivat ostaa taidoista aktivoimiaan varusteita. Rahaa pelaajat saavat suorittamalla toimeksiantojen työtehtäviä. Pelaajat voivat myös myydä vanhoja tai vihollisyksiköiltä saatuja varusteita kaupassa. Varusteen myyntihinta on pienempi kuin sen ostohinta. Varusteet on jaettu kolmelle hankkijalle, jotka edustavat pelin osapuolia: sininen, vihreä sekä punainen. Sininen hankkija myy pohjoisamerikkalaisia varusteita, vihreä eurooppalaisia varusteita ja punainen venäläisiä varusteita.

4.3 Estetiikka

> ”Estetiikka määrittää miltä pelisi näyttää, kuulostaa, tuoksuu, maistuu sekä tuntuu” (Schell 2008, 42).

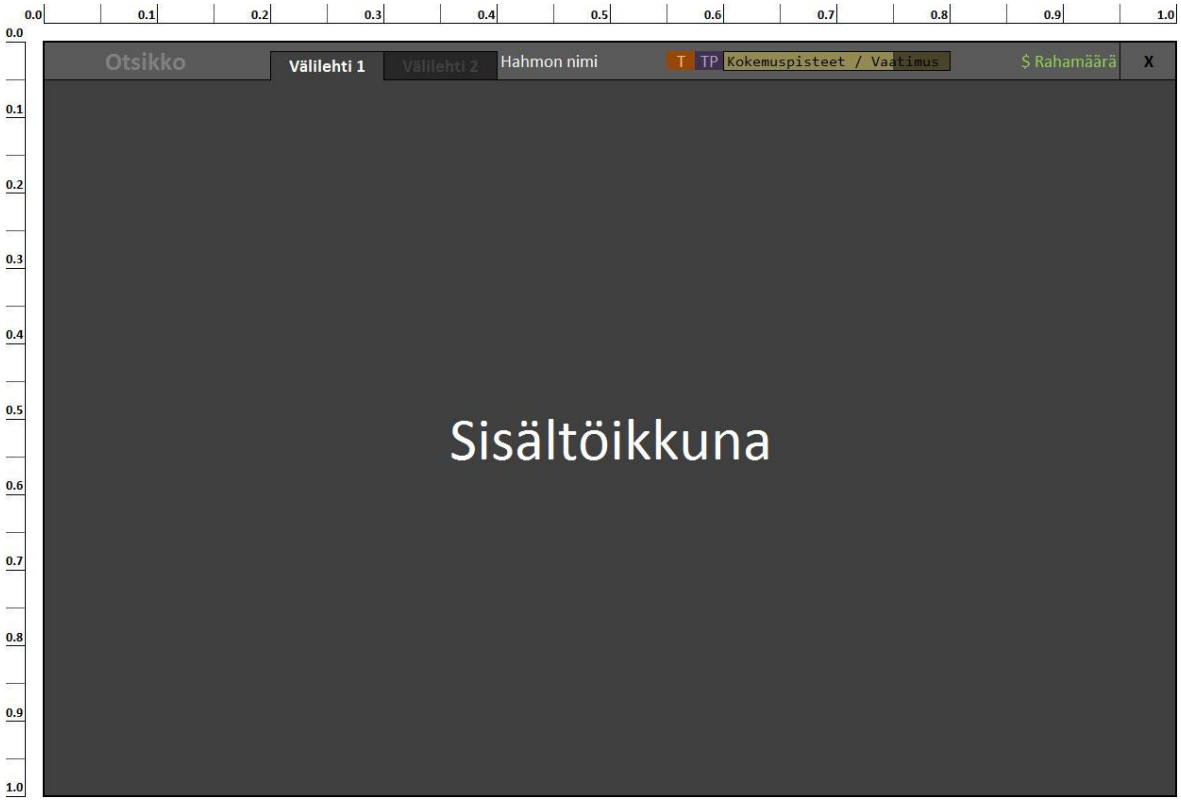
Tässä osassa esitetään pelimuodon graafiset sekä peliympäristöön liittyvät ominaisuudet.

4.3.1 Näkymät

Pelimuotoon luotiin kolme väliaikaista näkymää, joiden kautta pelaajat voivat olla vuorovaikutuksessa pelin toimintojen kanssa. Nämä näkymät ovat: ryhmänäkymä, kauppanäkymä sekä toimeksiantonäkymä. Pelimuotoon luotiin myös yksi pysyvä näkymä, tilannenäkymä, josta pelaajat voivat seurata edistymistään peli-istunnon joka vaiheessa. Spolskyn (2001, 44–45) mukaan hyvä käyttöliittymä on sellainen, johon ihmiset ovat tottuneet, joten olemassa olevien standardien mukaisia elementtejä tulee käyttää mahdollisimman paljon.

Sapluuna. Pelimuoto rakennettiin peliin, joka vaatii käyttäjiltään Microsoft Windows -käyttöjärjestelmän, joten sen väliaikaisissa näkymissä pyrittiin mallintamaan Windowsin standardeja. Näyttötilan säästämiseksi pelimuodon sisäisiä standardeja oli myös luotava, mutta ne pidettiin yhtenevinä. Näkymille suunniteltiin sapluuna, jonka päälle kaikki näkymät luotiin (Kuvio 21). A2: CO:ssa

näkymä-elementtien sijoitukset sekä koot ovat pelaajan käyttämän resoluution suhdelukuja. Näkymätilan molemmat akselit kulkevat koordinaatistossa nollasta yhteen. Sapluunassa näkymätilan akselit jaettiin 20:een pääosaan, jotka voitiin tarvittaessa jakaa neljään pienempään osaan. Tekstirivin korkeudeksi määriteltiin yksi pystysuora pääosa. Näkymätilaan määriteltiin kaksi aluetta: otsikkopalkki sekä sisältöikkuna. Otsikkopalkki on yhden pääosan korkuinen ja sisältää näkymän otsikon, kaksi mahdollista välilehteä, pelaajan valitseman hahmon nimen, hahmon tason, hahmon taitopisteet, hahmon kokemuspistepalkin, pelaajan rahamäärän sekä poistumispainikkeen. Sisältöikkuna sisältää kunkin näkymän omat elementit.



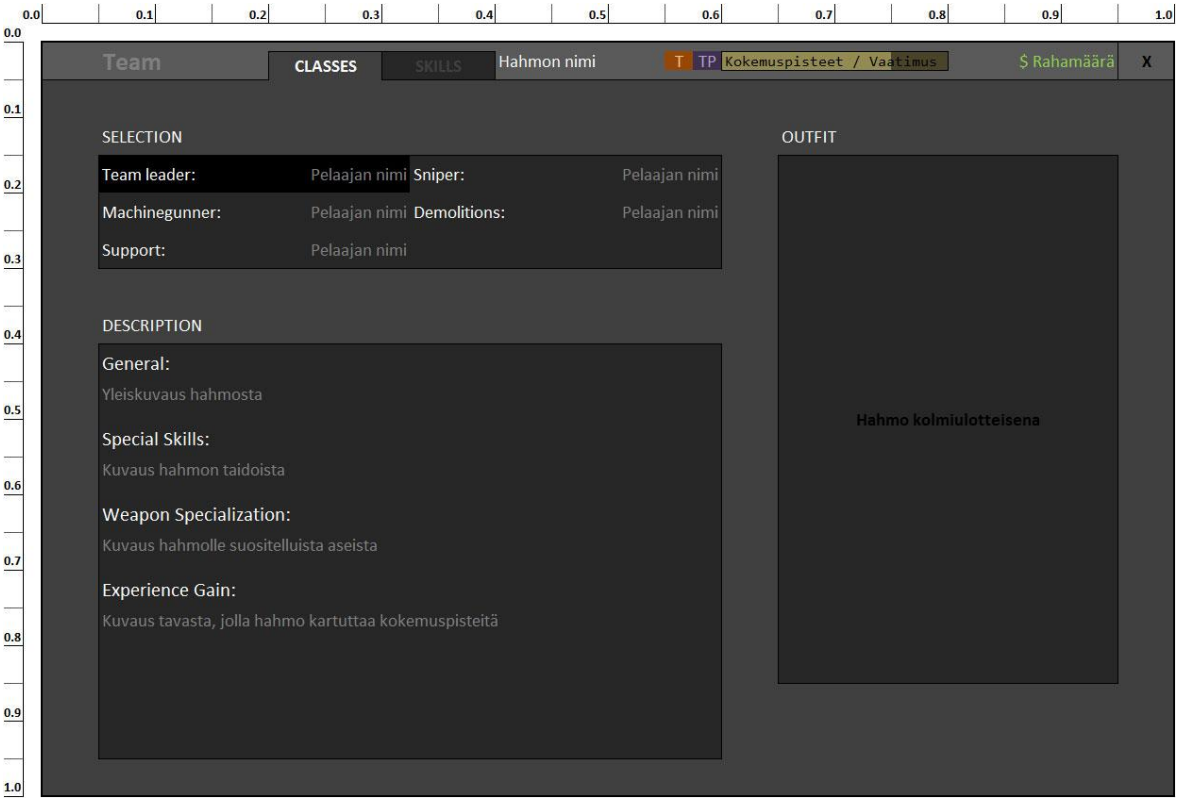
Kuvio 21. Näkymäsapluuna.

Värit ja tyylit. Näkymien taustaväreinä käytettiin kolmea harmaan sävyä. Näkymän otsikon väriksi valittiin harmaa ja muiden otsikko-elementtien värinä käytettiin valkoista. Pääotsikot kirjoitettiin isoilla kirjaimilla. Muun tekstin väriksi valittiin vaalean harmaa. Taso-elementin väriksi valittiin oranssi, jossa tausta on tumma ja numero vaalea. Taitopiste-elementti asetettiin käyttämään samaa kaavaa, mutta väriksi valittiin violetti. Kokemuspiste-elementissä käytettiin

etenemispalkki-tyyliä, jonka keskellä ilmoitetaan hahmon kokemuspisteet suhteessa seuraavan tason kokemuspistevaatimukseen. Kokemuspalkin taustan väriksi valittiin tumman ruskea sekä palkin väriksi tumman keltainen. Palkin tekstin väriksi valittiin musta. Pelaajan rahamäärä ilmoitetaan vihreällä tekstillä, jonka edessä esitetään $-merkki.

Valinta. Johnson (2008, 310) esittää, että järjestelmän on vastattava käyttäjän komentoihin korkeintaan 0,1 sekunnin viiveellä, muuten käyttäjä olettaa, että toimenpidettä ei ole vastaanotettu. Pelimuodon näkymiä varten luotiin uusi valinta-aktiivielementti. Se on perustoiminnaltaan Microsoftin standardia valintapainiketta (radio button) mukaileva, mutta visuaalisesti erilainen sekä sisältää kaksoistoiminnon: yhdesti vasenta hiiren painiketta napsautettua musta taustalaatikko siirtyy välittömästi kursorin alla olevan vaihtoehdon kohdalle korostaen valintaa. Tätä kutsutaan myöhemmin valitsemiseksi. Yhdesti napsauttaminen ei kuitenkaan aktivoi kyseistä vaihtoehtoa, vaan sillä voidaan tuoda esille valintaan liittyviä ominaisuuksia. Aktivoidakseen valinnan on pelaajan kaksoisnapsautettava vasenta hiiren painiketta haluamansa vaihtoehto-elementin päällä. Tätä kutsutaan myöhemmin aktivoimiseksi.

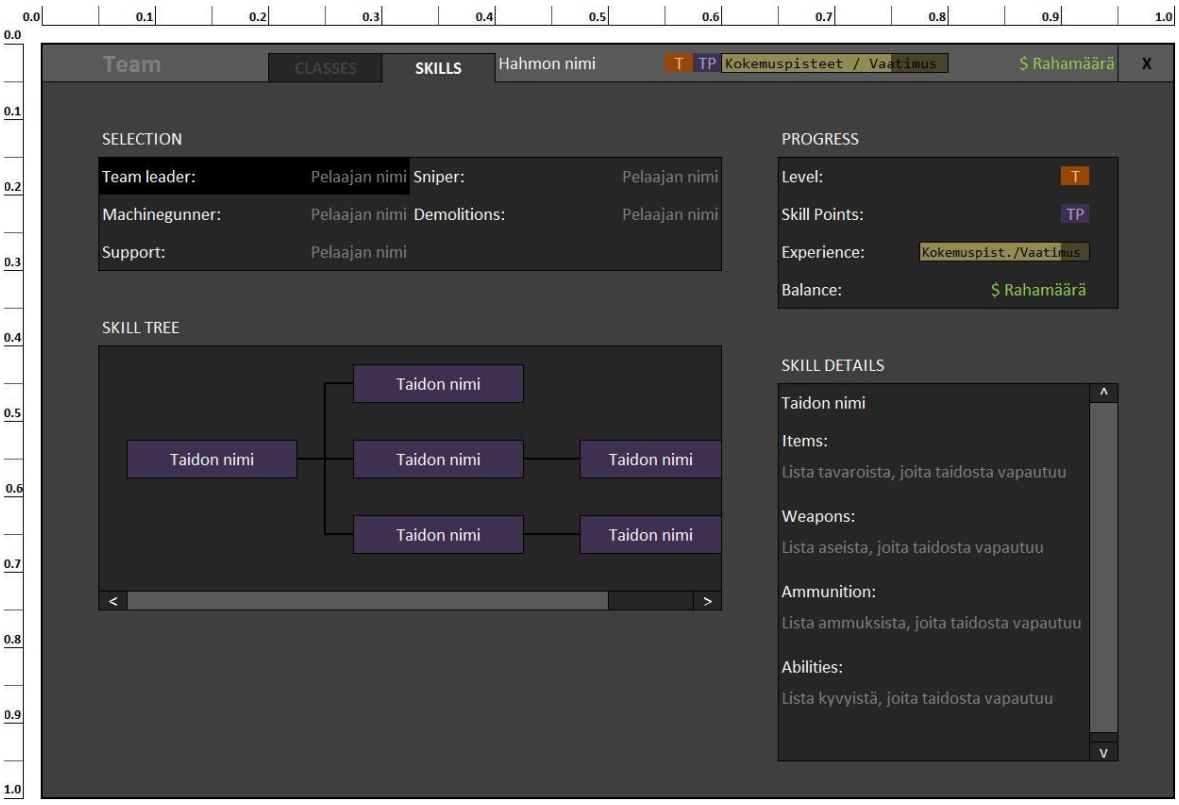
Ryhmänäkymä. Ryhmänäkymästä pelaajat voivat tutkia hahmojen ominaisuuksia, aktivoida haluamansa hahmon sekä valita hahmolleen taitoja. Ryhmänäkymä sisältää kaksi välilehteä: hahmot (classes) ja taidot (skills). Hahmot-välilehti (Kuvio 22) jaettiin kolmeen elementtiin: hahmovalinta (selection), kuvaus (description) sekä ulkoasu (outfit). Valinta-elementistä pelaajat aktivoivat itselleen hahmon. Yhdesti vaihtoehtoa napsauttamalla päivittyvät kuvaus- sekä ulkoasu-elementit vastaamaan valitun hahmon ominaisuuksia. Pelaajat aktivoivat hahmon kaksoisnapsauttamalla kyseistä vaihtoehtoa. Näin tehtyä pelaajan nimi siirtyy aktivoimansa hahmon kohdalle osoittaen pelaajille tekemänsä valinnan. Kuvaus-elementissä luetellaan hahmon pelimekaaniset ominaisuudet, kuten kyvyt, suositeltavat aseet sekä kokemuspisteiden saanti. Ulkoasu-elementissä pelaajan hahmo näytetään kolmiulotteisena mallina.



Kuvio 22. Hahmot-välilehti.

Ryhmänäkymän toinen välilehti, taidot (Kuvio 23), sisältää seuraavat elementit: hahmovalinta (selection), taitopuu (skill tree), kehitys (progress) sekä taidon yksityiskohdat (skill details). Johnson (2008, 23) kehottaa samojen käyttöliittymä -elementtien ja -toimintojen käyttöä yhteneviin operaatioihin, mikä tekee käyttöliittymästä yksinkertaisemman ja helpomman oppia. Hahmovalinta-elementin toiminta on sama kuin hahmot-välilehdellä, mutta kuvaus- ja ulkoasu-elementtien sijaan päivittyvät taitopuu-, kehitys-, sekä taidon yksityiskohdat -elementit vastaamaan hahmovalintaa. Taitopuusta pelaajat näkevät valitsemansa hahmon aktiiviset taidot sekä niitä seuraavat taidot. Taitopuu alkaa vasemmalta ja etenee oikealle. Taidot esitetään violetteina laatikoina sitoen ne värillä taitopiste -elementtiin. Jos pelaajalla on valinta-elementistä valittuna oma hahmonsa ja hahmolla on taitopisteitä, voi pelaaja kaksoisnapsauttamalla taitoa aktivoida sen, mikäli taitoon johtavat muut taidot on jo aktivoitu. Kehitys-elementti sisältää samoja elementtejä kuin otsikkopalkki: hahmon tason (level), hahmon taitopisteet (skill points), hahmon kokemuspisteet (experience) sekä pelaajan rahat (balance). Taso, taitopisteet sekä kokemuspisteet esittävät hahmovalinta-elementistä valitun

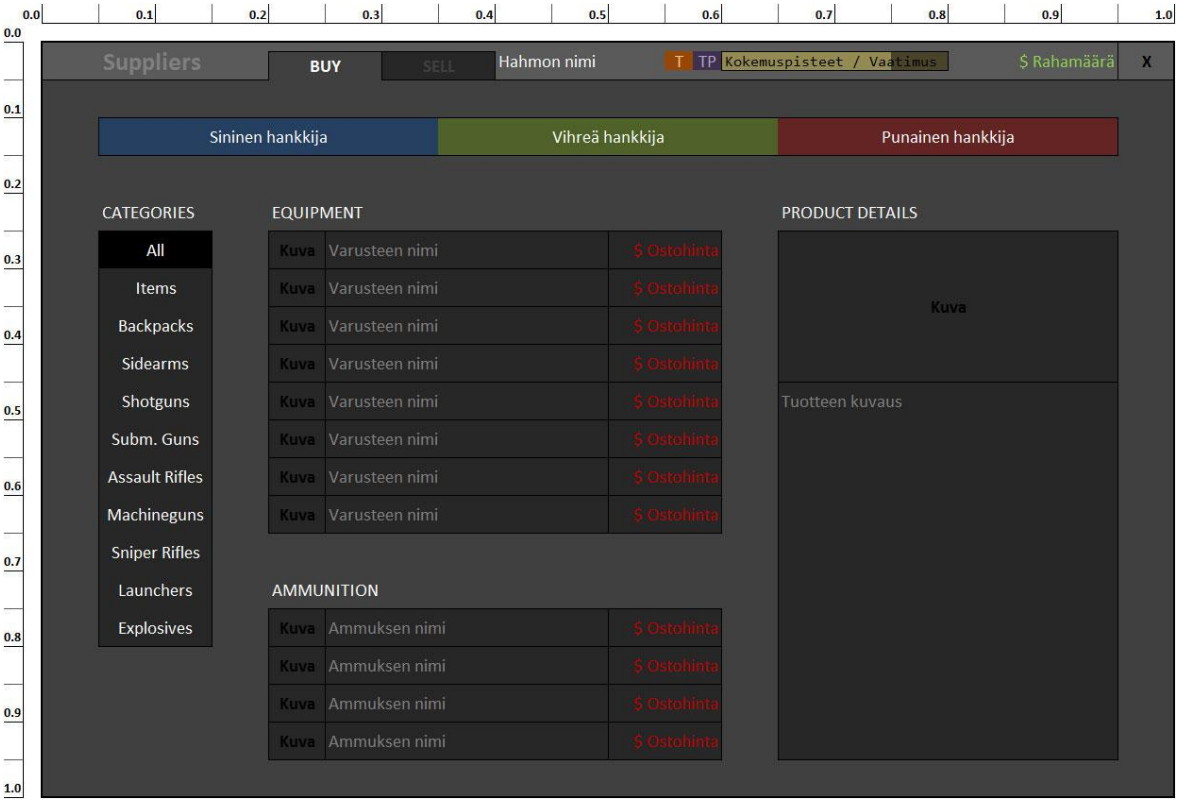
hahmon ominaisuuksia, kun taas otsikkopalkin elementit esittävät pelaajan aktivoiman hahmon ominaisuuksia. Yhdesti taitopainiketta napsauttamalla taidon yksityiskohdat -elementti päivittyy esittämään valitun taidon ominaisuuksia, kuten taidosta saatavia varusteita sekä kykyjä.



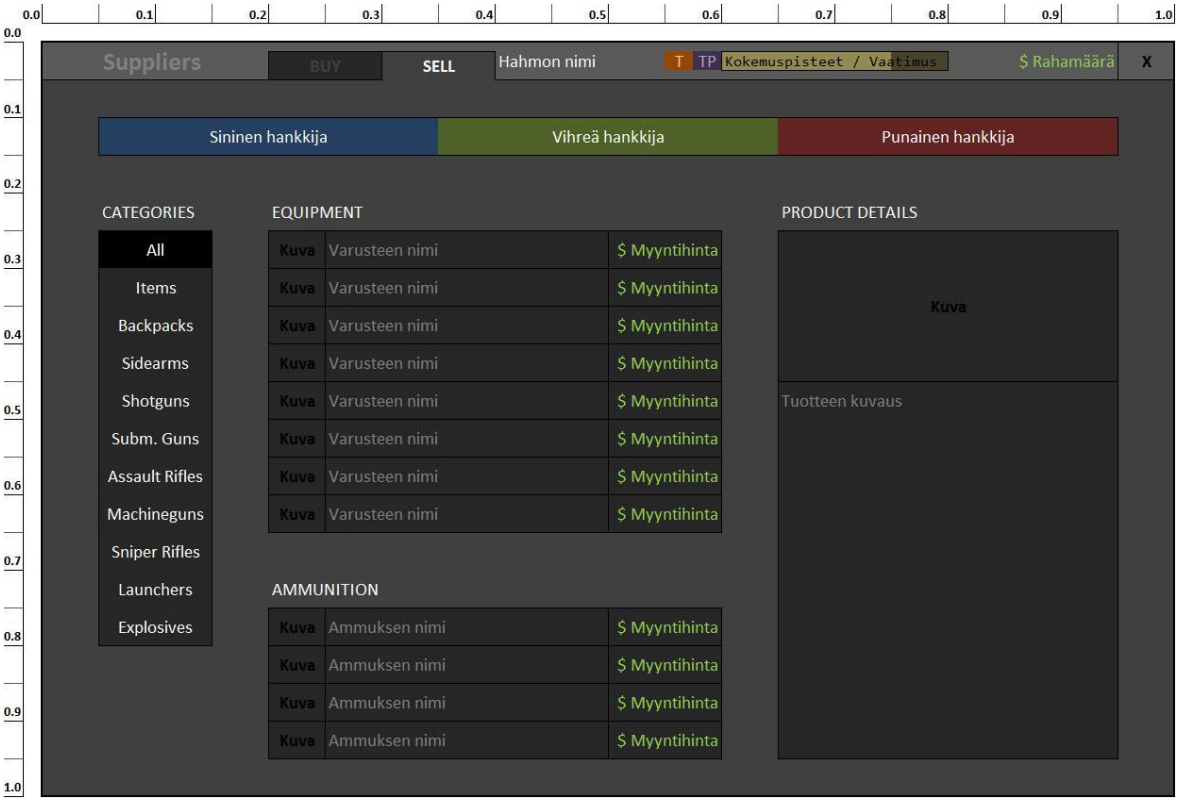
Kuvio 23. Taidot-välilehti.

Kauppanäkymä. Pelaaja ostaa sekä myy pelimuodossa esiintyviä varusteita kauppanäkymässä. Se sisältää kaksi välilehteä: osto (buy) ja myynti (sell). Molemmat välilehdet käyttävät samoja elementtejä, jotka ovat: hankkijavalitsin, kategoriavalitsin (categories), varustelista (equipment), ammuslista (ammunition) sekä tuotteen yksityiskohdat (product details). Hankkijavalitsin toteuttaa uutta valinta-elementtiä ja sillä pelaajat valitsevat, minkä hankkijan varusteita he haluavat selata. Kategoriavalitsin toteuttaa samaa valinta-elementtiä ja sillä pelaajat valitsevat selattavien varusteiden kategorian. Hankkijavalitsimen oletusarvo on sininen ja kategoriavalitsimen oletusarvo on kaikki (all). Pelaajan valittua uusi hankkija tai kategoria varustelista- sekä ammuslista-elementit päivittyvät esittämään valitulta hankkijalta (osto-välilehti) tai pelaajalta (myynti-

välilehti) löytyviä varusteita, jotka on luokiteltu valittuun kategoriaan. Varuste- sekä ammuslistat on jaettu kolmeen pystysarakkeeseen, joissa näkyvät varusteen kuva, nimi sekä hinta. Molemmat lista-elementit toteuttavat samaa kaksitoimisuutta, joka löytyy valinta-elementeistä. Yhdesti hiirtä napsauttamalla pelaaja voi valita varusteita tai ammuksia, jolloin niiden ominaisuudet näkyvät tuotteen yksityiskohdat -elementissä. Kahdesti hiirtä napsauttamalla pelaaja voi ostaa (osto-välilehti) tai myydä (myynti-välilehti) valitun varusteen. Osto-välilehteä (Kuvio 24) sekä myynti-välilehteä (Kuvio 25) erottava tekijä on se, että osto-välilehdessä varusteiden hinta näytetään punaisena, mikä kuvaa negatiivistä vaikutusta pelaajan rahamäärään ja myynti-välilehdellä hinta näytetään vihreänä, mikä kuvaa positiivista vaikutusta pelaajan rahamäärään. Näin välilehdet erotetaan visuaalisesti toisistaan.

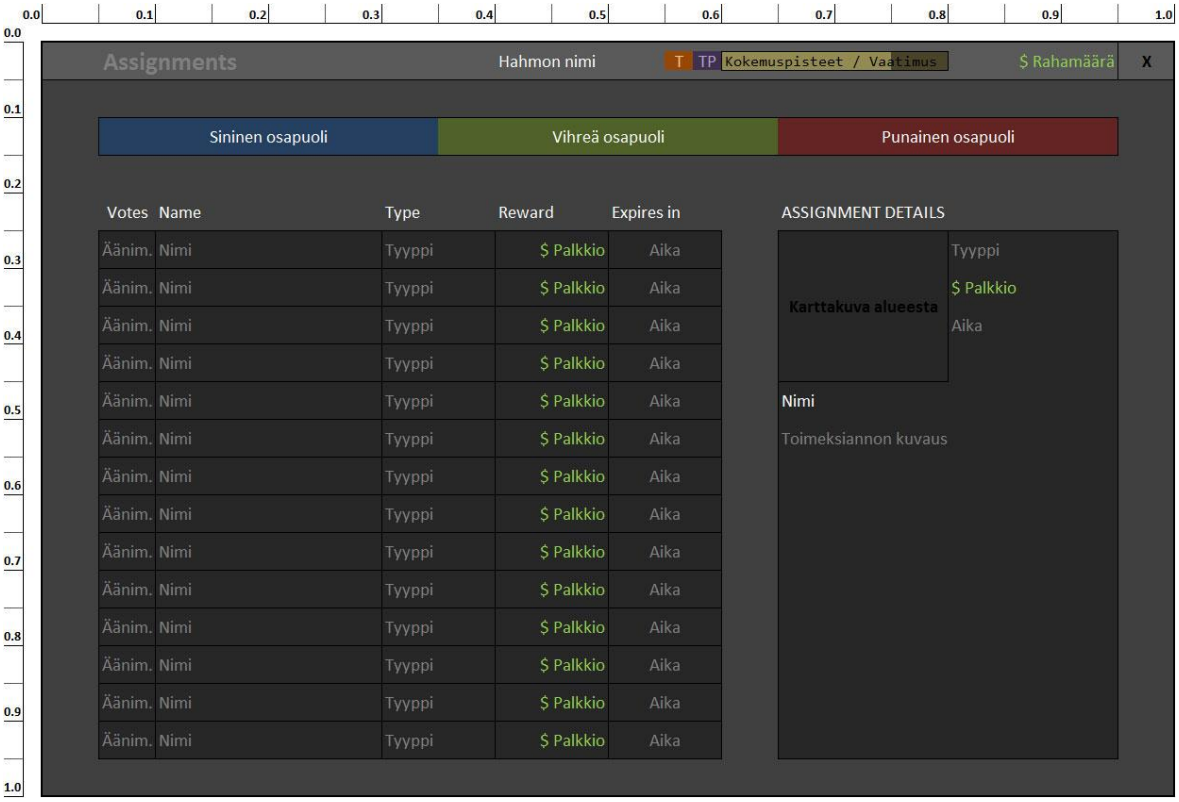


Kuvio 24. Osto-välilehti.



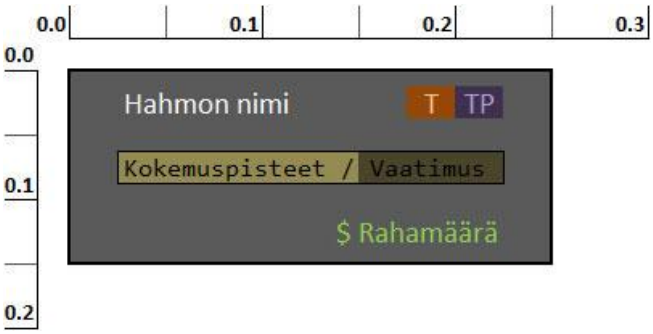
Kuvio 25. Myynti-välilehti.

Toimeksiantonäkymä. Pelaajille tarjolla olevat toimeksiannot esitetään toimeksiantonäkymässä (Kuvio 26). Siinä pelaajat voivat myös äänestää seuraavaksi pelattavaa toimeksiantoa. Toimeksiantonäkymän elementit ovat: osapuolivalitsin, toimeksiantolista sekä toimeksiannon yksityiskohdat (assignment details). Osapuolivalitsimen toiminta on identtinen kauppa-näkymän hankkijavalitsimen toimintaan. Pelaajan valittua osapuolen päivittyy toimeksiantolista esittämään valitun osapuolen tarjoamat toimeksiannot. Toimeksiantolista on jaettu viiteen pystysarakkeeseen, joissa näkyvät toimeksiannon saama äänimäärä, nimi, tyyppi, palkkio sekä kuinka kauan toimeksianto on tarjolla. Pelaajan valittua listasta toimeksianto päivittyy sen ominaisuudet toimeksiannon yksityiskohdat -elementtiin. Kaksoisnapsauttamalla listassa olevan toimeksiannon riviä antaa pelaaja sille äänensä. Tämä näkyy pelaajalle siten, että äänimäärä kyseisen toimeksiannon kohdalla muuttuu vaalean siniseksi. Pelaaja voi poistaa äänensä kaksoisnapsauttamalla saman toimeksiannon riviä uudelleen tai siirtää äänensä uuteen toimeksiantoon kaksoisnapsauttamalla sen riviä.



Kuvio 26. Toimeksiantonäkymä.

Tilannenäkymä. Pelaaja voi seurata edistymistään pysyvästä tilannenäkymästä (Kuvio 27). Tilannenäkymä sisältää samat tieto-elementit kuin väliaikaisten näkymien otsikkopalkki: pelaajan valitseman hahmon nimen, hahmon tason, hahmon taitopisteet, hahmon kokemuspisteet sekä pelaajan rahamäärän. Tilannenäkymä pysyy näytöllä koko peli-istunnon ajan.



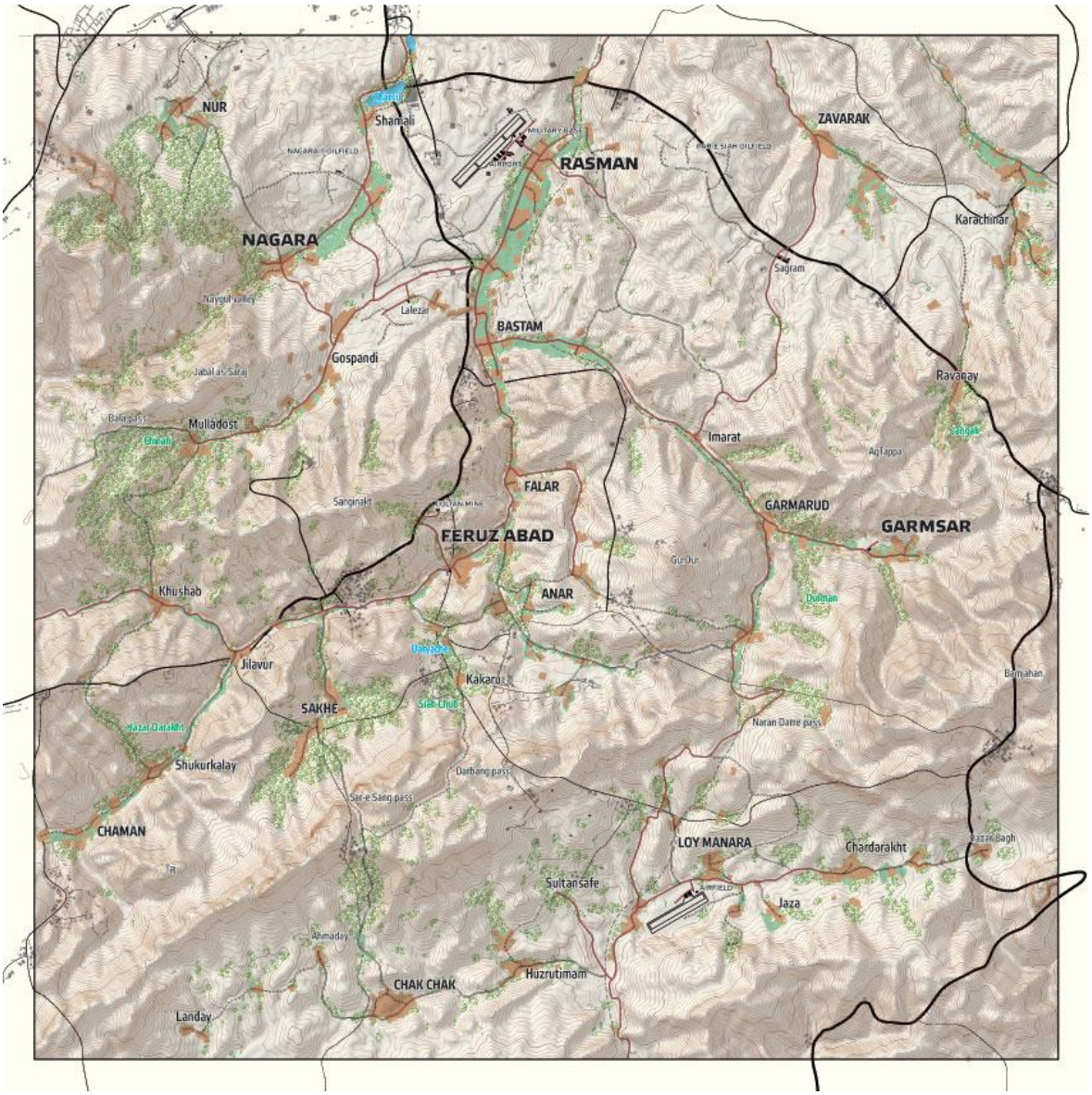
Kuvio 27. Tilannenäkymä.

4.3.2 Tapahtumapaikat

Schell (2008, 333–334) kirjoittaa, että hyvässä peliympäristössä on maamerkkejä, joista pelaaja tunnistaa pelin kannalta tärkeät kohteet ja jotka tekevät ympäristöstä mielenkiintoisen katsella.

Tukikohta. Tukikohdassa pelaajille annetaan mahdollisuus valmistautua seuraavaa toimeksiantoa varten vaarattomasti, joten sen täytyy tuntua turvalliselta. Siellä pelaajat voivat myös hankkia hahmoilleen varusteita, joten on uskottavampaa, jos tukikohtaan on loogiset kulkuyhteydet. Näiden asioiden vuoksi tukikohta päätettiin sijoittaa peliympäristöstä löytyvän suuren sotilaslentokentän viereen. Sotilaiden läsnäolo tuo turvallisuuden tunnetta, ja lentokoneiden laskeutuminen ja nousu loogisen kaupankäynnin mahdollisuuden. Pelaajat vaihtavat hahmoaan tukikohdassa, joten oletuksena on, että hahmot asuvat siellä. Tämän vuoksi ryhmänäkymän maamerkiksi valittiin parakkirakennus. Parakin lähelle päätettiin luoda myös muita kotoisia objekteja, kuten kaivo sekä vessat. Kauppanäkymän maamerkiksi valittiin metallikontti, jonka edessä on kasa ammuslaatikoita. Näin pelaaja assosioi kohteen välittömästi varusteisiin. Toimeksiantonäkymän maamerkiksi valittiin karttataulu, jonka edessä on iso pöytä sekä viisi tuolia: yksi jokaiselle hahmolle. Näin pelaajalle annetaan kuva käskynjakopaikasta, joka on helppo yhdistää toimeksiantoihin.

Asutuskeskukset. Toimeksiantojen tapahtumat sijoitetaan asutuskeskuksiin. Kun toimeksianto luodaan, sille valitaan tapahtumapaikaksi satunnainen asutuskeskus. Peliympäristössä on 28 valmiiksi rakennettua asutuskeskusta (Kuvio 28), jotka sisältävät rakennuksia, aitoja sekä muita pysyviä objekteja. Asutuskeskukset pyritään pitämään muuttamattomina koko peli-istunnon ajan, sillä pelaajien toivotaan oppivan niissä navigointi peli-istunnon edetessä. Myös kaikki pelaajien tekemä vaurio asutuskeskuksiin pysyy koko peli-istunnon ajan. Tämä tuo pelaajille tunteen, että he pystyvät vaikuttamaan peliympäristöön.



Kuvio 28. Takistanin kartta (ArmA 2 OA Takistan Map 2011).

4.4 Tarina

> ”Hyvä peli on kone, joka tuottaa tarinoita ihmisten sitä pelatessa” (Schell 2008, 266).

Pelimuodon tarinatyyliksi valittiin niin kutsuttu tarinakone, jossa peli ei sisällä esimäärättyä tarinaa, mutta esittää pelaajalle suuren määrän vaihtoehtoja, joista valitsemalla pelaajat luovat oman tarinansa. Tarinakone-tyyliä käyttäviä pelejä ovat esimerkiksi Transport Tycoon (1994) sekä The Sims (2000). Näissä peleissä ei ole käsikirjoitettua tarinaa, mutta pelaaja voi voikuttaa pelin kulkuun niin monin

tavoin, että jokainen pelikerta sisältää toisistaan eroavia kiinnostavia tapahtumia, jotka käsitetään pelin tarinana. Työn pelimuoto tarjoaa monia mahdollisuuksia pelaajille valintojen tekemiseen. Pelaajat voivat mm. valita

- millä hahmolla he pelaavat
- mitä toimeksiantoa he pelaavat
- mitkä toimeksiannon työtehtävät he suorittavat
- mitä varusteita he tuovat tukikohtaan toimeksiannosta
- mitkä varusteet he myyvät ja mitkä pitävät itsellään
- mitä taitoja he aktivoivat hahmolleen.

Pelimuodon teema, estetiikka sekä pelimekaniikka antavat puitteet lukemattomiin toisistaan eroaviin pelikokemuksiin ilman etukäteen luotua tarinaa: Hahmojen pelimekaaninen erilaisuus ja toimeksiantojen satunnaisuus tuovat pelimuotoon vaihtelevuutta peli-istuntojen välille ja hahmojen kartuttama kokemus antaa pelaajille saavuttamisen tunteen, sillä sen tuomat uudet taidot elävöittävät tarinaa peli-istunnon edetessä.

5 PELIMUODON TOTEUTUS

Toteutusosassa pelimuotoon suunnitellut ominaisuudet luotiin käyttämällä edellä mainittuja työkaluja. Toteutus suoritettiin vaiheittain, joiden välissä pelimuotoa testattiin. Ohjelmoinnissa keskityttiin yksinkertaisten ja nopeiden funktioiden sekä skriptien luomiseen ja selkeään koodiin, jonka rivit kommentoitiin tulevaisuuden muokkausta silmällä pitäen.

5.1 Ensimmäinen vaihe

Toteutuksen ensimmäisessä vaiheessa luotiin hyvä pohja tulevia toimintoja varten. Erityistä huomiota annettiin standardien määrittelyyn, jotta pelimuodon kehittäminen sekä laajentaminen on tulevaisuudessa mahdollista mahdollisimman helposti ja tehokkaasti. Ensimmäisessä vaiheessa luotiin myös pelaajien ja hahmojen hallinnan perustoiminnot, tukikohdan luontiin liittyvät funktiot sekä hahmot-välilehden graafinen ja toiminnallinen sisältö.

5.1.1 Teknologia

Hakemistorakenne. Pelimuodon tekeminen aloitettiin luomalla sille projektihakemisto, johon sijoitettiin **mission.sqm**-tehtävätiedosto, **description.ext** -alustustiedosto sekä **init.sqf**-komentotiedosto. Pelimuodolle luotiin oma alihakemisto nimeltään **assignment** projektihakemiston sisään. Pelimuodon hakemistoon luotiin eri tiedostotyypeille alihakemistot: alustustiedostoille **config**-hakemisto, komentotiedostolle **content**-hakemisto ja kuvatiedostoille **images**-hakemisto. **Assignment**-hakemistoon luotiin myös **initialization.sqf** -komentotiedosto, joka ajamalla pelimuoto käynnistetään.

Sisällönhallinta. Pelimuotoon oletettiin luotavan suuri määrä funktioita sekä skripteja, jotka täytyisi alustaa palvelimella sekä jokaisella asiakaskoneella pelimuodon käynnistyksessä. Näin ollen luotiin uusi sisällönhallintajärjestelmä, jossa funktiot jaettiin kahteen pääluokkaan riippuen niiden toimintaympäristöstä: palvelinfunktiot sekä asiakasfunktiot. Palvelinfunktioita ajetaan vain palvelimelta ja

asiakasfunktioita vain asiakaskoneilta. Molemmissa ajettavat funktiot määriteltiin palvelinfunktioiden pääluokkaan. Pääluokat jaettiin vielä alaluokkiin funktioiden käyttötarkoituksen mukaan. Kaikki funktiot määriteltiin komentotiedostoon **content.sqf**, joka ajetaan sekä palvelimella että asiakaskoneilla **initialization.sqf**-käynnistystiedostosta. Funktioiden lisäksi **content.sqf** sisältää silmukan, jossa funktiot alustetaan ja asiakaskoneiden tapauksessa odotetaan niin kauan, että palvelin on alustanut omat sekä asiakaskoneiden kanssa yhteiset funktiot. Funktioiden alustamisen jälkeen luodaan tukikohta sekä hahmojen yksiköt.

Koska funktiota, skripteja sekä muuttujia oletettiin luotavan suuri määrä samaan nimiavaruuteen, kehitettiin niiden nimeämiselle standardi. Se koostuu seuraavista osista, jotka kaikki kirjoitetaan isoilla kirjaimilla ja liitetään toisiinsa alaviivalla. Funktioiden ja skriptien tapauksessa nimi koostuu seuraavista osista:

1. lyhenne **ANT**, joka tulee pelimuodon nimestä: **A**ssig**N**men**T**
2. funktion tapauksessa **F**-kirjain ja skriptin tapauksessa **S**-kirjain
3. käyttöalue esim. **PLAYERS**, **CLASSES**, **SKILLS**, **SUPPLIERS** tai **ASSIGNMENTS**
4. toiminto esim. **GET**, **SET**, **ADD** tai **UPDATE**
5. toiminnon kohde, (useiden sanojen tapauksessa erotettuna alaviivoilla).

Esimerkkifunktio: **ANT_F_PLAYERS_GET_UNITS**

Esimerkkiskripti: **ANT_S_ ASSIGNMENTS_UPDATE_ASSIGNMENTS_LIST**

Muuttujien tapauksessa nimi koostuu seuraavista osista:

1. pelimuodon lyhenne **ANT**
2. tyyppi, joita ovat **BOOL**, **NUMBER**, **ARRAY** sekä **STRING**
3. käyttöalue
4. sisällön kuvaus (useiden sanojen tapauksessa erotettuna alaviivoilla).

Esimerkkimuuttuja: **ANT_NUMBER_ASSIGNMENT_COUNTDOWN_TIMER**

Pelaajienhallinta. Hahmojen ominaisuudet, kuten kokemuspisteet, taitopisteet sekä tasot ovat hahmokohtaisia, joten ne päätettiin tallentaa peli-istunnon ajaksi palvelimelle. Pelaajan rahat sekä varusteet sitä vastoin ovat pelaajakohtaisia, joten pelaajan poistuessa peli-istunnosta ne katoavat. Pelaajan myöhemmin palatessa palvelimelle hänen olisi siis aloitettava näiden resurssien kerääminen tyhjästä. Näiden ominaisuuksien haluttiin säilyvän koko istunnon ajan, joten ne päätettiin myös tallentaa palvelimelle. A2: CO sisältää komennon **getPlayerUID**, jolla voidaan palauttaa pelaajan yksilöllinen ID-numero. Tätä komentoa käyttäen tarkistetaan pelaajan ID-numeroa vastaavasta taulusta, onko pelaajalla palvelimelle luotuja henkilökohtaisia ominaisuuksia. Jos pelaaja yhdistää palvelimelle ensimmäistä kertaa, luodaan hänen ID-numeroansa vastaava taulu, johon pelaajan henkilökohtaiset ominaisuudet tallennetaan myöhemmin ladattavaksi.

5.1.2 Pelimekaniikka

Hahmot. Käynnistyksessä jokaiselle pelaajalle aktivoidaan automaattisesti hahmo ja luodaan sitä ilmentävä yksikkö. Jokaiselle viidelle hahmolle luodaan sen omat perusmuuttujat palvelimelle. Muuttujat ovat

- level, joka sisältää hahmon tason
- skillpoints, joka sisältää hahmon taitopisteet
- experience, joka sisältää hahmon kokemuspisteet sekä
- available, joka kertoo onko hahmo vapaana vai aktiivisena yhdellä pelaajista.

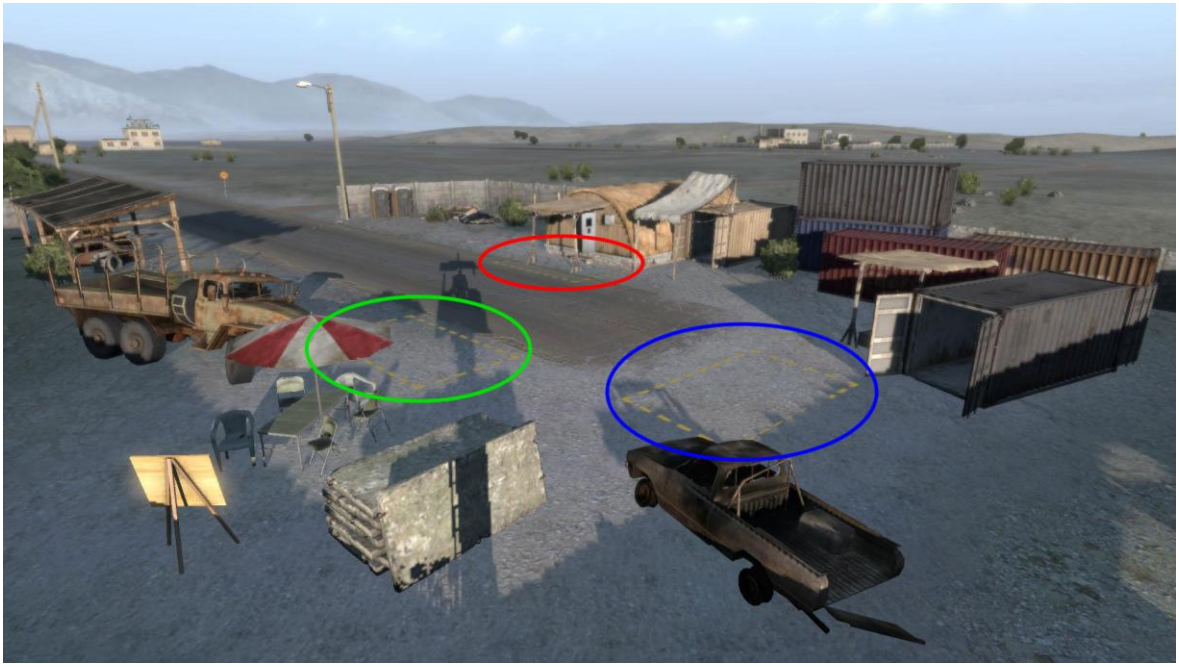
Available-muuttuja palauttaa arvon true, jos kyseinen hahmo on vapaa ja false, jos joku pelaajista on aktivoinut kyseisen hahmon. Pelaajan vaihtaessa hahmoa ajetaan funktio **ANT_F_CLASSES_SELECT_CLASS** (LIITE 1), joka tarkistaa onko uuden hahmon available-muuttuja true. Jos näin on, luodaan pelaajalle uutta hahmoa mallintava yksikkö, johon pelaajan kontrolli siirretään. Pelaajan vanha yksikkö poistetaan ja sen hahmon available-muuttujan arvoksi vaihdetaan true. Näin yhtä hahmoa voi kontrolloida vain yksi pelaaja kerrallaan. Hahmoille luotiin

myös oma luokka alustustiedostoon nimeltään **Classes**, jossa määritellään mm. hahmojen nimet sekä kuvaukset.

Kokemus. Kokemuspisteiden, taitopisteiden sekä tasojen hallintaa varten luotiin kaksi funktiota, joista toisella lisätään hahmolle taitopisteet ja toisen avulla testataan, josko taitopisteiden lisäys saavutti hahmolle uuden tason. Molemmat funktiot ajetaan palvelimella. **ANT_F_CLASSES_ADD_EXPERIENCE**-funktion (LIITE 2) parametreina syötetään hahmon ID-numero, joka on nollasta neljään (ryhmänjohtajan ollessa nolla ja räjähde-asiantuntijan neljä), sekä lisättävien kokemuspisteiden määrä. Funktio lisää pisteet hahmolle ja käyttäen funktiota **ANT_F_CLASSES_GET_GOAL_EXPERIENCE** (LIITE 3) laskee, josko hahmon uusi kokemuspistemäärä ylitti uuden tason rajan. Jos uusi taso saavutettiin, lisätään hahmon taso- sekä taitopiste-muuttujia yhdellä.

5.1.3 Estetiikka

Tukikohta. Funktioiden alustamisen jälkeen luodaan tukikohdan objektit sekä tapahtumankäsittelijät skriptillä **ANT_S_ASSIGNMENTS_INIT_BASE** (LIITE 4). Skripti ajetaan niin palvelimella kuin asiakaskoneillakin, mutta sen käyttäytyminen molemmilla on erilainen: Palvelimella skripti luo tukikohdan maamerkit sekä muut objektit ja siirtää kaikkien pelaajien yksiköt tukikohdan keskelle, kun taas asiakaskoneilla se luo asiakaskohtaiset tapahtumankäsittelijät maamerkkien eteen, joista maamerkkeihin sidotut näkymät avautuvat (Kuvio 29). Kuviossa on punaisella merkattuna alue, josta ryhmänäkymä aukeaa. Kauppanäkymän aukeamisalue on merkattu sinisellä. Vihreällä merkatusta alueesta aukeaa toimeksiantonäkymä. Taustalla näkyy tukikohdan vieressä sijaitseva lentokenttä.



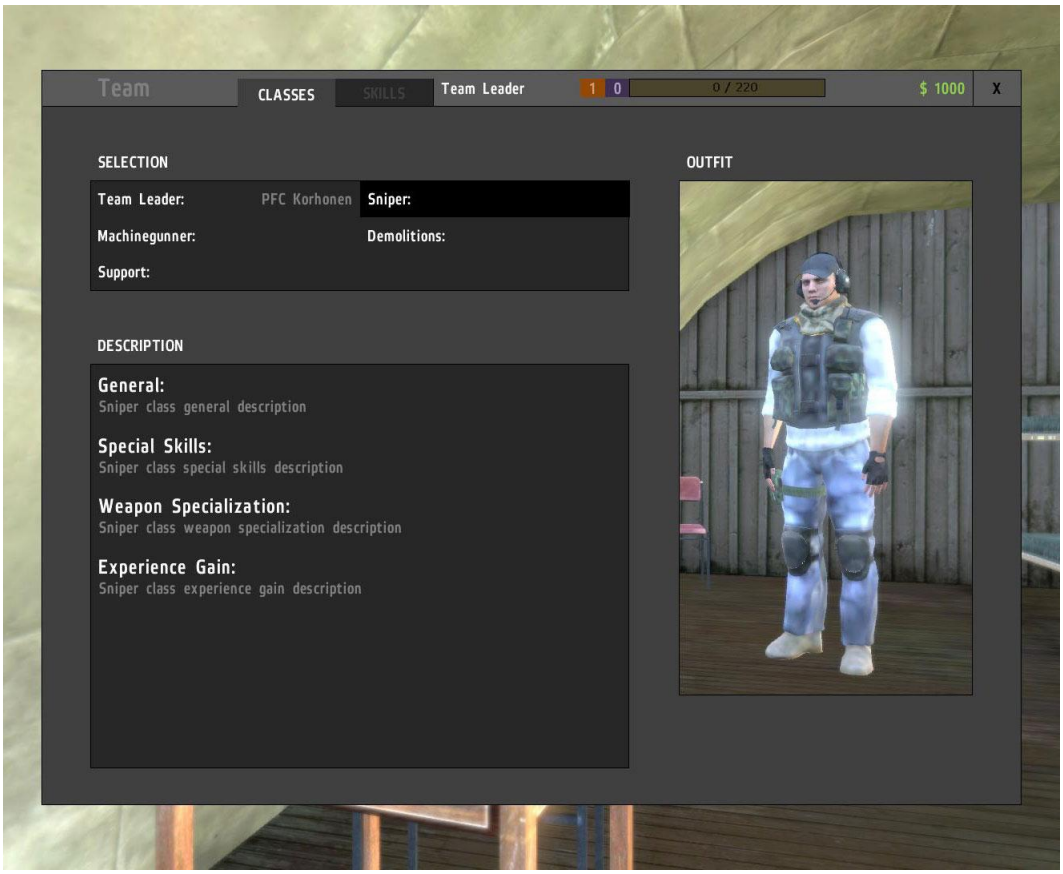
Kuvio 29. Pelimuodon tukikohta.

Hahmot-välilehti. Aluksi luotiin hahmovalinta-elementti, jolla pelaaja voi selata hahmojen ominaisuuksia sekä aktivoida itselleen hahmon. Jokainen viidestä valinnasta koostettiin kahdesta teksti-elementistä sekä yhdestä näkymättömästä painike-elementistä. Valkoisessa teksti-elementissä esitetään hahmon nimi, joka haetaan **Classes**-luokasta, ja harmaassa hahmon aktivoineen pelaajan nimi. Jos hahmo ei ole aktiivisena, harmaata tekstiä ei esitetä. Valintaa osoittamaan luotiin musta tausta-elementti, joka sijoitettiin tekstien sekä painikkeiden alle. Painikkeille luotiin kaksi hiiren painikkeen tapahtumankäsittelijää: yhden napsautuksen käsittelijä sekä kahden napsautuksen käsittelijä. Yhden napsautuksen käsittelijä asettaa hahmovalinta-muuttujan **ANT_NUMBER_CLASSES _SELECTION** arvon vastaamaan napsautetun hahmon ID-numeroa ja ajaa skriptin **ANT_S_CLASSES_UPDATE_CLASS_SELECTION** (LIITE 5), joka siirtää mustan tausta-elementin napsautetun painikkeen alle osoittaakseen tehtyä valintaa. Skripti päivittää myös kuvaus-, ulkoasu-, taitopuu-, kehitys- sekä taidon yksityiskohdat -elementit vastaamaan, välilehdestä riippuen, valitun hahmon ominaisuuksia. Myös harmaat teksti-elementit päivitetään vastaamaan hahmoja kontrolloivia pelaajia. Kahden napsautuksen käsittelijä ajaa funktion **ANT_F_CLASSES_SELECT_CLASS** ja antaa sille parametrina

kaksoisnapsautetun hahmon ID-numeron. Funktio tarkistaa hahmon saatavuuden ja hahmon ollessa vapaa, aktivoi sen pelaajalle. Tämän jälkeen ajetaan jälleen skripti **ANT_S_CLASSES_UPDATE_CLASS_SELECTION**, joka päivittää valinnan.

Kuvaus-elementti luotiin pelkästä teksti-elementistä, jossa otsikot esitetään valkoisella ja kuvaus-kappaleet harmaalla värillä. Kuvausta päivitettäessä tarkistetaan valittu hahmo **ANT_NUMBER_CLASSES _SELECTION**-muuttujasta ja hahmon kuvaus-merkkijonot haetaan **Classes**-luokasta.

Ulkoasu-elementti on itsessään vain pelkkä aukko näkymän taustavärissä. Koska A2: CO ei sisällä komentoa kolmiulotteisten mallien esittämiseen näkymissä, oli hahmo esitettävä näkymän taustalla peliympäristössä (Kuvio 30). A2: CO sallii virtuaalisten kameroiden luomisen peliympäristöön, joiden näkökulma voidaan siirtää pelaajan ruudulle. Luomalla virtuaalinen kamera ja esittämällä sen näkökulma näkymän taustalla voitiin ulkoasu-esikatselu eristää tukikohdan tapahtumista. Peli-istuntoon liittyessä jokaiselle pelaajalle luotiin oma sijainti korkealta peliympäristön nurkasta, jonne pelaajien oli mahdotonta nähdä tai päätyä. Sijaintiin sijoitettiin samanlainen parakki-objekti, jota käytetään ryhmänäkymän maamerkkinä. Virtuaalinen kamera sijoitettiin kuvaamaan parakin sisälle. Tämän antaa pelaajalle mielikuvan, että avatessaan ryhmänäkymän parakin edestä hän siirtyy sen sisälle, vaikka todellisuudessa sijainnit ovat kaukana toisistaan. Parakin sisälle määritettiin sijainti, johon pelaajan valitseman hahmon yksikkö luodaan esikatseltavaksi. Kun pelaaja valitsee uuden hahmon, haetaan sitä mallintava yksikkö Classes-luokasta ja luodaan se parakin sisälle. Koska pelaajalla saattaa olla mikä resoluutio tai näkymäkoko tahansa, oli kamera sijoitettava siten, että luotu yksikkö osuu ulkoasu-elementin aukon keskelle oikean kokoisena. Tätä varten luotiin skripti **ANT_S_CLASSES_OPEN_CLASS_OUTFIT** (LIITE 6), joka asettaa näkymän taustan mustaksi, luo kameran sekä käyttää hyväkseen komentoa **worldToScreen**, jolla annettu peliympäristön sijainti muokataan näkymän sijainniksi. Skripti siirtää kameraa, kunnes yksikön päälaen sekä jalkapohjien sijainti on ulkoasu-elementin sisällä. Kun yksikkö on elementin sisällä, näkymän musta tausta häivytetään pois paljastaen näin esikatseltava yksikkö.



Kuvio 30. Hahmot-välilehti pelissä.

5.1.4 Testaus

> ”Pelitestauksella on kaksi tarkoitusta: etsiä pelikoodista tai suunnittelusta virheitä ja esittää, mitkä osat pelistä toimivat kunnolla” (Schultz & Bryant 2011, 23).

Ensimmäisen vaiheen testaus päätettiin suorittaa lokaalisti, eli vain yhdellä tietokoneella. Testaus suoritettiin käynnistämällä A2: CO:n palvelinohjelma ja avaamalla siihen pelimuodon testiversio. Tämän jälkeen palvelimelle liityttiin samalla tietokoneella asiakkaana. Testauksen aiheina olivat hahmon valinta sekä aktivoiminen ja näkymien selkeys sekä toiminnallisuus. Siinä haettiin vastauksia seuraaviin kysymyksiin:

- Onnistuuko pelaajan tietojen tallentaminen palvelimelle ID-numeron avulla?
- Toimiiko hahmon valitseminen ja vaihtaminen ongelmitta?

- Pystytäänkö hahmolle lisäämään kokemuspisteitä ja toimiiko taso-järjestelmä?
- Syntyvätkö tukikohdan objektit oikeisiin paikkoihin ja onko tukikohdassa helppo suunnistaa?
- Toimiiko ryhmänäkymän hahmovalinta-elementti oikein ja näyttävätkö muut elementit oikeaa tietoa?

Testaamalla havainnoitiin, että pelaajatietojen tallentaminen palvelimelle onnistuu ongelmitta sekä hahmon valintaan tai vaihtamiseen ei liity ongelmia. Kokemuspiste- sekä taso-järjestelmissä havaittiin kuitenkin virheitä: taso lisääntyi aina, kun kokemuspisteet lisääntyivät. Funktiosta **ANT_F_CLASSES_GET_GOAL_EXPERIENCE**, jonka tehtävänä oli ilmoittaa seuraavaan tasoon vaadittava kokemuspistemäärä, löytyi kirjoitusvirhe, jonka seurauksena funktio palautti aina nollan. Virhe korjattiin onnistuneesti. Suurin osa tukikohdan objekteista syntyi oikealle paikalle, mutta muutamaa jouduttiin siirtämään. Ryhmänäkymän hahmovalinta- sekä kuvaus-elementin toiminta oli odotetun mukaista, mutta ulkoasu-elementissä havaittiin outo virhe. Jostain syystä parakki-objekti, jonka sisälle esikatseltava yksikkö luodaan, oli noin 30 asteen sivuttaisessa kulmassa kameraan nähden. Päivien virheen etsimisen jälkeen mitään siihen viittaavaa ei koodista kuitenkaan löydetty. Virhe ei vaikuttanut pelimuodon toimintaan muuten kuin ulkonäöllisesti, joten se jätettiin myöhemmin paikallistettavaksi.

5.2 Toinen vaihe

Toteutuksen toisessa vaiheessa keskityttiin taito- sekä kauppajärjestelmien luomiseen niin pelimekaanisesti kuin esteettisesti. Huomiota kiinnitettiin tällä kertaa erityisesti toimintojen moninpeli-käyttäytymiseen.

5.2.1 Pelimekaniikka

Taidot. Pelimuotoon sisällytettiin useita eri taitoja, joista jokaisella oli erilainen vaikutus pelaajahahmojen ryhmään tai yksittäiseen pelaajahahmoon. Taitojen

päätarkoituksena pidettiin erinäisten varusteiden sallimista pelaajien ostettavaksi. Taidoille luotiin oma alustustiedoston luokka **Skills**, johon jokaiselle taidolle luotiin oma luokkansa. Taitoluokille luotiin perusominaisuuksia, kuten taidon omistajahahmo, taidon aktivoimiseen vaadittavat taidot, taidon nimi, taidon kuvaus sekä lista varusteista, joita taito tuo kauppaan ostettavaksi. Koska taidot ovat uniikkeja, eli jokaista taitoa on pelimuodossa vain yksi, luotiin niitä varten yleinen muuttuja **ANT_ARRAY_SKILLS_SELECTED_SKILLS**, joka sisältää kaikkien hahmoille aktivoitujen taitojen luokkanimet. Tästä muuttujasta voitiin missä tahansa pelin vaiheessa tarkistaa, josko tietty taito on aktivoituna. Kun pelaaja aktivoi hahmolleen taidon, lisätään sen luokkanimi edellä mainittuun muuttujaan.

Taitojen aktivointiin luotiin funktio **ANT_F_SKILLS_SELECT_SKILL** (LIITE 7), joka ajetaan taidon aktivoiman pelaajan asiakaskoneella ja sille syötetään taidon luokkanimi. Funktio tarkistaa onko hahmolla taitopisteitä taidon aktivoimiseen, etsii taidon **Skills**-luokasta ja tarkistaa, onko hahmolle aktivoitu kaikki taidon vaatimat muut taidot. Jos näin on, hahmolta vähennetään yksi taitopiste, ja taito aktivoidaan lisäämällä se yleiseen muuttujaan **ANT_ARRAY_SKILLS_SELECTED_SKILLS**.

Kauppa. A2: CO:ssa on suuri määrä varusteita, joista jokainen oli sisällytettävä yhteen taidoista. Varusteet jaettiin taitoihin järjestyksessä hyödyllisyytensä mukaisesti. Näin pelaajat saavat pelimuodossa edetessään käyttöönsä edellisiä hyödyllisempiä varusteita. Varusteille luotiin oma luokka alustustiedostoon nimeltään **Products**, jossa jokaiselle varusteelle määriteltiin oma luokka. Luokkien niminä käytettiin varusteiden alkuperäisiä luokkanimiä, jotta luokkatietoihin viittaaminen olisi helppoa. Varusteluokassa määritellään mm. varusteen kategoria sekä hankkija, joita käytetään kauppanäkymässä varusteiden järjestämiseen.

Varusteiden hinnat oli määriteltävä siten, että varusteen hyöty pelimaailmassa olisi suhteutettuna sen hintaan. Hintojen asettaminen manuaalisesti sekalaisille varusteille, kuten kartalle tai savukranaatille, oli helppoa, sillä nämä varusteet ovat vailla vertailukohteita. Isoilla varustekategorioilla, kuten aseilla, tämä koettiin kuitenkin erittäin työlääksi, sillä jokaisella varusteella oli useita huomioon otettavia ominaisuuksia. Näitä ominaisuuksia oli verrattava toisten, saman kategorian varusteiden, ominaisuuksiin, jotta hinnat olisivat balanssissa. Aseiden, lippaiden sekä reppujen hintojen laskemiseen kehitettiin algoritmit, joilla kyseisen varusteen

hinta lasketaan sen ominaisuuksista. Pelimuodon käynnistyksessä ajetaan funktio **ANT_F_SUPPLIERS_CALCULATE_PRODUCT_BASE_PRICES** (LIITE 8), joka laskee edellä mainittujen varusteiden hinnan ja tallentaa ne dynaamisesti luotuihin muuttujiin, joista hintatietojen haku on vaivatonta. Aseen hinta lasketaan algoritmilla **(aseen käsittelynopeus * laukausten määrä sekunnissa) + (aseen tarkkuus * ammuksen lähtönopeus) * aseeseen kiinnitetyn tähtäinlaitteen suurennusarvo**. Lippaan hinta lasketaan sen sisältämistä ammuksista algoritmilla **(ammuksen räjähdysalueen koko + ammuksen tekemä vahinko) * ammusten määrä lippaassa**. Repun hinta lasketaan algoritmilla **reppuun mahtuvien aseiden määrä * reppuun mahtuvien lippaiden määrä**.

Koska varusteet ovat pelaajan eivätkä hahmon, oli niille luotava säilytyspaikka, jossa pelaajan ostamat varusteet pysyvät, vaikka hän vaihtaakin hahmoa kesken peli-istunnon. Kaupan maamerkin, eli kontin, viereen luotiin varustelaatikko, joka on asiakaskoneille lokaali. Tämä tarkoittaa, että sillä on eri esiintymä jokaisella asiakaskoneella. Tämän seurauksena pelaajan varustelaatikkoon asettamat varusteet näkyvät siis ainoastaan kyseiselle pelaajalle.

Varusteiden ostoa sekä myyntiä hallitaan funktiolla **ANT_F_SUPPLIERS_BUY_SELL_PRODUCT** (LIITE 9), joka ajetaan pelaajan koneella aina, kun pelaaja ostaa tai myy varusteen. Funktiolle syötetään varusteen hinta, tyyppi sekä luokkanimi. Hinta esitetään negatiivisenä lukuna, jos kyseessä on ostotapahtuma. Ostotapauksessa funktio tarkistaa, onko pelaajalla tarpeeksi rahaa varusteen ostamiseen. Tämän jälkeen osto- tai myyntitapahtuma suoritetaan ja pelaajan rahamäärään tehdään sen aiheuttama muutos. Lopuksi pelaajan varustelaatikkoon lisätään tai sieltä poistetaan kyseinen varuste.

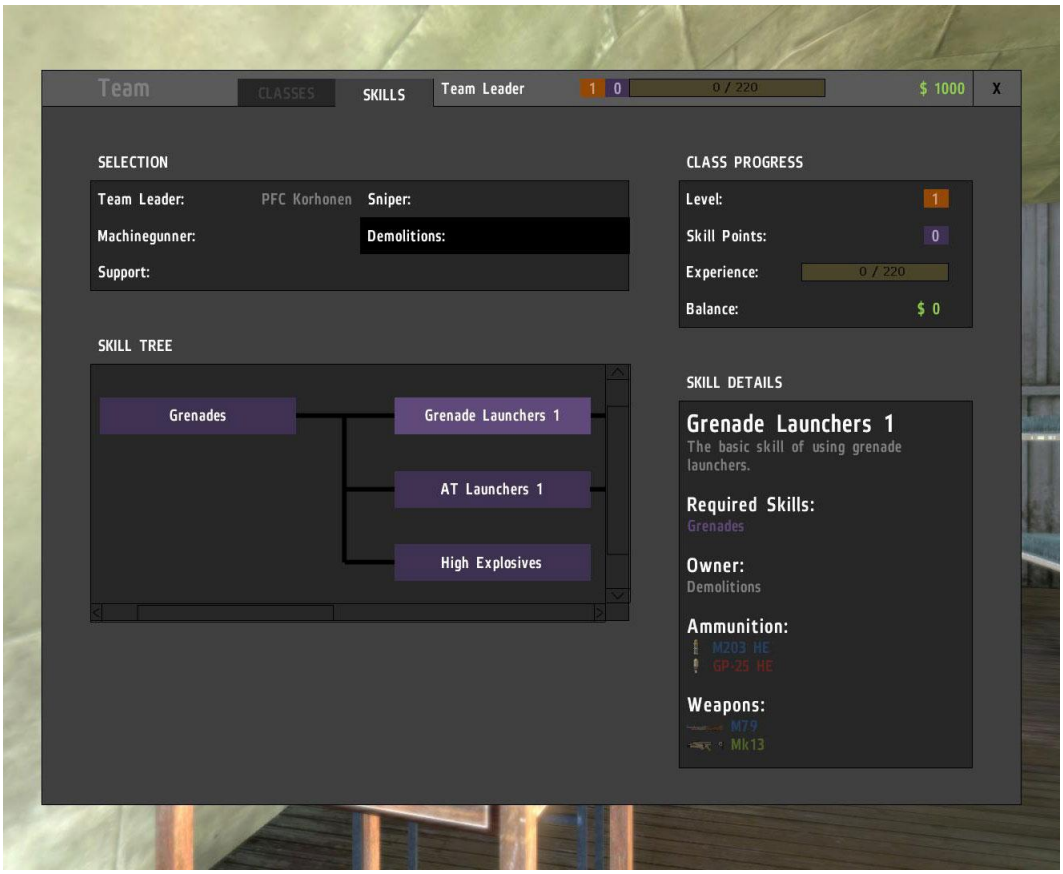
5.2.2 Estetiikka

Taidot-välilehti. Välilehdessä käytettiin hyväksi hahmot-välilehteen luotua hahmovalinta-elementtiä, jolla hahmoja voidaan valita sekä aktivoida. Tämä elementti sijoitettiin samaan kohtaan kuin edellisellä välilehdellä. Sen toiminta säilytettiin muutoin samana, mutta hahmoja valitessa päivittyvät taidot-välilehden

omat elementit esittämään hahmon ominaisuuksia. Taidot-välilehdelle luotiin kolme uutta elementtiä, jotka ovat taitopuu, taidon yksityiskohdat sekä kehitys.

Taitopuita oli luotava viisi: yksi jokaiselle hahmolle. Jokainen viidestä taitopuu-elementistä luotiin päällekkäin, mutta niistä esitetään pelaajalle vain hahmovalitsimella valittua hahmoa vastaava taitopuu. Jokainen puun taito luotiin painikkeeksi, jota yhdesti napsauttamalla ajetaan funktio **ANT_F_SKILLS_UPDATE_SKILL_DETAILS** (LIITE 10), joka päivittää taidon yksityiskohdat -elementin esittämään kyseisen taidon tietoja (Kuvio 31). Kaksoisnapsauttamalla taidon painiketta ajetaan yllä mainittu funktio **ANT_F_SKILLS_SELECT_SKILL**, jolla taito aktivoidaan. Taidon yksityiskohdat -elementti luotiin tekstielementistä, joka mahdollisti jäsennellyn tekstin esittämisen. Näin taidosta saatavat varusteet voitiin esittää listassa, värikoodattuna sekä kuvien kanssa, jotta taidon sisällön ymmärtäminen olisi helpompaa. Värikoodauksessa jokainen varuste esitettiin sitä myyvän hankkijan värillä.

Kehitys-elementin tarkoituksena on antaa pelaajille mahdollisuus nähdä kaikkien hahmojen sekä pelaajien tilanne peli-istunnon aikana. Kehitys-elementissä esitetään hahmovalinta-elementistä valitun hahmon tiedot otsikkopalkista tutussa järjestyksessä: hahmon taso, taitopisteet, kokemuspisteet, sekä hahmoa ohjaavan pelaajan rahamäärä.



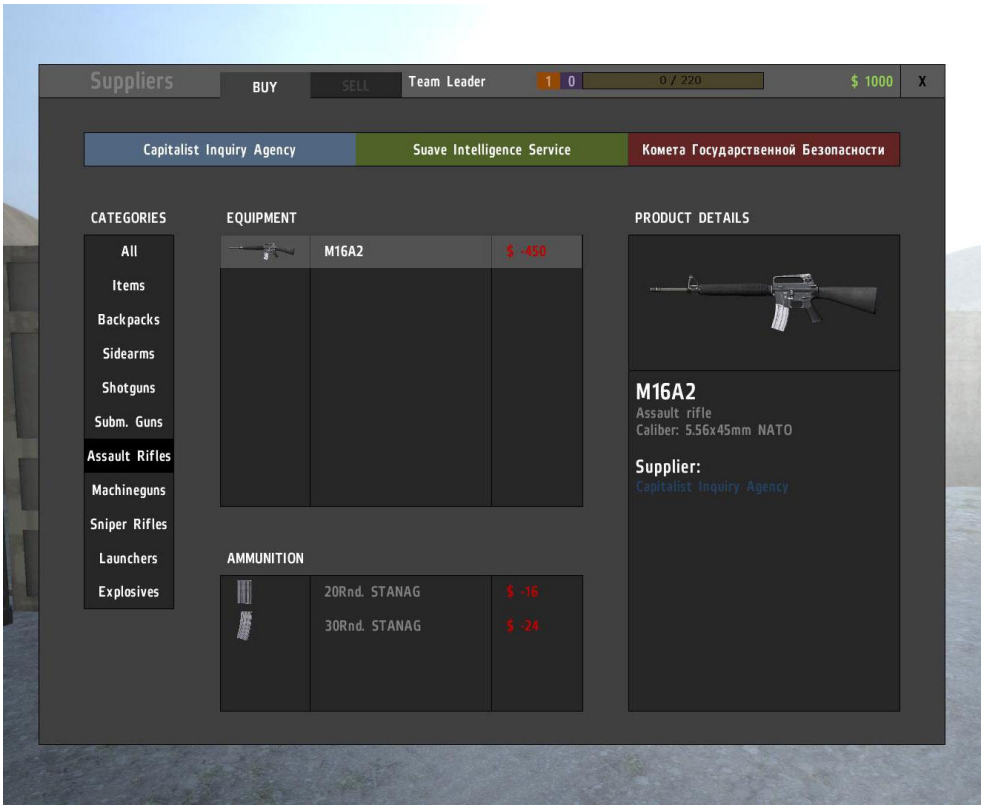
Kuvio 31. Taidot-välilehti pelissä.

Kauppanäkymä. Kauppanäkymän molemmat välilehdet, osto sekä myynti, sisältävät samat elementit. Ensimmäisenä näkymään luotiin hankkijavalitsin sekä kategoriavalitsin, joiden toiminta vastaa hahmovalinta-elementtiä. Valitsimia käytetään varustetulosten suodattamiseen hankkijan sekä kategorian perusteella. Kun jompaa kumpaa valitsinta napsautetaan yhdesti, päivitetään sen valinta valitsimelle varattuun muuttujaan ja ajetaan skripti **ANT_S_SUPPLIERS_UPDATE_PRODUCT_LIST** (LIITE 11), joka päivittää varuste- sekä ammuslistan esittämään haettuja varustetyyppejä. Skripti hakee listan varusteista **Products**-luokasta ja suodattaa ne valitun hankkijan sekä kategorian mukaan. Skripti myös tarkistaa, mitkä taidot pelaajat ovat aktivoineet ja näyttää listassa vain kyseisistä taidoista saatavat varusteet.

Varuste- sekä ammuslista luotiin valmiista monisarakkeisesta listaelementistä, johon määriteltiin kolme saraketta: varusteen kuva, nimi sekä hinta. Listoille luotiin valintaelementtejä vastaava kaksoistoiminto, jossa listan riviä yhdesti

napsauttamalla valitaan kyseinen rivi ja suoritetaan funktio **ANT_F_SUPPLIERS_UPDATE_PRODUCT_DETAILS** (LIITE 12), joka päivittää tuotteen yksityiskohdat -elementin esittämään tuotteen tietoja (Kuvio 32). Kaksoisnapsauttamalla riviä ajetaan yllä mainittu funktio **ANT_F_SUPPLIERS_BUY_SELL_PRODUCT**, joka suorittaa varusteen osto- tai myyntitoiminnon välilehdestä riippuen. Varuste- sekä ammuslistan valinnoista luotiin toisensa kumoavia, jotta pelaaja voi valita vain yhden varusteen tai ammuksen kerrallaan.

Tuotteen yksityiskohdat -elementti luotiin kahdesta osasta: kuvaelementistä sekä jäsennetystä tekstielementistä. Kuvassa esitetään valitun varusteen kuva listakuvaa isompana ja tiedoissa esitetään varustetta koskevat tiedot, kuten tuotteen nimi, kuvaus sekä hankkija.



Kuvio 32. Osto-välilehti pelissä.

5.2.3 Testaus

Toisen vaiheen testaus suoritettiin moninpeliympäristössä kolmella tietokoneella. Yksi koneista toimi sekä palvelimena että asiakaskoneena ja kaksi muuta konetta vain asiakaskoneina. Testauksen aiheina olivat taitojen valinta sekä aktivoiminen ja kauppanäkymän toiminta kokonaisuutena. Testauksessa haettiin vastauksia seuraaviin kysymyksiin:

- Toimiiko usean pelaajan tietojen tallentaminen palvelimelle?
- Onko hahmojen vaihtamisessa ongelmia usean pelaajan kanssa?
- Päivittyvätkö hahmojen sekä pelaajien tiedot toisille asiakaskoneille?
- Onnistuuko taitojen valitseminen ja aktivoiminen?
- Toimivatko kauppanäkymän välilehdet toivotulla tavalla?
- Näkyykö varustelaatikon sisältö lokaalisti?

Testi-istunnossa ei havaittu ongelmia usean pelaajan tietojen tallentamisessa palvelimelle. Myöskään hahmojen vaihtaminen pelaajien kesken ei tuottanut ongelmia. Pelaajien tiedot päivittyivät asiakaskoneille, mutta ne esitettiin vain, kun pelaaja sulki ja avasi taidot-välilehden. Ongelmana oli tietojen haku palvelimelta, joka tapahtui vain kun kyseisen asiakaskoneen pelaaja koki muutoksia oman hahmonsa tietoihin. Virhe korjattiin lisäämällä verkkotapahtumankäsittelijä, joka suoritti hahmotietojen haun palvelimelta jokaisella asiakaskoneella aina, kun yksi hahmoista koki tietojensa muutoksen. Taitojen valitsemisessa tai aktivoimisessa ei havaittu lokaaleja eikä verkkokohtaisia ongelmia. Kauppanäkymä toimi halutulla tavalla ja varustelaatikon sisältö näkyi vain sen omistamalle pelaajalle.

6 LOPPUSANAT

Työn tarkoituksena oli luoda moninpelimuoto ARMA 2: Combined Operations -tietokonepeliin. Pelimuotoa ei työn loppuvaiheessa saatu vielä julkaisuasteelle, mutta sen suunnittelu ja tähänastinen toteutus on erinomainen pohja jatkokehitykselle. Kehitystä tullaan jatkamaan kunnes pelimuoto saadaan koottua julkiseksi testiversioksi. Tämän jälkeen se tullaan julkaisemaan pelin virallisella keskustelulaudalla. Julkisesta testiversiosta saadun palautteen perusteella päätetään, jatketaanko kehitystä edelleen.

Työstä opittiin monia asioita, joista tärkeimpinä esille nousivat suunnittelun tärkeys pelinkehityksessä sekä käyttöliittymien looginen esitystapa käyttäjälle. Suunnittelussa eritoten muuhun alan sisältöön perehtyminen ja sen hyödyntäminen todettiin tehokkaaksi. Käyttöliittymissä standardien luomisen ja niistä kiinni pitämisen havaittiin selkeyttävän lopputuloksen lisäksi myös työprosessia.

LÄHTEET

Alien Swarm on Steam. 2011. [Verkkosivu]. Valve Corporation. [viitattu 2.11.2011]. Saatavana: http://store.steampowered.com/app/630/

Arma 2: Combined Operations Features. 2011. [Verkkosivu]. Bohemia Interactive. [viitattu 4.11.2011]. Saatavana: http://www.arma2.com/game-features/arma-2-co-game-features_en.html

ArmA 2 OA Takistan Map. 2011. [Kuva]. Bohemia Interactive. [14.11.2011]. Saatavana: http://arma3.deviantart.com/gallery/#/d49gxjg

Flick, J. 20.7.2010. ARMA 2 & ARMA 2: operation Arrowhead. [Verkkosivu]. Game Chronicles. [viitattu 13.10.2011]. Saatavana: http://www.gamechronicles.com/reviews/pc/arma2/co.htm

Evolution of the Real Virtuality Engine. 2008. [Verkkojulkaisu]. Bohemia Interactive. [25.10.2011]. Saatavana: http://www.arma2.com/index.php?option=com_rokdownloads&view=file&task=download&id=20%3Aevolution-of-real-virtuality-engine-pdf&lang=en

Schultz, C & Bryant, R. 2011. Game Testing: All In One. 2. painos. USA: Mercury Learning And Information.

Johnson, J. 2008. GUI Bloopers 2.0: Common User Interface Design Dont's and Dos. USA: Morgan Kaufmann Publishers.

Interesting Facts And Figures. 2008. [Verkkojulkaisu]. Bohemia Interactive. [25.10.1022]. Saatavana: http://www.arma2.com/index.php?option=com_rokdownloads&view=file&task=download&id=15%3Aarma-2-facts-and-figures-pdf&lang=en

Rabin, S. ym. 2005. Introduction to Game Development. USA: Charles River Media, Inc.

Killing Floor: Overview. 2010. [Verkkosivu]. Tripwire Interactive, LLC. [viitattu 2.11.2011]. Saatavana: http://www.killingfloorthegame.com/overview/

Left 4 Dead: Overview. 2009. [Verkkosivu]. Valve Corporation. [viitattu 2.11.2011]. Saatavana: http://www.l4d.com/l4d/game.htm

Ho, D. 2011. Notepad++ Home. [Verkkosivu]. [viitattu 1.11.2011]. Saatavana: http://notepad-plus-plus.org/

SQF syntax. 24.7.2011. [Verkkosivu]. [viitattu 18.10.2011]. Saatavana: http://community.bistudio.com/wiki/SQF_syntax

SQS to SQF conversion. 24.7.2011. [Verkkosivu]. [viitattu 24.10.2011]. Saatavana: http://community.bistudio.com/wiki/SQS_to_SQF_conversion

Takistan. 2011. [Verkkosivu]. Bohemia Interactive. [viitattu 28.10.2011]. Saatavana: http://www.arma2.com/arma-2-oa-takistan/index_en.html

Schell, J. 2008. The Art of Game Design: A book of lenses. USA: Morgan Kaufmann Publishers.

Dille, F & Platten, J. Z. 2007. The Ultimate Guide to Video Game Writing and Design. USA: Lone Eagle Publishing Company.

Spolsky, J. 2001. User Interface Design for Programmers. USA: Apress.

Warfare 2 Manual. 19.11.2010. [Verkkosivu]. [viitattu 26.10.2011]. Saatavana: http://community.bistudio.com/wiki/Warfare_2_Manual

LIITTEET

LIITE 1 Funktion ANT_F_CLASSES_SELECT_CLASS sisältö

```
// attempt to select a new class
ANT_F_CLASSES_SELECT_CLASS =
{
  private
  [
    "_newClassID",

    "_newClassName",
    "_playerUID", "_oldClassID", "_oldClassName",

    "_newPlayerUnitRank",

    "_outfitIDs", "_outfitClassName",
    "_oldPlayerUnit", "_oldPlayerUnitDirection", "_oldPlayerUnitPosition",
    "_newPlayerUnit"
  ];

  // import variables
  _newClassID = _this;

  // get new class name
  _newClassName = configName ((missionConfigFile >> "cfgAssignment" >> "Classes") select
_newClassID);

  // get player UID
  _playerUID = player call ANT_F_GET_PLAYER_UID;

  // get old class ID
  _oldClassID = ANT_LOGIC_PLAYER_DATA getVariable (_playerUID + "_CLASSID");

  // old class name
  _oldClassName = "";

  // player had a class
  if (_oldClassID >= 0) then
  {
    // get old class name
    _oldClassName = configName ((missionConfigFile >> "cfgAssignment" >> "Classes") select
_oldClassID);
  };

  // selected class is the same as the old class
  if (_newClassID == _oldClassID) then
  {
    // set player classID to no class
    ANT_LOGIC_PLAYER_DATA setVariable [_playerUID + "_CLASSID", -1, true];

    // player had a class
    if (_oldClassID >= 0) then
    {
      // set old class availability to true
      ANT_LOGIC_CLASS_DATA setVariable [_oldClassName + "_AVAILABLE", true, true];
    };

    // send public notice that class has been changed
    ANT_BOOL_CLASS_SELECTED = true;
    publicVariable "ANT_BOOL_CLASS_SELECTED";

    // update class selection
    [] spawn ANT_S_CLASSES_UPDATE_CLASS_SELECTION;
  }
  else // selected class is different from the old class
  {
    // class available
    if (ANT_LOGIC_CLASS_DATA getVariable (_newClassName + "_AVAILABLE")) then
    {
      // set player classID to selected class
      ANT_LOGIC_PLAYER_DATA setVariable [_playerUID + "_CLASSID", _newClassID, true];

      // set class availability to false
      ANT_LOGIC_CLASS_DATA setVariable [_newClassName + "_AVAILABLE", false, true];

      // player had a class
```

```
      if (_oldClassID >= 0) then
      {
        // set old class availability to true
        ANT_LOGIC_CLASS_DATA setVariable [_oldClassName + "_AVAILABLE", true, true];
      };

      // set new unit rank initially to private
      _newPlayerUnitRank = "PRIVATE";

      // select new unit rank according to the new class
      switch (_newClassID) do
      {
        case 0:
        {
          _newPlayerUnitRank = "COLONEL";
        };

        case 1:
        {
          _newPlayerUnitRank = "MAJOR";
        };

        case 2:
        {
          _newPlayerUnitRank = "CAPTAIN";
        };

        case 3:
        {
          _newPlayerUnitRank = "LIEUTENANT";
        };

        case 4:
        {
          _newPlayerUnitRank = "SERGEANT";
        };

        default
        {

        };
      };

      // get class outfit ID
      _outfitIDs = ANT_LOGIC_CLASS_DATA getVariable (_newClassName + "_OUTFIT");

      // get outfit class name
      _outfitClassName = (getArray (((missionConfigFile >> "cfgAssignment" >> "Wardrobes")
select (_outfitIDs select 0)) select (_outfitIDs select 1))) select (_outfitIDs select 2);

      // get old player unit
      _oldPlayerUnit = player;
      _oldPlayerUnitDirection = direction _oldPlayerUnit;
      _oldPlayerUnitPosition = position _oldPlayerUnit;

      // create new player unit
      _newPlayerUnit = (createGroup RESISTANCE) createUnit [_outfitClassName,
_oldPlayerUnitPosition, [], 0, "NONE"];
      _newPlayerUnit setUnitRank _newPlayerUnitRank;

      // select new player unit
      selectPlayer _newPlayerUnit;

      // change player unit variable
      ANT_LOGIC_PLAYER_DATA setVariable [_playerUID + "_UNIT", _newPlayerUnit, true];

      // transfer loadout
      [_oldPlayerUnit, _newPlayerUnit] call ANT_F_TRANSFER_LOADOUT;

      // get rid of old player unit
      [_oldPlayerUnit] joinSilent grpNull;
      _oldPlayerUnit setIdentity ("ANT_" + _newClassName);
      _oldPlayerUnit setDamage 1;
      _oldPlayerUnit setPosASL [0, 0, 0];
      deleteVehicle _oldPlayerUnit;

      // place new player to where the old player was
```

```
        _newPlayerUnit setDir _oldPlayerUnitDirection;
        _newPlayerUnit setPos _oldPlayerUnitPosition;

        // send public notice that class has been changed
        ANT_BOOL_CLASS_SELECTED = true;
        publicVariable "ANT_BOOL_CLASS_SELECTED";

        // update class selection
        [] spawn ANT_S_CLASSES_UPDATE_CLASS_SELECTION;
      };
    };
};
```

LIITE 2 Funktion ANT_F_CLASSES_ADD_EXPERIENCE sisältö

```
// add experience to a class
ANT_F_CLASSES_ADD_EXPERIENCE =
{
  private
  [
    "_classID", "_experienceToAdd",

    "_className",
    "_oldLevel", "_oldSkillpoints", "_oldExperience",

    "_newExperience",

    "_newLevel", "_checkExperience",
    "_goalExperience",

    "_newSkillpoints"
  ];

  // import variables
  _classID = _this select 0;
  _experienceToAdd = _this select 1;

  // get class name
  _className = configName ((missionConfigFile >> "cfgAssignment" >> "Classes") select
_classID);

  // get old level, skill points and experience
  _oldLevel = ANT_LOGIC_CLASS_DATA getVariable (_className + "_LEVEL");
  _oldSkillpoints = ANT_LOGIC_CLASS_DATA getVariable (_className + "_SKILLPOINTS");
  _oldExperience = ANT_LOGIC_CLASS_DATA getVariable (_className + "_EXPERIENCE");

  // calculate new experience
  _newExperience = _oldExperience + _experienceToAdd;

  // calculate new level
  _newLevel = 0;
  _checkExperience = 0;

  for [{_newLevel = _oldLevel}, {_checkExperience >= 0}, {_newLevel = _newLevel + 1}] do
  {
    // get goal experience
    _goalExperience = ([_classID, _newLevel] call ANT_F_CLASSES_GET_GOAL_EXPERIENCE);

    // substract goal
    _checkExperience = _newExperience - _goalExperience;
  };

  _newLevel = _newLevel - 1;

  // calculate new skill points
  _newSkillpoints = _oldSkillpoints + (_newLevel - _oldLevel);

  // set new level, skill points and experience
  ANT_LOGIC_CLASS_DATA setVariable [_className + "_LEVEL", _newLevel, true];
  ANT_LOGIC_CLASS_DATA setVariable [_className + "_SKILLPOINTS", _newSkillpoints, true];
  ANT_LOGIC_CLASS_DATA setVariable [_className + "_EXPERIENCE", _newExperience, true];

  // send public notice that experience has been gained
  ANT_NUMBER_EXPERIENCE_GAINED = _classID;
  publicVariable "ANT_NUMBER_EXPERIENCE_GAINED";

  // if server is a client
  if (!isDedicated) then
  {
    // update stats
    [] spawn ANT_S_UPDATE_STATS;
  };
};
```

LIITE 3 Funktion ANT_F_CLASSES_GET_GOAL_EXPERIENCE sisältö

```
// return an experience goal for a certain level
ANT_F_CLASSES_GET_GOAL_EXPERIENCE =
{
  private
  [
    "_classID", "_level",
    "_levelModifier",

    "_goalExperience"
  ];

  // import variables
  _classID = _this select 0;
  _level = _this select 1;

  // get level modifier
  _levelModifier = getNumber (((missionConfigFile >> "cfgAssignment" >> "Classes") select
_classID) >> "levelModifier");

  // calculate goal experience
  _goalExperience = floor (((2 * _levelModifier) ^ _level) * 100);

  // return goal experience
  _goalExperience;
};
```

LIITE 4 Skriptin ANT_S_ASSIGNMENTS_INIT_BASE sisältö

```
// initialize base
ANT_S_ASSIGNMENTS_INIT_BASE =
{
  private
  [
    "_baseTemplate",
    "_basePosition", "_baseArmoryBox", "_baseTriggers", "_baseObjects",

    "_baseObjectsArray",

    "_playerUnits",

    "_playerUID",
    "_playerArmoryBox"
  ];

  // get base template data
  _baseTemplate = ANT_LOGIC_ASSIGNMENTS_DATA getVariable "BASE_TEMPLATE";
  _basePosition = _baseTemplate select 0;
  _baseArmoryBox = _baseTemplate select 1;
  _baseTriggers = _baseTemplate select 2;
  _baseObjects = _baseTemplate select 3;

  // server
  if (isServer) then
  {
    // set on assignment variable to false
    ANT_LOGIC_ASSIGNMENTS_DATA setVariable ["ON_ASSIGNMENT", false, true];

    // clear assignment markers
    {
      // delete marker
      deleteMarker _x;
    }
    forEach ANT_ARRAY_ASSIGNMENTS_ASSIGNMENT_MARKERS;

    // create base objects array
    _baseObjectsArray = [];

    // create objects
    {
      private
      [
        "_objectData", "_objectClassName", "_objectPosition", "_objectDirection",
        "_object"
      ];

      // get object data
      _objectData = _x;
      _objectClassName = _objectData select 0;
      _objectPosition = _objectData select 1;
      _objectDirection = _objectData select 2;

      // create object
      _object = createVehicle [_objectClassName, _objectPosition, [], 0, "CAN_COLLIDE"];
      _object setDir _objectDirection;
      _object setPos _objectPosition;

      // make all objects indestructible
      _object allowDamage false;

      // add object to base objects array
      _baseObjectsArray = _baseObjectsArray + [_object];
    }
    forEach _baseObjects;

    // update base objects
    ANT_LOGIC_ASSIGNMENTS_DATA setVariable ["BASE_OBJECTS", _baseObjectsArray, true];

    // get player units
    _playerUnits = [] call ANT_F_GET_PLAYER_UNITS;

    // for all player units
```

```
    {
      // move player units
      _x setPos _basePosition;

      // make units invulnerable
      _x allowDamage false;
    }
    forEach _playerUnits;

    // update assignments
    [] call ANT_F_ASSIGNMENTS_UPDATE_ASSIGNMENTS;
  };

  // client
  if (!isDedicated) then
  {
    // get player UID
    _playerUID = player call ANT_F_GET_PLAYER_UID;

    // get player armory box
    _playerArmoryBox = ANT_LOGIC_PLAYER_DATA getVariable (_playerUID + "_ARMORY_BOX");

    // transport armory box to base
    _playerArmoryBox setPos (_baseArmoryBox select 1);

    // clear assignment triggers
    {
      // delete trigger
      deleteVehicle _x;
    }
    forEach ANT_ARRAY_ASSIGNMENTS_ASSIGNMENT_TRIGGERS;

    // clear assignment triggers array
    ANT_ARRAY_ASSIGNMENTS_ASSIGNMENT_TRIGGERS = [];

    // update base triggers array
    ANT_ARRAY_ASSIGNMENTS_BASE_TRIGGERS = _baseTriggers call
ANT_F_ASSIGNMENTS_CREATE_TRIGGERS;

    // wait until all base objects have been created
    waitUntil {count (ANT_LOGIC_ASSIGNMENTS_DATA getVariable "BASE_OBJECTS") > 0};

    // get base objects array
    _baseObjectsArray = ANT_LOGIC_ASSIGNMENTS_DATA getVariable "BASE_OBJECTS";

    // make all objects indestructible
    {
      _x allowDamage false;
    }
    forEach _baseObjectsArray;

    // get player units
    _playerUnits = [] call ANT_F_GET_PLAYER_UNITS;

    // for all player units
    {
      // make units invulnerable
      _x allowDamage false;
    }
    forEach _playerUnits;

    // open dialog on classes page
    ANT_ARRAY_DIALOG_PAGE_CLASSES call ANT_F_DIALOG_OPEN_PAGE;
  };
};
```

LIITE 5 Skriptin ANT_S_CLASSES_UPDATE_CLASS_SELECTION sisältö

```
// update class selection
ANT_S_CLASSES_UPDATE_CLASS_SELECTION =
{
  private
  [
    "_baseDialog",

    "_classesRecognitionControl", "_skillsRecognitionControl",

    "_teamLeaderPlayerText", "_teamLeaderPlayerText2", "_machinegunnerPlayerText",
"_machinegunnerPlayerText2", "_supportPlayerText", "_supportPlayerText2",
"_sniperPlayerText", "_sniperPlayerText2", "_demolitionsPlayerText",
"_demolitionsPlayerText2",

    "_allPlayerUIDs",
    "_allPlayerUIDsCount",
    "_i",
    "_playerUID", "_playerName", "_playerClassID", "_playerInGame"
  ];

  // prepare for GUI operations
  disableSerialization;

  // get base dialog
  _baseDialog = (findDisplay 427001);

  // base dialog is open
  if (!(isNull _baseDialog)) then
  {
    // get 'classes' page's recognition control
    _classesRecognitionControl = _baseDialog displayCtrl 427220;

    // get 'skills' page's recognition control
    _skillsRecognitionControl = _baseDialog displayCtrl 427301;

    // 'classes' or 'skills' page is open
    if (ctrlShown _classesRecognitionControl || ctrlShown _skillsRecognitionControl) then
    {
      // get controls
      _classSelector = _baseDialog displayCtrl 427203;
      _teamLeaderPlayerText = _baseDialog displayCtrl 427205;
      _teamLeaderPlayerText2 = _baseDialog displayCtrl 427206;
      _machinegunnerPlayerText = _baseDialog displayCtrl 427208;
      _machinegunnerPlayerText2 = _baseDialog displayCtrl 427209;
      _supportPlayerText = _baseDialog displayCtrl 427211;
      _supportPlayerText2 = _baseDialog displayCtrl 427212;
      _sniperPlayerText = _baseDialog displayCtrl 427214;
      _sniperPlayerText2 = _baseDialog displayCtrl 427215;
      _demolitionsPlayerText = _baseDialog displayCtrl 427217;
      _demolitionsPlayerText2 = _baseDialog displayCtrl 427218;

      // update class selector
      [_classSelector, _teamLeaderPlayerText, _machinegunnerPlayerText,
_supportPlayerText, _sniperPlayerText, _demolitionsPlayerText] call
ANT_F_CLASSES_UPDATE_CLASS_SELECTOR;

      // empty player name texts
      _teamLeaderPlayerText2 ctrlSetText "";
      _machinegunnerPlayerText2 ctrlSetText "";
      _supportPlayerText2 ctrlSetText "";
      _sniperPlayerText2 ctrlSetText "";
      _demolitionsPlayerText2 ctrlSetText "";

      // get array of player UIDs
      _allPlayerUIDs = ANT_ARRAY_PLAYER_UIDS;

      // loop through the players
      _allPlayerUIDsCount = count _allPlayerUIDs;

      for [{_i = 0}, {_i < _allPlayerUIDsCount}, {_i = _i + 1}] do
      {
        // get player info
        _playerUID = _allPlayerUIDs select _i;
```

```
        _playerName = ANT_LOGIC_PLAYER_DATA getVariable (_playerUID + "_NAME");
        _playerClassID = ANT_LOGIC_PLAYER_DATA getVariable (_playerUID + "_CLASSID");
        _playerInGame = ANT_LOGIC_PLAYER_DATA getVariable (_playerUID + "_INGAME");

        // player in game
        if (_playerInGame) then
        {
          // display player names on right classes
          switch (_playerClassID) do
          {
            case 0:
            {
              _teamLeaderPlayerText2 ctrlSetText _playerName;
            };

            case 1:
            {
              _machinegunnerPlayerText2 ctrlSetText _playerName;
            };

            case 2:
            {
              _supportPlayerText2 ctrlSetText _playerName;
            };

            case 3:
            {
              _sniperPlayerText2 ctrlSetText _playerName;
            };

            case 4:
            {
              _demolitionsPlayerText2 ctrlSetText _playerName;
            };

            default
            {

            };
          };
        };
      };

      // 'classes' page is open
      if (ctrlShown _classesRecognitionControl) then
      {
        // update class description
        [] call ANT_F_CLASSES_UPDATE_CLASS_DESCRIPTION;

        // update class outfit
        [] call ANT_F_CLASSES_UPDATE_CLASS_OUTFIT;
      }
      else
      {
        // 'skills' page is open
        if (ctrlShown _skillsRecognitionControl) then
        {
          // update skill details
          [] call ANT_F_CLASSES_UPDATE_SKILL_DETAILS;

          // update skill tree
          [] call ANT_F_CLASSES_UPDATE_SKILL_TREE;
        };
      };
    };
  };

  // update stats
  [] spawn ANT_S_UPDATE_STATS;
};
```

LIITE 6 Skriptin ANT_S_CLASSES_OPEN_CLASS_OUTFIT sisältö

```
// open class outfit
ANT_S_CLASSES_OPEN_CLASS_OUTFIT =
{
  private
  [
    "_baseDialog", "_loadBack", "_loadText",

    "_distance", "_deviationX", "_deviationY",
    "_finalScreenHeight", "_finalScreenXPos", "_finalScreenYPos",
    "_loopTime",

    "_playerUID",

    "_posData", "_posObject", "_posBottom", "_posTop", "_posMiddle",
    "_posCamera", "_posCameraTarget",

    "_cam",

    "_dir"
  ];

  // prepare for GUI operations
  disableSerialization;

  // get base dialog
  _baseDialog = (findDisplay 427001);

  // loading background control
  _loadBack = _baseDialog displayCtrl 427100;

  // loading text control
  _loadText = _baseDialog displayCtrl 427226;

  // if outfit camera has not been created
  if (isNull ANT_OBJECT_DIALOG_CLASS_OUTFIT_CAMERA) then
  {
    // fade to black
    0 cutText ["", "BLACK OUT", 0.0001];

    // show loading background and text
    _loadBack ctrlShow true;
    _loadBack ctrlCommit 0;
    _loadText ctrlShow true;
    _loadText ctrlCommit 0;

    // camera starting values
    _distance = 7;
    _deviationX = 0;
    _deviationY = 0;

    // unit final values
    _finalScreenHeight = 0.6;
    _finalScreenXPos = 0.8;
    _finalScreenYPos = 0.5;

    // timeout used for loops
    _loopTime = 0.001;

    // get player UID
    _playerUID = player call ANT_F_GET_PLAYER_UID;

    // outfit position data
    _posData = ANT_LOGIC_PLAYER_DATA getVariable (_playerUID + "_OUTFIT_POSITION_DATA");
    _posObject = _posData select 0;
    _posBottom = _posData select 1;

    _posTop = [_posBottom select 0, _posBottom select 1, (_posBottom select 2) + 2];
    _posMiddle = [_posBottom select 0, _posBottom select 1, (_posBottom select 2) + 1];

    // camera position
    _posCamera = _posObject modelToWorld [(_posMiddle select 0) + _deviationX, (_posMiddle
select 1) + _distance, (_posMiddle select 2) + _deviationY];
```

```
    // camera target position
    _posCameraTarget = _posObject modelToWorld [(_posMiddle select 0) + _deviationX,
(_posMiddle select 1), (_posMiddle select 2) + _deviationY];

    // create camera
    _cam = "camera" camCreate [0,0,0];
    _cam camSetPos _posCamera;
    _cam camSetTarget _posCameraTarget;
    _cam cameraEffect ["INTERNAL", "BACK"];
    _cam camCommit 0;

    // disable cinema border
    showCinemaBorder false;

    // update camera variable
    ANT_OBJECT_DIALOG_CLASS_OUTFIT_CAMERA = _cam;

    // wait until position is on the screen
    while {count (worldToScreen _posBottom) == 0} do
    {
      sleep 0.1;
    };

    // adjust camera distance - go closer
    while {((((worldToScreen (_posObject modelToWorld _posBottom)) select 1) -
((worldToScreen (_posObject modelToWorld _posTop)) select 1)) < _finalScreenHeight} do
    {
      _distance = _distance - 0.5;
      _cam camSetPos (_posObject modelToWorld [(_posMiddle select 0) + _deviationX,
(_posMiddle select 1) + _distance, (_posMiddle select 2) + _deviationY]);
      _cam camCommit 0;

      sleep _loopTime;
    };

    // adjust camera distance - go farther
    while {((((worldToScreen (_posObject modelToWorld _posBottom)) select 1) -
((worldToScreen (_posObject modelToWorld _posTop)) select 1)) > _finalScreenHeight} do
    {
      _distance = _distance + 0.01;
      _cam camSetPos (_posObject modelToWorld [(_posMiddle select 0) + _deviationX,
(_posMiddle select 1) + _distance, (_posMiddle select 2) + _deviationY]);
      _cam camCommit 0;

      sleep _loopTime;
    };

    // adjust camera's horizontal position - go left
    while {(((worldToScreen (_posObject modelToWorld _posMiddle)) select 0) <
_finalScreenXPos} do
    {
      _deviationX = _deviationX + 0.5;
      _cam camSetPos (_posObject modelToWorld [(_posMiddle select 0) + _deviationX,
(_posMiddle select 1) + _distance, (_posMiddle select 2) + _deviationY]);
      _cam camSetTarget (_posObject modelToWorld [(_posMiddle select 0) + _deviationX,
(_posMiddle select 1), (_posMiddle select 2) + _deviationY]);
      _cam camCommit 0;

      sleep _loopTime;
    };

    // adjust camera's horizontal position - go right
    while {(((worldToScreen (_posObject modelToWorld _posMiddle)) select 0) >
_finalScreenXPos} do
    {
      _deviationX = _deviationX - 0.01;
      _cam camSetPos (_posObject modelToWorld [(_posMiddle select 0) + _deviationX,
(_posMiddle select 1) + _distance, (_posMiddle select 2) + _deviationY]);
      _cam camSetTarget (_posObject modelToWorld [(_posMiddle select 0) + _deviationX,
(_posMiddle select 1), (_posMiddle select 2) + _deviationY]);
      _cam camCommit 0;

      sleep _loopTime;
    };

    // adjust camera's vertical position - go up
    while {(((worldToScreen (_posObject modelToWorld _posMiddle)) select 1) <
```

```
_finalScreenYPos} do
    {
      _deviationY = _deviationY + 0.5;
      _cam camSetPos (_posObject modelToWorld [(_posMiddle select 0) + _deviationX,
(_posMiddle select 1) + _distance, (_posMiddle select 2) + _deviationY]);
      _cam camSetTarget (_posObject modelToWorld [(_posMiddle select 0) + _deviationX,
(_posMiddle select 1), (_posMiddle select 2) + _deviationY]);
      _cam camCommit 0;

      sleep _loopTime;
    };

    // adjust camera's vertical position - go down
    while {((worldToScreen (_posObject modelToWorld _posMiddle)) select 1) >
_finalScreenYPos} do
    {
      _deviationY = _deviationY - 0.01;
      _cam camSetPos (_posObject modelToWorld [(_posMiddle select 0) + _deviationX,
(_posMiddle select 1) + _distance, (_posMiddle select 2) + _deviationY]);
      _cam camSetTarget (_posObject modelToWorld [(_posMiddle select 0) + _deviationX,
(_posMiddle select 1), (_posMiddle select 2) + _deviationY]);
      _cam camCommit 0;

      sleep _loopTime;
    };

    // update outfit unit
    [] call ANT_F_CLASSES_UPDATE_CLASS_OUTFIT_UNIT;

    // unit initial direction
    _dir = 0;

    // start rotating unit
    [_dir, (_posObject modelToWorld _posBottom)] spawn ANT_S_CLASSES_CLASS_OUTFIT_ROTATE;

    // fade from black
    0 cutText ["", "BLACK IN", 0.5];
  };

  // hide loading background and text
  _loadBack ctrlShow false;
  _loadText ctrlShow false;

  _loadBack ctrlCommit 0;
  _loadText ctrlCommit 0;
};
```

LIITE 7 Skriptin ANT_F_SKILLS_SELECT_SKILL sisältö

```
// attempt to select a new skill
ANT_F_SKILLS_SELECT_SKILL =
{
  private
  [
    "_selectedSkillName",
    "_playerUID", "_classID",

    "_requiredSkillsArray", "_selectedSkillsArray",

    "_className", "_skillpoints"
  ];

  // import variables
  _selectedSkillName = _this;

  // get player UID
  _playerUID = player call ANT_F_GET_PLAYER_UID;

  // get class ID
  _classID = ANT_LOGIC_PLAYER_DATA getVariable (_playerUID + "_CLASSID");

  // player's own class selected
  if (_classID == ANT_NUMBER_CLASSES_SELECTION) then
  {
    // get required skills array
    _requiredSkillsArray = getArray (missionConfigFile >> "cfgAssignment" >> "Skills" >>
_selectedSkillName >> "requiredSkills");

    // get selected skills array
    _selectedSkillsArray = ANT_LOGIC_CLASS_DATA getVariable "SELECTED_SKILLS";

    // all required skills have been selected and skill has not been already selected
    if (({!(_x in _selectedSkillsArray)} count _requiredSkillsArray) <= 0 &&
!(_selectedSkillName in _selectedSkillsArray)) then
    {
      // get classname from config
      _className = configName ((missionConfigFile >> "cfgAssignment" >> "Classes") select
ANT_NUMBER_CLASSES_SELECTION);

      // get skill points
      _skillpoints = ANT_LOGIC_CLASS_DATA getVariable (_className + "_SKILLPOINTS");

      // class has skill points
      if (_skillpoints > 0) then
      {
        // substract skill points
        ANT_LOGIC_CLASS_DATA setVariable [_className + "_SKILLPOINTS", _skillpoints - 1,
true];

        // add skill to selected skills
        ANT_LOGIC_CLASS_DATA setVariable ["SELECTED_SKILLS", _selectedSkillsArray +
[_selectedSkillName], true];

        // send public notice that skill has been selected
        ANT_BOOL_SKILL_SELECTED = true;
        publicVariable "ANT_BOOL_SKILL_SELECTED";

        // update class selection
        [] spawn ANT_S_CLASSES_UPDATE_CLASS_SELECTION;
      };
    };
  };
};
```

LIITE 8 Funktion ANT_F_SUPPLIERS_CALCULATE_PRODUCT_BASE _PRICES sisältö

```
// calculate product base prices
ANT_F_SUPPLIERS_CALCULATE_PRODUCT_BASE_PRICES =
{
  private
  [
    "_productTypeArray", "_productTypeArrayCount",

    "_i",
    "_productType",
    "_productConfig", "_productConfigCount",

    "_j",
    "_productClassName", "_productPrice"
  ];

  // create product type array
  _productTypeArray = ["Items", "Ammunition", "BackPacks", "Weapons"];

  // get product type array count
  _productTypeArrayCount = count _productTypeArray;

  // loop through product type array
  for [{_i = 0}, {_i < _productTypeArrayCount}, {_i = _i + 1}] do
  {
    // get product type
    _productType = _productTypeArray select _i;

    // get product config
    _productConfig = (missionConfigFile >> "cfgAssignment" >> "Products" >> _productType);

    // count number of config entries
    _productConfigCount = count _productConfig;

    // loop through config entries
    for [{_j = 0}, {_j < _productConfigCount}, {_j = _j + 1}] do
    {
      // get product class name from config
      _productClassName = configName (_productConfig select _j);

      // get product price
      _productPrice = [_productClassName, _productType] call
ANT_F_SUPPLIERS_CALCULATE_PRODUCT_PRICE;

      // save product base price
      ANT_LOGIC_SUPPLIERS_DATA setVariable [(_productType + "_" + _productClassName +
"_BASE_PRICE"), _productPrice, true];
    };
  };
};
```

LIITE 9 Funktion ANT_F_SUPPLIERS_BUY_SELL_PRODUCT sisältö

```
// attempt to buy a product
ANT_F_SUPPLIERS_BUY_SELL_PRODUCT =
{
  private
  [
    "_productPrice", "_productType", "_productClassName",
    "_playerUID", "_playerBalance",

    "_playerArmoryBox"
  ];

  // import variables
  _productPrice = _this select 0;
  _productType = _this select 1;
  _productClassName = _this select 2;

  // get player UID
  _playerUID = player call ANT_F_GET_PLAYER_UID;

  // get player balance
  _playerBalance = ANT_LOGIC_PLAYER_DATA getVariable (_playerUID + "_BALANCE");

  // player is buying the product
  if (_productPrice < 0) then
  {
    // player has enough money
    if ((_playerBalance + _productPrice) >= 0) then
    {
      // set player balance
      ANT_LOGIC_PLAYER_DATA setVariable [_playerUID + "_BALANCE", (_playerBalance +
_productPrice), true];

      // get player armory box
      _playerArmoryBox = ANT_LOGIC_PLAYER_DATA getVariable (_playerUID + "_ARMORY_BOX");

      // add product to the armory box according to the product type
      switch (_productType) do
      {
        case "Items":
        {
          _playerArmoryBox addWeaponCargo [_productClassName, 1];
        };

        case "BackPacks":
        {
          _playerArmoryBox addBackpackCargo [_productClassName, 1];
        };

        case "Ammunition":
        {
          _playerArmoryBox addMagazineCargo [_productClassName, 1];
        };

        case "Weapons":
        {
          _playerArmoryBox addWeaponCargo [_productClassName, 1];
        };

        default
        {

        };
      };

      // send public notice that a product has been bought
      ANT_BOOL_PRODUCT_BOUGHT = true;
      publicVariable "ANT_BOOL_PRODUCT_BOUGHT";

      // update stats
      [] spawn ANT_S_UPDATE_STATS;
    };
  }
  else
```

```
  {
    // player is selling the product
    if (_productPrice > 0) then
    {
      // get player armory box
      _playerArmoryBox = ANT_LOGIC_PLAYER_DATA getVariable (_playerUID + "_ARMORY_BOX");

      _typeString = "";

      // remove product from the armory box according to the product type
      switch (_productType) do
      {
        case "Items":
        {
          _typeString = "Weapon";
        };

        case "BackPacks":
        {
          _typeString = "Backpack";
        };

        case "Ammunition":
        {
          _typeString = "Magazine";
        };

        case "Weapons":
        {
          _typeString = "Weapon";
        };

        default
        {

        };
      };

      // type is selected
      if (_typeString != "") then
      {
        // get product cargo
        _productCargo = call compile (format ["get%1Cargo _playerArmoryBox;",
_typeString]);
        _productCargoNames = _productCargo select 0;
        _productCargoAmounts = _productCargo select 1;

        // get cargo position and amount
        _productPos = _productCargoNames find _productClassName;
        _productAmount = _productCargoAmounts select _productPos;

        // only one product left
        if (_productAmount <= 1) then
        {
          _productCargoNames set [_productPos, objNull];
          _productCargoAmounts set [_productPos, objNull];

          _productCargoNames = _productCargoNames - [objNull];
          _productCargoAmounts = _productCargoAmounts - [objNull];
        }
        else // more than one product left
        {
          _productCargoAmounts set [_productPos, (_productAmount - 1)];
        };

        // clear armory box
        call compile (format ["clear%1CargoGlobal _playerArmoryBox;", _typeString]);

        // count products
        _productCargoNamesCount = count _productCargoNames;

        // loop through products
        for [{_i = 0}, {_i < _productCargoNamesCount}, {_i = _i + 1}] do
        {
          // create product cargo array
          _productArray = [(_productCargoNames select _i), (_productCargoAmounts select
_i)];
```

```
          // add product to the armory box
          call compile (format ["_playerArmoryBox add%1Cargo _productArray;",
_typeString]);
        };

        // set player balance
        ANT_LOGIC_PLAYER_DATA setVariable [_playerUID + "_BALANCE", (_playerBalance +
_productPrice), true];

        // send public notice that a product has been sold
        ANT_BOOL_PRODUCT_SOLD = true;
        publicVariable "ANT_BOOL_PRODUCT_SOLD";

        // update product list
        [] spawn ANT_S_DIALOG_UPDATE_PRODUCT_LIST;

        // update stats
        [] spawn ANT_S_UPDATE_STATS;
      };
    };
  };
};
```

LIITE 10 Funktion ANT_F_SKILLS_UPDATE_SKILL_DETAILS sisältö

```
// update skill description
ANT_F_SKILLS_UPDATE_SKILL_DETAILS =
{
  private
  [
    "_baseDialog",
    "_detailsHeader", "_detailsText",

    "_skillsConfig", "_skillName", "_skillDescription", "_skillOwnerClass",
"_requiredSkillsArray", "_unlocksClass",
    "_detailsString", "_detailsRowNumber",

    "_requiredSkill",

    "_unlocksArrays", "_unlocksArraysCount",
    "_previousType",

    "_i",
    "_unlocksArray", "_type", "_array", "_configName", "_color",
    "_arrayCount",

    "_j",
    "_productClassName",
    "_productANTConfig", "_productANTName", "_productANTSupplier",
    "_productBISConfig", "_productBISName", "_productBISPicture",
    "_productName", "_productPicture",

    "_detailsTextPosition", "_detailsTextNewHeight"
  ];

  // prepare for GUI operations
  disableSerialization;

  // get base dialog
  _baseDialog = (findDisplay 427001);

  // get controls
  _detailsHeader = _baseDialog displayCtrl 427320;
  _detailsText = _baseDialog displayCtrl 427323;

  // skill is selected
  if (ANT_STRING_SKILLS_SELECTION != "") then
  {
    // get skill details
    _skillsConfig = (missionConfigFile >> "cfgAssignment" >> "Skills" >>
ANT_STRING_SKILLS_SELECTION);
    _skillName = getText (_skillsConfig >> "name");
    _skillDescription = getText (_skillsConfig >> "description");
    _skillOwnerClass = getText (_skillsConfig >> "ownerClass");
    _requiredSkillsArray = getArray (_skillsConfig >> "requiredSkills");
    _unlocksClass = (_skillsConfig >> "unlocks");

    // create details string variables
    _detailsString = "<t size='1.5'>" + _skillName + "</t><br />";
    _detailsRowNumber = 2.5;

    _detailsString = _detailsString + "<t color='#808080'>" + _skillDescription + "</t><br
/>";
    _detailsRowNumber = _detailsRowNumber + 1 + ([1, _skillDescription] call
ANT_F_DIALOG_CALCULATE_STRUCTURED_TEXT_ROW_NUMBER);

    _detailsString = _detailsString + "<br /><t size='1.3'>Required Skills:</t>";
    _detailsRowNumber = _detailsRowNumber + 1.3;

    // there are required skills
    if ((count _requiredSkillsArray) > 0) then
    {
      // add required skills to the details string
      {
        // get required skill
        _requiredSkill = getText (missionConfigFile >> "cfgAssignment" >> "Skills" >> _x
>> "name");
```

```
        _detailsString = _detailsString + "<br /><t color='#60497b'>" + _requiredSkill +
"</t>";
        _detailsRowNumber = _detailsRowNumber + 1;
      }
      forEach _requiredSkillsArray;
    }
    else
    {
      // add 'none' to the details string
      _detailsString = _detailsString + "<br /><t color='#808080'>None</t>";
      _detailsRowNumber = _detailsRowNumber + 1;
    };

    // add owner class to the details string
    _detailsString = _detailsString + "<br /><br /><t size='1.3'>Owner:</t>";
    _detailsRowNumber = _detailsRowNumber + 2.3;

    _detailsString = _detailsString + "<br /><t color='#808080'>" + _skillOwnerClass +
"</t>";
    _detailsRowNumber = _detailsRowNumber + 1;

    // create unlocks arrays
    _unlocksArrays =
    [
      // type, array, config name
      ["Items", (getArray (_unlocksClass >> "items")), "cfgWeapons"],
      ["Ammunition", (getArray (_unlocksClass >> "ammunition")), "cfgMagazines"],
      ["Backpacks", (getArray (_unlocksClass >> "backPacks")), "cfgVehicles"],
      ["Weapons", (getArray (_unlocksClass >> "weapons")), "cfgWeapons"]
    ];

    // get unlocks arrays count
    _unlocksArraysCount = count _unlocksArrays;

    // create previous name
    _previousType = "";

    // loop through items
    for [{_i = 0}, {_i < _unlocksArraysCount}, {_i = _i + 1}] do
    {
      // get unlocks array data
      _unlocksArray = _unlocksArrays select _i;
      _type = _unlocksArray select 0;
      _array = _unlocksArray select 1;
      _configName = _unlocksArray select 2;

      // get array count
      _arrayCount = count _array;

      // name is different from previous name, name given and has items
      if (_type != _previousType && _type != "" && _arrayCount > 0) then
      {
        // show name as title
        _detailsString = _detailsString + "<br /><br /><t size='1.3'>" + _type + ":</t>";
        _detailsRowNumber = _detailsRowNumber + 2.3;

        // set previous name
        _previousType = _type;
      };

      // loop through array
      for [{_j = 0}, {_j < _arrayCount}, {_j = _j + 1}] do
      {
        // get product config data
        _productClassName = (_array select _j);

        // get product ANT config data
        _productANTConfig = (missionConfigFile >> "cfgAssignment" >> "Products" >> _type
>> _productClassName);
        _productANTName = getText (_productANTConfig >> "name");
        _productANTSupplier = getText (_productANTConfig >> "supplier");

        // get product BIS config data
        _productBISConfig = (configFile >> _configName >> _productClassName);
        _productBISName = getText (_productBISConfig >> "displayName");
        _productBISPicture = getText (_productBISConfig >> "picture");
```

```
        _productName = _productANTName;

        // ANT name is empty
        if (_productName == "") then
        {
          // use BIS name
          _productName = _productBISName;
        };

        // use BIS image
        _productPicture = _productBISPicture;

        // set default color
        _color = "#808080";

        // get product color according to supplier
        switch (_productANTSupplier) do
        {
          case "Blue":
          {
            _color = "#254061";
          };

          case "Green":
          {
            _color = "#4f6228";
          };

          case "Red":
          {
            _color = "#632523";
          };

          default
          {

          };
        };

        // add item to details string
        _detailsString = _detailsString + "<br /><img size='1' image='" + _productPicture
+ "'/> <t color='" + _color + "'>" + _productName + "</t>";
        _detailsRowNumber = _detailsRowNumber + 1;
      };
    };

    // set details control height
    _detailsTextPosition = ctrlPosition _detailsText;
    _detailsTextNewHeight = ((ctrlPosition _detailsHeader) select 3) * (_detailsRowNumber
/ 2);
    _detailsText ctrlSetPosition [_detailsTextPosition select 0, _detailsTextPosition
select 1, _detailsTextPosition select 2, _detailsTextNewHeight];
    _detailsText ctrlCommit 0;

    // set details control text
    _detailsText ctrlSetStructuredText (parseText _detailsString);
  }
  else // skill has not been selected
  {
    // clear details text
    _detailsText ctrlSetStructuredText (parseText "");
  };
};
```

LIITE 11 Skriptin ANT_S_SUPPLIERS_UPDATE_PRODUCT_LIST sisältö

```
// update product list
ANT_S_SUPPLIERS_UPDATE_PRODUCT_LIST =
{
  private
  [
    "_baseDialog",

    "_supplierString",

    "_productTypeString",

    "_valueColor",

    "_selectedSkills", "_selectedSkillsCount",
    "_productsArray", "_ammunitionArray",

    "_i",
    "_skillClassName",
    "_skillsConfig", "_skillName", "_skillDescription", "_skillOwnerClass",
"_unlocksClass",
    "_unlocksArrays", "_unlocksArraysCount",

    "_j",
    "_unlocksArray", "_type", "_array", "_configName", "_arrayCount",

    "_k",
    "_productClassName",
    "_productANTConfig", "_productANTSupplier", "_productANTType", "_productANTName",
"_productANTDescription",
    "_productBISConfig", "_productBISName", "_productBISDescription",
"_productBISPicture",
    "_productName", "_productDescription", "_productPicture", "_productBasePrice",
"_productPrice",
    "_productDataArray",

    "_playerUID",
    "_playerArmoryBox",
    "_armoryItemNames", "_armoryItemAmounts",
    "_itemConfig", "_itemConfigCount", "_itemsArray",
    "_armoryWeaponArrays", "_armoryWeaponNames", "_armoryWeaponAmounts",
"_armoryWeaponNamesCount",

    "_armoryWeaponName", "_armoryItemAmount",

    "_armoryItemArrays",

    "_armoryBoxArrays", "_armoryBoxArraysCount",

    "_armoryBoxArray", "_arrayNames", "_arrayAmounts",
    "_arrayNamesCount",

    "_productAmount",

    "_supplierProductList",
    "_supplierAmmunitionList"
  ];

  // prepare for GUI operations
  disableSerialization;

  // supplier and item type has been selected
  if (ANT_NUMBER_SUPPLIERS_SELECTION >= 0 && ANT_NUMBER_SUPPLIERS_PRODUCT_TYPE_SELECTION
>= 0) then
  {
    // get base dialog
    _baseDialog = (findDisplay 427001);

    // check which supplier is selected
    _supplierString = "";

    switch (ANT_NUMBER_SUPPLIERS_SELECTION) do
    {
      case 0:
```

```
      {
        _supplierString = "Blue";
      };

      case 1:
      {
        _supplierString = "Green";
      };

      case 2:
      {
        _supplierString = "Red";
      };

      default
      {

      };
    };

    // supplier is selected
    if (_supplierString != "") then
    {
      // check which type is selected
      _productTypeString = "";

      switch (ANT_NUMBER_SUPPLIERS_PRODUCT_TYPE_SELECTION) do
      {
        case 0:
        {
          _productTypeString = "All";
        };

        case 1:
        {
          _productTypeString = "Items";
        };

        case 2:
        {
          _productTypeString = "BackPacks";
        };

        case 3:
        {
          _productTypeString = "Sidearms";
        };

        case 4:
        {
          _productTypeString = "Shotguns";
        };

        case 5:
        {
          _productTypeString = "SubmGuns";
        };

        case 6:
        {
          _productTypeString = "AssaultRifles";
        };

        case 7:
        {
          _productTypeString = "Machineguns";
        };

        case 8:
        {
          _productTypeString = "SniperRifles";
        };

        case 9:
        {
          _productTypeString = "Launchers";
        };
```

```
        case 10:
        {
          _productTypeString = "Explosives";
        };

        default
        {

        };
      };

      // item type is selected
      if (_productTypeString != "") then
      {
        // create value color array
        _valueColor = [];

        // buy page open
        if (!(ctrlEnabled (_baseDialog displayCtrl 427103))) then
        {
          // get selected skills
          _selectedSkills = ANT_LOGIC_CLASS_DATA getVariable "SELECTED_SKILLS";

          // count selected skills
          _selectedSkillsCount = count _selectedSkills;

          // there are selected skills
          if (_selectedSkillsCount > 0) then
          {
            // create list box arrays
            _productsArray = [];
            _ammunitionArray = [];

            // loop through skills
            for [{_i = 0}, {_i < _selectedSkillsCount}, {_i = _i + 1}] do
            {
              // get skill name
              _skillClassName = _selectedSkills select _i;

              // get skill details
              _skillsConfig = (missionConfigFile >> "cfgAssignment" >> "Skills" >>
_skillClassName);
              _skillName = getText (_skillsConfig >> "name");
              _skillDescription = getText (_skillsConfig >> "description");
              _skillOwnerClass = getText (_skillsConfig >> "ownerClass");
              _unlocksClass = (_skillsConfig >> "unlocks");

              // create unlocks arrays
              _unlocksArrays =
              [
                // type, array, config name
                ["Items", (getArray (_unlocksClass >> "items")), "cfgWeapons"],
                ["Ammunition", (getArray (_unlocksClass >> "ammunition")),
"cfgMagazines"],
                ["BackPacks", (getArray (_unlocksClass >> "backPacks")), "cfgVehicles"],
                ["Weapons", (getArray (_unlocksClass >> "weapons")), "cfgWeapons"]
              ];

              // get unlocks arrays count
              _unlocksArraysCount = count _unlocksArrays;

              // loop through unlocks arrays
              for [{_j = 0}, {_j < _unlocksArraysCount}, {_j = _j + 1}] do
              {
                // get unlocks array data
                _unlocksArray = _unlocksArrays select _j;
                _type = _unlocksArray select 0;
                _array = _unlocksArray select 1;
                _configName = _unlocksArray select 2;

                // get array count
                _arrayCount = count _array;

                // loop through array
                for [{_k = 0}, {_k < _arrayCount}, {_k = _k + 1}] do
                {
```

```
                    // get product class name
                    _productClassName = (_array select _k);

                    // get product ANT config data
                    _productANTConfig = (missionConfigFile >> "cfgAssignment" >> "Products"
>> _type >> _productClassName);
                    _productANTSupplier = getText (_productANTConfig >> "supplier");
                    _productANTType = getText (_productANTConfig >> "type");

                    // product has the right supplier and type
                    if (_productANTSupplier == _supplierString && _productANTType ==
_productTypeString || _productANTSupplier == _supplierString && _productTypeString ==
"All" || _productANTSupplier == "" && _productANTType == _productTypeString ||
_productANTSupplier == "" && _productTypeString == "All") then
                    {
                      // get product ANT config data
                      _productANTName = getText (_productANTConfig >> "name");
                      _productANTDescription = getText (_productANTConfig >> "description");

                      // get product BIS config data
                      _productBISConfig = (configFile >> _configName >> _productClassName);
                      _productBISName = getText (_productBISConfig >> "displayName");
                      _productBISDescription = getText (_productBISConfig >>
"descriptionShort");
                      _productBISPicture = getText (_productBISConfig >> "picture");

                      _productName = _productANTName;

                      // ANT name is empty
                      if (_productName == "") then
                      {
                        // use BIS name
                        _productName = _productBISName;
                      };

                      _productDescription = _productANTDescription;

                      // ANT description is empty
                      if (_productDescription == "") then
                      {
                        // use BIS description
                        _productDescription = _productBISDescription;
                      };

                      // use BIS image
                      _productPicture = _productBISPicture;

                      // get product base price
                      _productBasePrice = ANT_LOGIC_SUPPLIERS_DATA getVariable (_type + "_"
+ _productClassName + "_BASE_PRICE");

                      // calculate product price
                      _productPrice = -1 * (_productBasePrice);

                      // create product data array
                      _productDataArray =
                      [
                        ["", _productName, (format ["$ %1", _productPrice])],
                        [0, 0, 0],
                        [_productPicture, format ["['%1', '%2', '%3', '%4', '%5']",
_productPicture, _productName, _productDescription, _productANTSupplier, _productANTType],
format ["[%1, '%2', '%3']", _productPrice, _type, _productClassName]]
                      ];

                      // product is ammunition
                      if (_type == "Ammunition") then
                      {
                        _ammunitionArray = _ammunitionArray + [_productDataArray];
                      }
                      else // product is not ammunition
                      {
                        _productsArray = _productsArray + [_productDataArray];
                      };
                    };
                  };
                };
              };
```

```
          };

          // set value color to red
          _valueColor = [0.75, 0, 0, 1];
        }
        else // sell page open
        {
          // get player UID
          _playerUID = player call ANT_F_GET_PLAYER_UID;

          // get player armory box
          _playerArmoryBox = ANT_LOGIC_PLAYER_DATA getVariable (_playerUID +
"_ARMORY_BOX");

          // create armory box items arrays
          _armoryItemNames = [];
          _armoryItemAmounts = [];

          // get items from config
          _itemConfig = (missionConfigFile >> "cfgAssignment" >> "Products" >> "Items");
          _itemConfigCount = count _itemConfig;
          _itemsArray = [];

          for [{_i = 0}, {_i < _itemConfigCount}, {_i = _i + 1}] do
          {
            _itemsArray = _itemsArray + [configName (_itemConfig select _i)];
          };

          // create armory box weapons array
          _armoryWeaponArrays = getWeaponCargo _playerArmoryBox;
          _armoryWeaponNames = _armoryWeaponArrays select 0;
          _armoryWeaponAmounts = _armoryWeaponArrays select 1;

          // count armory box weapons array
          _armoryWeaponNamesCount = count _armoryWeaponNames;

          // loop through armory box weapons array
          for [{_i = 0}, {_i < _armoryWeaponNamesCount}, {_i = _i + 1}] do
          {
            // get armory box weapon
            _armoryWeaponName = _armoryWeaponNames select _i;

            // weapon is an item
            if (_armoryWeaponName in _itemsArray) then
            {
              // get item amount
              _armoryItemAmount = _armoryWeaponAmounts select _i;

              // add item to item arrays
              _armoryItemNames = _armoryItemNames + [_armoryWeaponName];
              _armoryItemAmounts = _armoryItemAmounts + [_armoryItemAmount];

              // mark item for deletion from weapons array
              _armoryWeaponNames set [_i, objNull];
              _armoryWeaponAmounts set [_i, objNull];
            };
          };

          // delete items from the weapons arrays
          _armoryWeaponNames = _armoryWeaponNames - [objNull];
          _armoryWeaponAmounts = _armoryWeaponAmounts - [objNull];

          // combine arrays
          _armoryWeaponArrays = [_armoryWeaponNames, _armoryWeaponAmounts];
          _armoryItemArrays = [_armoryItemNames, _armoryItemAmounts];

          // create list box arrays
          _productsArray = [];
          _ammunitionArray = [];

          // create unlocks arrays
          _armoryBoxArrays =
          [
            // type, array, config name
            ["Items", _armoryItemArrays, "cfgWeapons"],
            ["Ammunition", (getMagazineCargo _playerArmoryBox), "cfgMagazines"],
            ["BackPacks", (getBackpackCargo _playerArmoryBox), "cfgVehicles"],
```

```
            ["Weapons", _armoryWeaponArrays, "cfgWeapons"]
          ];

          // get unlocks arrays count
          _armoryBoxArraysCount = count _armoryBoxArrays;

          // loop through unlocks arrays
          for [{_i = 0}, {_i < _armoryBoxArraysCount}, {_i = _i + 1}] do
          {
            // get unlocks array data
            _armoryBoxArray = _armoryBoxArrays select _i;
            _type = _armoryBoxArray select 0;
            _arrayNames = (_armoryBoxArray select 1) select 0;
            _arrayAmounts = (_armoryBoxArray select 1) select 1;
            _configName = _armoryBoxArray select 2;

            // get array count
            _arrayNamesCount = count _arrayNames;

            // loop through array
            for [{_j = 0}, {_j < _arrayNamesCount}, {_j = _j + 1}] do
            {
              // get product data
              _productClassName = _arrayNames select _j;
              _productAmount = _arrayAmounts select _j;

              // get product ANT config data
              _productANTConfig = (missionConfigFile >> "cfgAssignment" >> "Products" >>
_type >> _productClassName);
              _productANTSupplier = getText (_productANTConfig >> "supplier");
              _productANTType = getText (_productANTConfig >> "type");

              // product has the right supplier and type
              if (_productANTSupplier == _supplierString && _productANTType ==
_productTypeString || _productANTSupplier == _supplierString && _productTypeString ==
"All" || _productANTSupplier == "" && _productANTType == _productTypeString ||
_productANTSupplier == "" && _productTypeString == "All") then
              {
                // get product ANT config data
                _productANTName = getText (_productANTConfig >> "name");
                _productANTDescription = getText (_productANTConfig >> "description");

                // get product BIS config data
                _productBISConfig = (configFile >> _configName >> _productClassName);
                _productBISName = getText (_productBISConfig >> "displayName");
                _productBISDescription = getText (_productBISConfig >>
"descriptionShort");
                _productBISPicture = getText (_productBISConfig >> "picture");

                _productName = _productANTName;

                // ANT name is empty
                if (_productName == "") then
                {
                  // use BIS name
                  _productName = _productBISName;
                };

                _productDescription = _productANTDescription;

                // ANT description is empty
                if (_productDescription == "") then
                {
                  // use BIS description
                  _productDescription = _productBISDescription;
                };

                // use BIS image
                _productPicture = _productBISPicture;

                // get product base price
                _productBasePrice = ANT_LOGIC_SUPPLIERS_DATA getVariable (_type + "_" +
_productClassName + "_BASE_PRICE");

                // calculate product price
                _productPrice = round ((_productBasePrice / 3) * 2);
```

```
                // create product data array
                _productDataArray =
                [
                  ["", format ["%1 x %2", _productAmount, _productName], (format ["$ %1",
_productPrice])],
                  [0, 0, 0],
                  [_productPicture, format ["['%1', '%2', '%3', '%4', '%5']",
_productPicture, _productName, _productDescription, _productANTSupplier, _productANTType],
format ["[%1, '%2', '%3']", _productPrice, _type, _productClassName]]
                ];

                // product is ammunition
                if (_type == "Ammunition") then
                {
                  // add product data to the ammunition array
                  _ammunitionArray = _ammunitionArray + [_productDataArray];
                }
                else // product is not ammunition
                {
                  // add product data to the products array
                  _productsArray = _productsArray + [_productDataArray];
                };
              };
            };
          };

          // set value color to green
          _valueColor = [0.5703125, 0.8125, 0.3125, 1];
        };

        // get controls
        _supplierProductList = _baseDialog displayCtrl 427440;
        _supplierAmmunitionList = _baseDialog displayCtrl 427446;

        // update product lists
        lnbClear _supplierProductList;
        lnbAddArray [427440, _productsArray];

        lnbClear _supplierAmmunitionList;
        lnbAddArray [427446, _ammunitionArray];

        // get equipment list row count
        _supplierProductListRowCount = (lnbSize 427440) select 0;

        // loop through rows
        for [{_i = 0}, {_i < _supplierProductListRowCount}, {_i = _i + 1}] do
        {
          // set row picture
          lnbSetPicture [427440, [_i, 0], (lnbData [427440, [_i, 0]])];

          // set price color
          lnbSetColor [427440, [_i, 2], _valueColor];
        };

        // get ammunition list row count
        _supplierAmmunitionListRowCount = (lnbSize 427446) select 0;

        // loop through rows
        for [{_i = 0}, {_i < _supplierAmmunitionListRowCount}, {_i = _i + 1}] do
        {
          // set row picture
          lnbSetPicture [427446, [_i, 0], (lnbData [427446, [_i, 0]])];

          // set price color
          lnbSetColor [427446, [_i, 2], _valueColor];
        };

        // equipment is selected
        if ((lnbCurSelRow 427440) > -1) then
        {
          // get selection data from equipment list
          ANT_ARRAY_SUPPLIERS_PRODUCT_SELECTION_DATA = call compile (lnbData [427440,
[(lnbCurSelRow 427440), 1]]);
        }
        else
        {
          // ammunition is selected
```

```
          if ((lnbCurSelRow 427446) > -1) then
          {
            // get selection data from ammunition list
            ANT_ARRAY_SUPPLIERS_PRODUCT_SELECTION_DATA = call compile (lnbData [427446,
[(lnbCurSelRow 427446), 1]]);
          }
          else // nothing is selected
          {
            ANT_ARRAY_SUPPLIERS_PRODUCT_SELECTION_DATA = [];
          };
        };

        // update product details
        [] call ANT_F_SUPPLIERS_UPDATE_PRODUCT_DETAILS;
      };
    };
  };
};
```

LIITE 12 Funktion ANT_F_SUPPLIERS_UPDATE_PRODUCT_DETAILS sisältö

```
// update product details
ANT_F_SUPPLIERS_UPDATE_PRODUCT_DETAILS =
{
  private
  [
    "_baseDialog",
    "_detailsPicture", "_detailsText",

    "_productName", "_productDescription", "_productSupplier", "_productType",
    "_detailsString", "_detailsRowNumber",

    "_detailsTextPosition", "_detailsTextNewHeight"
  ];

  // prepare for GUI operations
  disableSerialization;

  // get base dialog
  _baseDialog = (findDisplay 427001);

  // get controls
  _detailsPicture = _baseDialog displayCtrl 427452;
  _detailsText = _baseDialog displayCtrl 427454;

  // product is selected
  if ((count ANT_ARRAY_SUPPLIERS_PRODUCT_SELECTION_DATA) > 0) then
  {
    // set product picture
    _detailsPicture ctrlSetText (ANT_ARRAY_SUPPLIERS_PRODUCT_SELECTION_DATA select 0);

    // get product config data
    _productName = ANT_ARRAY_SUPPLIERS_PRODUCT_SELECTION_DATA select 1;
    _productDescription = ANT_ARRAY_SUPPLIERS_PRODUCT_SELECTION_DATA select 2;
    _productSupplier = ANT_ARRAY_SUPPLIERS_PRODUCT_SELECTION_DATA select 3;
    _productType = ANT_ARRAY_SUPPLIERS_PRODUCT_SELECTION_DATA select 4;

    // create details string variables
    _detailsString = "<t size='1.5'>" + _productName + "</t><br />";
    _detailsRowNumber = 2.5;

    // add product description
    _detailsString = _detailsString + "<t color='#808080'>" + _productDescription +
"</t><br />";
    _detailsRowNumber = _detailsRowNumber + 1 + ([1, _productDescription] call
ANT_F_DIALOG_CALCULATE_STRUCTURED_TEXT_ROW_NUMBER);

    // set supplier name according to supplier data
    switch (_productSupplier) do
    {
      case "Blue":
      {
        _productSupplier = "<br /><t color='#254061'>Capitalist Inquiry Agency</t>";
      };

      case "Green":
      {
        _productSupplier = "<br /><t color='#4f6228'>Suave Intelligence Service</t>";
      };

      case "Red":
      {
        _productSupplier = "<br /><t color='#632523'>Комета Государственной
Безопасности</t>";
      };

      default
      {
        _productSupplier = "<br /><t color='#808080'>All</t>";
      };
    };

    // add product supplier to the details string
    _detailsString = _detailsString + "<br /><t size='1.3'>Supplier:</t>";
```

```
    _detailsRowNumber = _detailsRowNumber + 1.3;

    _detailsString = _detailsString + _productSupplier;
    _detailsRowNumber = _detailsRowNumber + 2;

    // set details control height
    _detailsTextPosition = ctrlPosition _detailsText;
    _detailsTextNewHeight = (((ctrlPosition _detailsPicture) select 3) / 3.5) *
(_detailsRowNumber / 2);
    _detailsText ctrlSetPosition [_detailsTextPosition select 0, _detailsTextPosition
select 1, _detailsTextPosition select 2, _detailsTextNewHeight];
    _detailsText ctrlCommit 0;

    // set details control text
    _detailsText ctrlSetStructuredText (parseText _detailsString);
  }
  else // product has not been selected
  {
    // clear details picture and text
    _detailsPicture ctrlSetText "";
    _detailsText ctrlSetStructuredText (parseText "");
  };
};
```