TAMPEREEN TEKNILLINEN YLIOPISTO

JUHO-MIKKO HEINONEN

VUOROPOHJAISEN PELIN SYNKRONOINTI EPÄVARMASSA VERKKOYMPÄRISTÖSSÄ

Diplomityö

Tarkastaja: professori Seppo Kuikka
Tarkastaja ja aihe hyväksytty
automaatio-, kone- ja materiaali-
tekniikan tiedekuntaneuvoston
kokouksessa 4. marraskuuta 2009

TIIVISTELMÄ

TAMPEREEN TEKNILLINEN YLIOPISTO
Automaatiotekniikan koulutusohjelma
HEINONEN, JUHO-MIKKO: Vuoropohjaisen pelin synkronointi epävarmassa verkkoympäristössä
Diplomityö, 40 sivua
Toukokuu 2010
Pääaine: Automaatioteknologia
Tarkastaja: professori Seppo Kuikka
Avainsanat: Synkronointi, tiedostopalvelin, transaktiot, verkkoympäristö

Synkronointi on keino saattaa usean sovelluksen tietomallit samaan tilaan. Tämä voidaan toteuttaa suoraan sovellusten välisellä tiedonvaihdolla, tai käyttäen erillistä palvelinta synkronoinnin apuna. Molemmissa tavoissa tiedonsiirto voidaan suorittaa käyttäen standardeja verkkoprotokollia, mutta jos verkko ei ole aina käytettävissä, on sovellusten välinen tiedonvaihto ongelmallista järjestää. Mikäli sovellukset sijaitsevat samalla koneella tai käytettäessä synkronointipalvelinta voivat sovellukset ottaa yhteyden hyväksi katsomallaan tavalla, yksinkertaisimmillaan tavallisella tiedostorajapinnalla.

Työssä toteutettiin synkronointimoduuli vuoropohjaiselle pelille. Peli toimii ympäristössä, jossa ohjelmistot pääsevät käsiksi samaan verkkojakoon, mutta käyttäjien ei odoteta voivan vaikuttaa palomuuriasetuksiin. Lisäksi osa pelin tiedoista koostuu dokumenteista, joiden on oltava luettavissa myös pelin ulkopuolelta. Pelissä on kaksi roolia: tuomari ja pelaaja. Yhteen peliin voi liittyä useampia pelaajia, mutta tuomareita on vain yksi. Pelin käyttöympäristöstä ja rakenteesta johtuen synkronointitavaksi valittiin tiedostopohjainen synkronointi tiedostopalvelimen avulla, jossa tiedonvaihto tapahtuu tiedostorajapintaa hyväksi käyttäen. Tiedostopalvelimella olevat tiedot tallennettiin salaamattomaan tiedostorakenteeseen. Tällä valinnalla käyttäjät voivat tarvittaessa avata synkronointipalvelimella olevia dokumentteja ilman peliä, käyttäjien ei tarvitse muuttaa palomuuriasetuksia ja synkronointi voidaan tarvittaessa suorittaa ulkoisella siirrettävällä medialla, kuten USB-muistilla.

ABSTRACT

TAMPERE UNIVERSITY OF TECHNOLOGY
Master's Degree Programme in Automation Engineering
HEINONEN, JUHO-MIKKO: Synchronization of the turn based game in an uncertain network environment
Master of Science Thesis, 40 pages
May 2010
Major: Automation Technology
Examiner: Professor Seppo Kuikka
Keywords: Synchronization, fileserver, transactions, network environment

Synchronization is a way to set data models of multiple applications into the same state. This can be achieved by direct data exchange between applications or by using a separate server to help synchronization. In both cases data transmission can be done by using standard network protocols, but if a network is unavailable, data transmission may be difficult to be arranged. When applications are on the same computer or synchronization server is used, applications can communicate by the way they see proper, for example using file interface.

In this thesis a synchronization module for a turn based game was implemented. The game operates in an environment where applications can access same network share, but users are not expected to be able to configure the firewall settings. In addition, part of the game data consists of documents, which must be readable without using the game. In the game there are two roles: referee and player. In a single game there can be multiple players, but only one referee. Because of limitations in use and structure of the game, synchronization method using files with a help of file server was chosen. In this method data exchanges are done by file interface. In the file server data is stored in file structure without encryption. By this choice users can access files in the file server without the game, firewall configuration can be left untouched and synchronization can be done by using mobile media such as an USB-memory.

ALKUSANAT

Tämä diplomityö on tehty työskennellessäni Insta DefSec Oy:ssä Tampereella. Esitän kiitokseni työn tarkastajalle, professori Seppo Kuikalle, sekä työn ohjaajalle Seppo Merikoskelle. Haluan myös kiittää työtovereitani Insta DefSec Oy:llä työn käytännön osuuden testauksessa avustamisessa sekä toteutusehdotusten antamisesta.

Kiitokset ansaitsee myös vaimoni Aino Haikala, joka on tukenut minua työn tekemisessä ja hoitanut poikaamme Iiroa antaen minulle aikaa työn tekemiseen. Myös poikani Iiro ansaitsee kiitokset, sillä hänen kasvamisensa seuraaminen on antanut paljon ilonaiheita työn tekemisen aikana.

Tampereella 23.5.2010,

Juho-Mikko Heinonen

SISÄLLYS

1. JOHDANTO

Verkkoyhteyksien jatkuvasti nopeutuessa ja levittyessä ovat hajautetut ohjelmat ja verkon avulla pelattavat moninpelit lisääntyneet voimakkaasti. Jotta hajautetuilla ohjelmilla saavutetaan etua perinteisiin ohjelmiin verrattuna, tulee näiden ohjelmien pystyä kommunikoimaan keskenään. Aina pelkkä kommunikointi ei ole kuitenkaan tarpeeksi, vaan ohjelmien tulee saada tietomallinsa täsmälleen samanlaisiksi. Tätä tilojen samaksi saattamista kutsutaan synkronoinniksi.

Synkronoinniksi kutsutaan myös muiden kuin ohjelmien tietomallin saattamista samaan tilaan. Tietokoneiden käytön muututtua yhä enemmän yksi kone – monta käyttäjää -tyyppisestä käytöstä yksi käyttäjä – monta konetta -tyyppiseksi, on käyttäjille muodostunut tarve saattaa kaikkien koneiden tiedostot yhteneviksi. Hajautettuihin ohjelmistoihin ja moninpeleihin verrattuna on tavallista, että käyttäjän tietokoneet eivät ole verkkoyhteydessä toisiinsa, tai verkkoyhteys on päällä vain tietyissä tilanteissa. Myös tällaista ongelmaa varten on luotu synkronointisovelluksia.

Kun hajautetut ohjelmistot eivät pysty käyttämään verkkoyhteyksiä hyödykseen synkronoitumista varten, ei synkronointia voida suorittaa totutuilla tavoilla. Tämän työn tavoitteena olikin toteuttaa synkronointi peliin, joka toimii epävarmassa verkkoympäristössä. Pelin synkronoinnissa ei täten voida luottaa verkkoyhteyksien avulla tehtävään synkronointiin, vaan ongelmaa tulee tarkastella jollakin muulla tavoin.

Synkronoitava peli muodosti reunaehdot synkronointitoteutukselle, sillä synkronointiosuus toteutettiin erillisenä moduulina, joka otettiin pelissä käyttöön ohjelmistokehystä käyttäen. Reunaehdoista tärkeimmät olivat ympäristövaatimukset, käytetty kehityskieli sekä pelin synkronoitava tietomalli. Lisäksi pelin vuoropohjaisuus ja normaalin etenemismallin tunteminen ohjasivat toteutettavan synkronointimoduulin kehitystä.

Synkronointimoduulia suunniteltaessa huomioon otettiin moduulin toteutuksessa käytettävien tekniikoiden hyödynnettävyys myös muissa vastaavissa synkronointiongelmissa. Moduulissa käytettiin myös abstraktiotasoja helpottamaan tätä uudelleenkäytettävyyttä sekä mahdollistamaan moduulin kehittäminen myöhemmin.

Seuraavassa luvussa kerrotaan moninpeleihin liittyvistä pelitavoista ja pelityypeistä ja näiden vaikutuksista pelien toteuttamiseen.

Luvussa 3 kerrotaan datan synkronoinnin erilaisista muodoista, datan eheyden säilyttämisestä synkronoinnin yhteydessä sekä datan käsittelystä synkronoinnissa.

Luvussa 4 kerrotaan työn lähtökohdista, kuten toteutettavan työn käyttöympäristöstä, ja selvitetään synkronoitavan pelin toimintaa, toteutustapaa ja tietomallia.

Luvussa 5 kerrotaan työn toteuttamisesta. Ensin kerrotaan miten toteutettavan moduulin toteutusta suunniteltiin, jonka jälkeen esitetään toteutuksen yksityiskohdat. Lopuksi kerrotaan moduulin käyttötavoista pelin osana.

Luvussa 6 tarkastellaan työn onnistumista käyttövarmuuden, suorituskyvyn ja käytettävyyden osalta sekä pohditaan mahdollisia kehityskohtia.

Luvussa 7 tehdään yhteenveto työstä.

2. MONINPELAAMINEN

Moninpelaamisella tarkoitetaan saman pelin pelaamista useamman pelaajan toimesta. Käytännössä tämä voi tarkoittaa pelin pelaamista yhdellä koneella, usealla lähiverkossa toimivalla koneella tai internetiin kytkeytyneillä koneilla.

2.1. Yksittäisellä koneella pelattavat moninpelit

Yksittäisellä koneella pelattavien moninpelien keskeisinä ongelmina ovat miten pelaajat voivat vaikuttaa peliin sekä miten eri pelaajat voivat tarkkailla peliä omasta näkökulmastaan. Moninpelien toteuttaminen yhdelle koneelle ei kuitenkaan aiheuta merkittäviä ongelmia, sillä ne ovat luonteeltaan laajennettuja yksinpelejä.

Yleinen tapa ratkaista usean pelaajan vaikuttaminen peliin on usean ohjauslaitteen, kuten näppäimistön ja joystickin käyttäminen. Tällöin eri ohjauslaitteet vaikuttavat pelin eri osiin. Pelin tapahtumien näyttämisen ongelman ratkaisu riippuu pelin tyypistä. Mikäli peli on kuvattu pelaajan näkökulmasta, on kaikille pelaajille tuotettava oma näkymä pelistä. Usein tämä on toteutettu jakamalla näyttö pelaajien kesken. Jos taas peli on kuvattu kauempaa, voidaan kaikki pelaajat näyttää samalla ruudulla yhtäaikaisesti. Joskus myös tällaisissa pelityypeissä käytetään ruudun jakamista, jolloin näytettävä kuva voidaan keskittää pelaajan ohjaamaan hahmoon.

Toinen tapa toteuttaa yhdellä koneella tapahtuva moninpeli, on pelin jakaminen vuoroihin. Tällöin pelaajat vaihtavat vuoroja jonkin logiikan mukaisesti ja käyttävät pelaamiseen samoja ohjauslaitteita ja näyttöjä. (Wierzbicki 1999)

2.2. Usealla koneella pelattavat moninpelit

Usealla koneella pelattavat moninpelit poikkeavat yhdellä koneella pelattavista enimmäkseen toteutuksen tasolla; peli voi ulkoisesti näyttää samalta riippumatta siitä, pelataanko sitä yhdellä vai usealla koneella. Kuitenkin myös ulkoiset erot ovat selvät sillä saman koneen ääreen ei ole tarvetta mahduttaa useaa pelaajaa, pelaajat voivat vaikuttaa peliin samanlaisilla ohjainlaitteilla ja näyttöruutua ei tarvitse jakaa muiden pelaajien kesken.

Merkittävin toteutuksen eroista on tarve välittää pelitapahtumia usean koneen välillä. Tällöin pelaajan tekemät muutokset eivät vaikuta ainoastaan yhteen tietomalliin, vaan muutokset tulee välittää kaikille samaan peliin liittyville pelaajille.

Muutosten välittämiseen voidaan käyttää suoraa tiedonvaihtoa pelien kesken. Tällaisessa suorassa tiedonvaihdossa pelit voivat verkottua rengasmaisesti tai verkkomaisesti. Rengasmaisessa verkkoratkaisussa ei kaistan tarve muodostu liian

suureksi suurillakaan pelaajamäärillä, mutta koska tietojen tulee kulkea monen solmun kautta, voi tämä tapa hidastaa tiedonvaihtoa suuresti. Verkkomaisessa verkkoratkaisussa kaikki pelit kommunikoivat toistensa kanssa, joka vaatii paljon vapaata kaistaa. Kuitenkin tiedot muutoksista tulevat samanaikaisesti kaikille, joka parantaa pelitilojen yhdenmukaisuutta.

Toinen tapa muutosten välittämiseen on palvelimen käyttö. Tällöin kaikki pelaajat ovat yhteydessä palvelimeen, joka pitää yllä pelin tilaa ja välittää siihen liittyvät muutokset peleille. Palvelin voi olla täysin erillinen tai yksi peleistä voi toimia palvelimen ominaisuudessa. Mikäli yksi peleistä on palvelin, tulee sitä ajavan laitteiston olla tarpeeksi tehokas, jotta se voi hallita yksittäisen pelaajan tietojen lisäksi koko pelin tilaa. Toteutuksessa pitää olla myös huomioituna palvelimena toimivan pelin mahdollisesti lyhyemmät vasteajat, jotka saattavat tuottaa lisäetua muihin peleihin nähden.

Kun pelejä pelataan usealla koneella, voivat pelaajat pyrkiä saamaan epäreilua etua muihin nähden käyttämällä muokattuja peliversioita. Nämä tapaukset voidaan huomioida varmistamalla pelien muokkaamattomuus sekä valitsemalla vaihdettava tieto siten, että vaihdettavaa tietoa muokkaamalla ei voida vaikuttaa pelin kulkuun. (Cox 2000)

2.3. Vuoropohjaiset moninpelit

Vuoropohjaiset moninpelit ovat yksinkertaisimmillaan lauta- tai korttipelejä, joista on tehty digitaalinen versio. Vaikka yksinkertaisin lähtökohta vuoropohjaiselle pelille on lautapeli, laajentavat nämä pelit usein pelien perusrakenteita tai niitä ei ole lainkaan olemassa.

Vuoropohjaisissa moninpeleissä jokainen pelaaja vie vuorollaan peliä haluamaansa suuntaan. Vuoro voi koostua yhdestä siirrosta, kuten shakissa, tai monimutkaisesta yhdistelmästä siirtoja. Jos vuorot ovat peräkkäisiä, joutuvat toiset pelaajat odottamaan vuorossa olevan pelaajan tekemää siirtoa. Vuorot voivat olla myös yhtäaikaisia, jolloin jokainen pelaaja tekee siirtonsa yhtäaikaisesti ja vasta kaikkien ollessa valmiita paljastetaan siirrot samanaikaisesti.

Vuoropohjaisissa moninpeleissä ei tiedonvaihto ole useinkaan ongelma, sillä vuorojen tapahtumat voidaan lähettää muille pelaajille vuoron loputtua. Näissä tapauksissa lähetyksen viipyminen ei ole suuri ongelma, sillä vuoron tapahtumat eivät ole sidottuja reaaliaikaan. Koska tiedonvaihdon nopeus ei ole ongelma, voidaan vuoropohjaisten pelien verkkoratkaisuna käyttää ongelmitta suoraa tiedonvaihtoa pelien kesken tai palvelimen avulla tapahtuvaa tiedonvaihtoa. Lisäksi vuoropohjaisia pelejä voidaan pelata helposti samalla koneella, sillä vuoron vaihtuessa voidaan vaihtaa pelaajaa, joka suorittaa oman vuoronsa samoilla ohjainlaitteilla kuin ensimmäinen pelaaja.

Vuoron päätteeksi voidaan lähettää lista pelaajan tekemistä muutoksista tai vuoron päätteeksi muodostunut pelitilanne. Molemmat tavat ovat toimivia ja parempi

tapa riippuu pelin tyypistä: Mikäli vuoro koostuu yhdestä siirrosta, on parempi lähettää ainoastaan kuvaus siirrosta, sillä se on paljon pienempi ja samalla on mahdollista varmistaa siirron oikeellisuus. Jos vuoro koostuu monesta pienestä toiminnosta, mahtuu näiden muodostama lopputilanne usein pienempään tilaan kuin toiminnot itsessään. Joissakin peleissä on kuitenkin tarkoituksenmukaista näyttää muille pelaajille vuoron pienimmätkin tapahtumat, erityisesti mikäli samaan lopputilanteeseen on mahdollista päästä useita eri reittejä.

Vuoropohjaisten pelien pelitilanteet voidaan säilyttää muistin lisäksi levyllä olevissa tiedostoissa. Jos tilanteet on tallennettu näin, voidaan pelitilanteen välittämiseen pelaajien kesken käyttää luvussa 3.1.1 kuvattua tiedostojen synkronointia.

2.4. Reaaliaikaiset moninpelit

Reaaliaikaisissa moninpeleissä ei pelillä ole yleensä koskaan pysyvää tilaa, vaan peliin liittyvät asiat ovat jatkuvassa muutoksessa. Muutokset voivat olla lähtöisin pelaajan toimista tai olla pelin luomia. Pelin tyypistä riippuu ovatko muutokset hitaita ja helposti ennustettavia vai nopeita ja ennustamattomia.

Toisin kuin vuoropohjaisissa peleissä, reaaliaikaisissa peleissä pelaajan vaikutus peliin riippuu pelaajan ohjauksen lisäksi myös ajasta, jolloin ohjaus tehdään. Pelaajan tekemään ohjauksen laatuun ja ajoitukseen taas vaikuttavat muiden pelaajien toiminta, jonka vuoksi pelaajien tulee saada tieto muiden pelaajien toimista mahdollisimman pikaisesti. Tästä johtuen ei kokonaispelitilanteen välittäminen pelaajien kesken ole usein järkevää reaaliaikaisissa peleissä, vaan pelaajat lähettävät tietoa tekemistään muutoksista.

Mikäli pelin tilanteet muuttuvat nopeasti, ei edes muutosten lähettäminen aina riitä, vaan pelien tulee ennustaa muiden pelaajien käyttäytymistä sujuvan pelitilanteen luomiseksi. Ennusteet eivät voi aina olla oikeita, jolloin pelin tilaa korjataan vastaamaan oikeita tapahtumia. Mikäli korjaukset eivät kohdistu pelaajan näkemäalueelle tai eivät ole juuri sillä hetkellä merkittäviä pelaajan kannalta, ei korjausten tekemisellä ole välitöntä kiirettä. Toisaalta mikäli muutokset vaikuttavat suoraan pelaajan tilaan tai vaativat pelaajalta välitöntä reagointia, tulee muutosten näkyä pelaajalla heti. Tällaisissa tilanteissa voidaan käyttää tapaa, jossa pelaajille välitetään hänen kannaltaan tärkeät muutokset korkealla prioriteetilla ja muut muutokset lähetetään mahdollisuuksien mukaan. Saman pelin pelaajilla on tällöin hieman eroava malli pelistä, mutta koska mallien erot eivät olennaisesti vaikuta pelin kulkuun, se ei aiheuta ongelmia. (O’Brien 1997)

3. DATAN SYNKRONOINTI

Tietoverkkojen kehittyessä hajautetut tietojärjestelmät ja sovellukset ovat tulleet yhä suositummiksi. Näissä tapauksissa samaa dataa käsitellään usean eri toimijan toimesta, monesta eri paikasta käsin. Jotta hajautuksesta saadaan hyötyä, tulee tehtyjen muutosten näkyä kaikille samaa dataa käsitteleville tahoille. Datan synkronoinnissa on kyse tavasta, jolla yhteisessä käytössä oleva data saadaan pysymään yhtenäisenä käyttöpaikasta ja ajasta riippumatta.

Luvussa käsitellään ensimmäiseksi neljää eri datan synkronoinnin muotoa. Seuraavaksi käydään läpi synkronoinnissa käsiteltävän datan eheyden säilyttämistä. Lopuksi käsitellään tapoja, joiden avulla synkronointi voidaan konkreettisesti suorittaa.

3.1. Datan synkronoinnin muodot

Vaikka datan synkronoinnin päämääränä on saattaa kaksi tai useampi datajoukko saman sisältöisiksi, on tämän päämäärän saavuttamiseen useita erilaisia tapoja. Lisäksi eri muodot mahdollistavat erityyppisiä tapoja muokata ja tarkastella synkronoitavaa datajoukkoa.

3.1.1. Tiedostojen synkronointi

Tiedostojen synkronointi tarkoittaa kahden käyttäjän tekemän tiedostojoukon yhtenäiseksi saattamista siten, että muodostetut joukot eivät sisällä turhia kopioita tiedostoista, ja joissa ei ole tiedostojen lisäksi muuta tietoa. Tiedostojen synkronointia käytetään usein, kun tietojoukkoa käsitellään usealla koneella.

Tiedostojen synkronointia suorittavilla ohjelmistoilla on kaksi päätavoitetta: havaita ristiriitaiset päivitykset ja levittää ristiriidattomia muutoksia. Näiden tavoitteiden saavuttaminen on kuitenkin vaikeaa, sillä pieni väärinkäsitys saattaa johtaa datan häviämiseen. Erityisesti käsitteen ’käyttäjän tekemä päivitys’ tarkoittaman sisällön voi tulkita useammalla tavalla, mikä johtaa vääriin synkronointituloksiin. (Balasubramaniam 1998) Tämän seurauksena monet ohjelmistot joutuvat epäselvissä tilanteissa pyytämään käyttäjää valitsemaan synkronoitavat tiedostot.

3.1.2. Versionhallinta

Versionhallinta hallitsee muutoksia tiedostoissa ja hakemistoissa. Muutokset tallennetaan sovelluksesta riippuen joko erilliseen säilytyspaikkaan tai ainoastaan hakemistojen yhteydessä oleviin aputiedostoihin. Muutoshistorian avulla käyttäjä voi myöhemmin palauttaa vanhempia versioita tarkkaillusta datajoukosta, tai tarkastella

datajoukkoon tehtyjä muutoksia. Näiden ominaisuuksien vuoksi versionhallintaa kutsutaankin joskus ”aikakoneeksi”. (Collins-Sussman 2008)

Usein versionhallintaa käytetään hallitsemaan usean käyttäjän muokkaamia datajoukkoja. Tällöin versionhallinnan avulla saadaan toteutettua myös datajoukkojen synkronoituminen kaikille käyttäjille. Koska datajoukon muutoshistoria säilytetään, eivät synkronoitaessa tehdyt mahdolliset väärät valinnat säilytettävästä versiosta aiheuta tehdyn työn menettämistä, vaan päätökset on aina peruttavissa palauttamalla datajoukon tila johonkin edelliseen, oikeampaan tilaan.

Versionhallintaa käytetään usein ohjelmistokehityksen apuna. Tämän seurauksena moniin versionhallintaa tarjoaviin sovelluksiin on lisätty tukea helpottamaan ohjelmistokoodin versiointia. Tällaisia tukia ovat muun muassa ohjelmistokoodin yhdistäminen ja ohjelmistorakenteessa tapahtuvien muutosten välittäminen versionhallintaan.

3.1.3. Hajautetut tiedostojärjestelmät

Hajautetut tiedostojärjestelmät tarjoavat mahdollisuuden päästä käsiksi dataan monesta eri paikasta. Lisäksi dataa on usein mahdollista käsitellä käyttäen erilaisia yhteysmekanismeja. Tällainen hajautettu tiedostojärjestelmä voidaan nähdä myös synkronointia poistavana tekijänä, sillä useasti data on näennäisesti vain yhdessä paikassa. Järjestelmän avulla kuitenkin mahdollistetaan saman datan näkyminen samanlaisena usealle eri käyttäjälle, minkä vuoksi myös hajautetut tiedostojärjestelmät ovat yksi datan synkronoinnin muoto.

Käsiteltäessä dataa hajautetussa tiedostojärjestelmässä on tavallista, että käsiteltävästä datasta tehdään kopio, jota käyttäjä muokkaa. Tiedostojärjestelmän toteutuksesta riippuu miten muut käyttäjät näkevät tällaisen käsittelyssä olevan datan. Yleisimmässä toteutustavassa muut käyttäjät näkevät datasta ennen muokkaamisen aloittamista olleen version, mutta kyseistä versioita ei voida ottaa muokkaukseen useassa paikassa. Toinen usein käytetty versio sallii monen käyttäjän muokata dataa, mutta muokatuista versioista ainoastaan yksi valitaan säilytettäväksi ja muissa versiossa tehdyt muutokset häviävät.

Hajautetussa tiedostojärjestelmässä olevan datan käsittely vaatii tavallisesti kiinteää yhteyttä järjestelmän ja sitä käyttävän asiakkaan välillä. Toteutuksista löytyy onneksi myös poikkeuksia tähän rajoittavaan tekijään. Esimerkiksi CODA-järjestelmällä voidaan tiedostojärjestelmään kuuluvia tiedostoja käsitellä myös yhteydettömässä tilassa. Yhteydettömässä tilassa tehdyt muutokset siirretään järjestelmään kun yhteys jälleen muodostetaan. (Kon 1996)

3.1.4. Peilaaminen

Peilaaminen tarkoittaa datan kopioimista useaan eri sijaintiin. Tällä tavoin peilattu data on helpommin saatavilla ja sen säilyvyys on varmempaa. Muutoksia dataa tehdään ainoastaan ensisijaiseen kopioon, josta ne jaetaan muihin peilattuihin datajoukkoihin.

Esimerkiksi SQL Server -tietokannassa käytetään tätä menettelyä (Rizzo 2008). Peilaamisella saavutetaan datan parempi saatavuus ja mikäli ensisijainen kopio jostain syystä vioittuu, voidaan peilatuista joukoista tehdä sille korvaava versio. Peilaamista käytetään levyjärjestelmien ja tietokantojen varmistamiseen sekä verkkosivustojen saatavuuden parantamiseen.

3.2. Datan eheys synkronoinnissa

Jotta data on käyttökelpoista myös synkronoinnin jälkeen, on sen eheydestä varmistuttava. Eheyteen kuuluu yksittäisten tietoalkioiden korruptoimattomuus, sekä se että nämä alkiot muodostavat yhdessä mielekkään datajoukon.

3.2.1. Transaktiot

Synkronoitava data säilyy eheänä, kun synkronointi hoidetaan transaktiota käyttämällä. Transaktio on joukko tapahtumia, jotka suoritetaan siten että ne näyttävät yhdeltä. Transaktion avulla datajoukko siirretään eheästä tilasta toiseen, eheään, tilaan. Transaktiossa on neljä pääominaisuutta: atomisuus, eheys, eristyvyys ja pysyvyys.

Atomisuudella tarkoitetaan transaktion suorittamista kokonaisuudessaan. Jos transaktiota ei voida suorittaa kokonaan, se perutaan. Näin ei tapahdu puolittaisia muutoksia synkronoitavassa datassa.

Eheydellä varmistetaan synkronoitavan datan siirtyvän eheästä tilasta toiseen. Transaktion aikana voi datan tila olla tilapäisesti rikkonainen, mutta jos sitä ei saada takaisin ehjäksi, tulee koko transaktio peruuttaa.

Eristyvyydellä varmistetaan että rikkonaista dataa ei käytetä päätöksentekoon. Eristyvyys tarkoittaa myös, että transaktioiden tulee toimia toisistaan riippumatta ja ne eivät saa nähdä toistensa keskeneräisiä tiloja.

Pysyvyydellä varmistetaan transaktion vaikutuksen säilyminen, vaikka järjestelmä kaatuisi transaktion jälkeen. Pysyvyys ei kuitenkaan estä samojen tietojen muuttamista myöhemmän transaktion avulla. (Mullender 1993, p.329-331)

3.2.2. Ristiriidat

Koska synkronoitavia datajoukkoja voidaan muokata toisistaan riippumatta, on mahdollisuus että nämä muutokset ovat ristiriidassa toistensa kanssa. Tällaiset ristiriidat voivat syntyä ennen ensimmäistä synkronointia tai kun synkronoitavaa dataa muokataan tai lisätään synkronoinnin jälkeen. Ristiriitaisuus voi ilmetä saman dataelementin erilaisina muokkauksina tai erillisten dataelementtien muutoksista johtuvasta synkronoitavan joukon päätymisestä epäyhtenevään tilaan.

Ristiriidat voidaan havaita pitämällä yllä tietämystä dataelementtien muutoksista, jolloin voidaan päätellä ovatko vertailtavista elementeistä muuttuneet vain toinen, molemmat vai ei kumpikaan. Yksittäisen elementin kohdalla voi kyseessä olla ristiriita, mikäli molemmat elementit ovat muuttuneet, kun taas datajoukon kohdalla

ristiriita saattaa esiintyä kun molemmissa joukoissa esiintyy muutoksia. Ristiriitojen käsittely voidaan automatisoida, mikäli synkronoitavan datan sisältö antaa tähän mahdollisuuden. Tämä automatisointi voi sisältää luvussa 3.2.3 käsiteltävää datan yhdistämistä tai erilaisiin sääntöihin perustuvaa säilytettävän version valitsemista. Automatisointia ei kuitenkaan ole aina mahdollista suorittaa, jolloin käyttäjän tulee valita synkronoitavasta datasta säilytettävät elementit.

3.2.3. Datan yhdistäminen

Datan yhdistämisellä pyritään vähentämään tehtyjen muutosten katoamista synkronoinnin yhteydessä. Mikäli samaa dataa on muokattu yhtä aikaa, eivät molemmat muokkaukset voi säilyä sellaisenaan. Mikäli muokatuista datoista valitaan vain toinen, menetetään toisen muokkaamiseen tehty työ. Tämän vuoksi synkronointi sisältää usein mahdollisuuden yhdistää dataa. Mikäli synkronoitavan datan luonne on tarkasti tiedossa, voidaan synkronointitoteutuksessa datan yhdistämistä automatisoida. Yleisin esimerkki tällaisesta on ohjelmistokehityksessä käytetty versionhallinta, jossa ohjelmistokoodia voidaan yhdistää niin kauan kuin muutokset ovat tapahtuneet koodin eri kohtiin. Mikäli automatisointi ei onnistu, voidaan käyttäjälle näyttää synkronoitavien datojen eroavaisuudet, jotka hän yhdistää manuaalisesti ennen synkronoinnin jatkumista.

3.3. Datan käsittely synkronoinnissa

3.3.1. Synkronointitoteutuksen arkkitehtuuri

Arkkitehtuurin valinnalla luodaan pääsuunnat sovellukselle. Synkronointiongelman ratkaisua voidaan lähestyä kahden toisistaan melkoisesti poikkeavan arkkitehtuurimallin kautta, jotka on esitelty seuraavaksi.

Asiakas-palvelin -malli

Asiakas-palvelin -mallissa toteutuksessa on kaksi erillistä roolia; asiakas ja palvelin. Usein asiakkaana toimivia sovelluksia on useita ja palvelimia on vain yksi tai muutama. Mallissa asiakkaat ovat yhteydessä palvelimen kanssa, eivätkä ole tietoisia toisten, samaa palvelinta käyttävien, asiakkaiden olemassaolosta. Näin yhden asiakkaan poistuminen järjestelmästä ei vaikuta muiden asiakkaiden toimintaan, mutta jos palvelin ei ole saatavilla ei yksikään asiakas pysty toimimaan. (Reese 2000)

Synkronointiongelmassa palvelin sisältää tavallisesti keskitetyn kuvan synkronoitavasta datasta, jota asiakkaat päivittävät omien tarpeidensa mukaan. Asiakkaan ja palvelimen toimintoja ei ole määritetty mallissa, jolloin toiminnot voidaan jakaa sovelluksen tarpeen mukaiseksi. Tavallisimmin käytetyt jaot on nimetty asiakkaana toimivan sovelluksen monimutkaisuuden mukaan. Nämä mallit ovat ohut ja raskas -asiakas (thick-client). Ohut-asiakas mallissa asiakkaan tehtäviin kuuluu lähinnä esittää palvelimelta saamaansa dataa, palvelimen hoitaessa tarvittavan logiikan. Raskas-

asiakas -mallissa asiakas taas hoitaa sovelluslogiikan ja palvelimen tehtävänä on toimia lähinnä tietovarastona.

Vertaisverkkomalli

Vertaisverkkomallissa sovellukset kommunikoivat keskenään ilman erillisen keskitetyn järjestelmän apua tai lupaa. Mikäli jokin sovellus poistuu verkosta, pystyvät muut sovellukset jatkamaan toimintaansa myös ilman sitä. Myös uuden sovelluksen liittyminen verkkoon onnistuu ilman toiminnan keskeyttämistä. Joskus vertaisverkkoon on liitetty myös apupalvelin, jonka avulla muut sovellukset voivat löytää muita samassa verkossa sijaitsevia sovelluksia (Androutsellis-Theotokis & Spinellis 2004).

Synkronointiongelmassa vertaisverkon jokainen sovellus pitää yllä tietoa synkronoitavasta datasta ja muutokset datassa leviävät sovellukselta toiselle. Tämä saattaa aiheuttaa datan eriytymistä, mikäli muutoksien leviämisen koordinointia ei ole otettu huomioon sovellusta suunniteltaessa.

3.3.2. Synkronoitavan datan tallennusmuoto

On mahdollista että synkronoitavaa dataa ei tallenneta mihinkään fyysiseen muotoon, vaan käytetään ainoastaan muistissa olevia tietorakenteita. Usein kuitenkin synkronoitava data on luonteeltaan sellaista, että sitä tulee pystyä käsittelemään myöhemminkin, jolloin se pitää olla tallennettavissa jollekin fyysiselle medialle. Tämän tallennusmuodon valinta vaikuttaa synkronoinnissa mahdollisien ratkaisujen määrään sekä synkronointitehokkuuteen.

Tietokanta

Tietokanta on kokoelma toisiinsa liittyvää tietoa, jonka alkioita voidaan helposti muokata, lisätä tai poistaa. Tietokannan rakenne suunnitellaan käyttötarpeen mukaan, jolloin siellä olevaa tietoa on mahdollista käsitellä tehokkaasti. Tietokantaan säilöttävät tiedot voidaan myös pakata, mikä pienentää tietokannan tarvitsemaa levytilaa, sekä toteutustavasta riippuen myös mahdollisesti vähentää tietoliikenteen määrää tietokantaa käytettäessä.

Tietokantaa käytetään yleensä erillisten hallintajärjestelmien kautta, jotka vastaavat tietokannan luomisesta, muokkaamisesta ja hallitsee tietokannan käyttöoikeuksia. Käyttöoikeuksien avulla voidaan määritellä käyttäjille muokkaamismahdollisuudet eri tietokannan alkioihin.

Usein tietokanta toteuttaa luvussa 3.2.1 esitetyt transaktio-ominaisuudet, mikä varmistaa säilytettävän tiedon eheyden. Lisäksi tietokannassa oleva tieto voidaan salata, jotta siihen ei päästä käsiksi käyttämättä hallintajärjestelmää. Samoin tietokannan alkioita voidaan lukita, millä estetään vahingollinen tietojen muokkaaminen.

Tiedostorakenne

Tiedostorakenne muodostuu tiedostojärjestelmään hakemistojen avulla järjestetystä joukosta tiedostoja. Näitä tiedostoja voidaan hallita käytetyn tiedostojärjestelmän

tarjoamien palveluiden avulla. Niihin kuuluvat yleensä tiedostojen lisääminen, poistaminen ja päivittäminen, tiedostojen käyttöoikeuksien asettaminen ja tiedostojen metatietojen asettaminen. Tiedostoissa olevat metatiedot voivat sisältää esimerkiksi tietoja muuttamispäivästä ja tiedoston lukitustiloista. Nämä tiedot ovat kuitenkin vain viitteellisiä ja käyttävä ohjelmisto voi ohittaa lukituksesta kertovan tiedon ja muokata tiedostoa siitä huolimatta. Tämän vuoksi metadatan asettamista ei voida pitää luotettavana tapana antaa ohjeita tiedoston käsittelystä muille ohjelmistoille.

Tiedostorakenteeseen tallennettuihin tietoihin on usein helppo päästä käsiksi käyttöjärjestelmän palveluita käyttäen, joten käsittelyyn ei tarvita erillisiä ohjelmia. Tarvittaessa tiedostorakenne voidaan kuitenkin myös salata, jolloin sen käsitteleminen vaatii salauksen purkuun kykenevän ohjelmiston.

Pakattu tiedostorakenne

Pakattu tiedostorakenne on pohjimmiltaan samanlainen kuin edellä esitetty tiedostorakenne. Tämä tiedostojoukko on pakattu käyttäen valittua pakkausalgoritmia, jolloin joukkoa voidaan käsitellä yhtenä, pakattuna, tiedostona. Tiedostojoukon keskittämisen lisäksi pakkaus vähentää tiedostojen tarvitsemaa levytilaa sekä mahdollistaa näiden salauksen. Toisaalta tiedostorakenteen pakkaaminen ja purkaminen vievät prosessointiaikaa, mikä voi laiteympäristöstä riippuen hidastaa tietojen käsittelyä merkittävästi.

Pakkausalgoritmin valinnasta riippuen voidaan pakattuihin tiedostoihin päästä käsiksi myös ilman erillisiä ohjelmia tai tiedostojen käsittely voidaan rajoittaan tietyn ohjelmiston kautta tehtäväksi. Tällä valinnalla voidaan rajata pakattujen tietojen käsittely ainoastaan niitä varten suunnatuille ohjelmille.

3.3.3. Tiedonvaihto

Jotta useaa sovellusta voidaan mielekkäästi käyttää yhteisen päämäärän saavuttamiseen, tulee näiden sovellusten kyetä vaihtamaan tietoa keskenään. Tiedon vaihtoon on kehitetty monia erilaisia lähestymistapoja, joista tässä esitellään kolme potentiaalista vaihtoehtoa käytettäväksi datan synkronoinnissa. Esiteltävistä tavoista ensimmäinen liittyy yleiseen tiedonvaihtoon, kahden seuraavan ollessa hajautettuihin järjestelmiin erikoistuneita tapoja.

Tiedostot

Tiedostojen avulla tapahtuvassa tiedonsiirrossa dataa sisältävät tiedostot välitetään kokonaisuudessaan synkronoitavien sovellusten välillä. Nämä tiedostot voivat olla käyttötarkoituksen mukaan sovelluskohtaista dataa, dokumentteja, kuvaa, tietokantoja tai muita tarpeellisia tiedostoja. Sovellukset käsittelevät tiedostoja tarvitsemallaan tavalla ja mikäli käsittelyn seurauksena tiedostojen sisältö vaihtuu, palautetaan muokatut (tai kaikki) tiedostot.

Tiedostojen käyttö tiedon siirtämisessä vaatii mahdollisesti suuren määrän tiedonsiirtoa, mikä vaikeuttaa tavan käyttämistä hitaissa tiedonsiirtoympäristöissä.

Toisaalta kokonaisten tiedostojen käsittelyllä voidaan välttää tiedon kokoamista erillisistä paloista, mikä parantaa tiedon eheyttä ja vähentää prosessointitarvetta.

Etäproseduurikutsut

Etäproseduurikutsujen (RPC, Remote Procedure Call) avulla voivat sovellukset käyttää toisten sovellusten palveluja. Tässä toimintatavassa kutsuva ohjelma lähettää kutsuttavalle pyynnön, joka sisältää kutsuttavan prosessin nimen ja sen tarvitseman datan. Kutsuttava ohjelma käsittelee pyynnön ja lähettää vastauksen käsittelyn tuloksesta. Koska etäproseduurikutsut perustuvat suhteellisen selkeään ja yksinkertaiseen tekniikkaan, on niiden avulla mahdollista luoda luotettavia hajautettuja järjestelmiä. (Crichlow 2001, p.106) Synkronointitapauksessa kutsuttavat prosessit olisivat esimerkiksi ilmoituksia muutoksesta datassa ja parametreina annettaisiin tiedot muutoksen laadusta.

Pistoke (socket)

Pistoke kuvaa ohjelmalle varattuja tietorakenteita, joiden avulla verkkoliikennöinti tapahtuu (Jokinen 2010). Pistokkeita käytettäessä on toinen sovelluksista aina asiakkaan ja toinen palvelimen roolissa. Tämän roolitus johtuu siitä että palvelimen on luotava oma pistoke, johon asiakas voi ottaa yhteyden oman pistokkeensa kautta. (Crichlow 2001, p.111-113) Kun kaksi ohjelmaa on näin saatu yhdistettyä toisiinsa, voidaan pistoketta käsitellä kuten tiedostoa: Toinen ohjelma kirjoittaa tietoja ja toinen lukee sen. Pistokkeiden käyttö vastaa joltakin osin etäkutsuja, mutta suurimpana erona etäkutsuihin nähden on vastausten odottamisen puuttuminen. Pistokkeisiin voidaan kirjoittaa tietoa, mutta tämän jälkeen ei sinne mahdollisesti tullutta tietoa tarvitse lukea ohjelman etenemisen sallimiseksi.

4. TYÖN LÄHTÖKOHDAT

Synkronointimoduuli toteutettiin pellin, jonka käyttöympäristö, luonne ja toteutustekniikka määrittelivät lähtökohdat työn tekemiselle. Tässä luvussa kuvataan toteutuksen käyttöympäristö ja kerrotaan pelin toimintalogiikasta sekä toteutuksesta.

4.1. Ympäristö

Jotta peliä voidaan mielekkäästi käyttää, tulee ympäristön olla annettujen vähimmäisvaatimusten mukainen. Ympäristö voi myös tarjota vähimmäisvaatimuksia laajentavia toimintamahdollisuuksia.

Pelin ohjelmistoympäristöksi vaaditaan Windows XP -käyttöjärjestelmä. Käyttöjärjestelmään tulee olla asennettuna Microsoft Office – Word 2003 sekä Java-virtuaalikoneen version 1.6. Pelin asennusaikana vaaditaan käyttäjältä järjestelmävalvojan oikeuksia käyttöjärjestelmään, mutta tavallinen käyttö onnistuu myös käyttäen rajoitettua käyttäjätiliä.

Pelin laiteympäristöksi tarvitaan vähintään 2GHz prosessorilla, 1GB RAM-muistilla varustettu tietokone, jossa on 5GB vapaata kovalevytilaa. Lisäksi pelin käyttämiseen vaaditaan näyttö, näppäimistö ja hiiri.

Pelin oletettu käyttöympäristö koostuu vähimmäisvaatimukset täyttävästä kannettavasta tietokoneesta. Koneessa olevaan Windows XP -käyttöjärjestelmään on käyttäjällä ainoastaan rajoitetun käyttäjän oikeudet. Tietokoneella ei ole yhteyttä internetiin, mutta sillä on yhteys muiden pelin instanssien kanssa yhteiseen verkkolevyyn tai verkkojakoon. Lisäksi käytettävissä on siirrettävää mediaa, kuten USB-muisteja ja kirjoitettavia CD/DVD-levyjä.

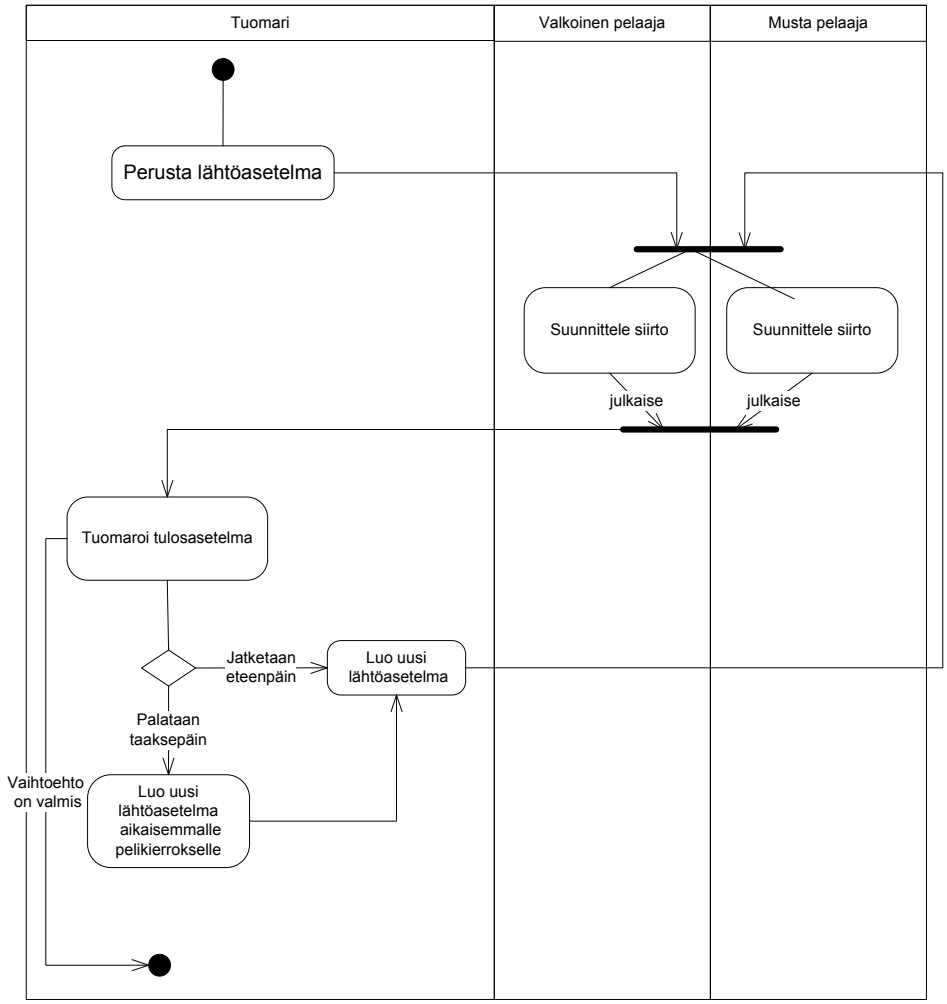
4.2. Synkronoitava peli

Synkronoitava peli on luonteeltaan vuoropohjainen strategiapeli. Siinä kahden puolen pelaajat (valkoinen ja musta) tekevät kokonaissuunnitelmat tavoitteestaan. Tämän jälkeen tuomari asettaa pelin alkutilanteen ja pelattavan tilanteen keston. Tämän jälkeen pelaajat tekevät tilanteeseen suunnitelmat, jotka parhaiten tukevat heidän kokonaissuunnitelmiensa toteutumista. Tämän jälkeen tuomari määrittelee tilanteen lopputuloksen pelaajien suunnitelmien perusteella ja tekee uuden alkutilanteen määrittelemänsä lopputuloksen perusteella.

4.2.1. Pelin toimintalogiikka

Pelissä on kolme eriävää roolia: tuomari, valkoinen pelaaja ja musta pelaaja. Jotta peliä voidaan pelata, tulee jokainen rooli olla edustettuna. Lisäksi pelaaja-rooleissa voi olla useita pelejä. Pelissä voidaan pelata yhtä aikaa useaa vaihtoehtoa, jotka voivat sisältää useampia avoimia kierroksia.

Pelin alkaessa tuomari luo kierroksille alkutilanteet, jotka välitetään pelaajille synkronoinnin avulla. Näiden alkutilanteiden perusteella pelaajat luovat suunnitelman omasta toiminnastaan kierroksen aikana. Jos mustan tai valkoisen pelaajan roolissa on usea käyttäjä, on jokaisella mahdollisuus tehdä suunnitelma erilliseen kierrokseen. Saman kierroksen yhtäaikaista suunnittelua ei tueta. Kun suunnitelmat ovat valmiit, välitetään ne tuomarille ja toisille saman puolen pelaajille synkronoinnin avulla. Kun tuomari on saanut molempien puolien suunnitelmat kierroksen kulusta, hän päättää miten suunnitelmat toteutuvat. Päätöksensä perusteella tuomari luo uuden kierroksen, joka jatkuu lopputilanteesta tai on yhtenäinen jonkin edellisen kierroksen alkutilanteen kanssa. Luotu kierros synkronoidaan pelaajien saataville. Kuvattu pelin kulku on esitetty graafisesti alla olevassa kuvassa (Kuva 4-1).



Kuva 4-1. Pelin kulku

Missä tahansa pelin vaiheessa voivat tuomari tai pelaajat muokata yhteisesti käytettävissä olevaa joukkolistaa. Listan muutokset välittyvät synkronoinnin avulla muille käyttäjille, jotka voivat ottaa muutokset huomioon omissa toiminnoissaan.

Tuomarilla on mahdollisuus lisätä ja muokata tekstiasiakirjoja, joiden perusteella pelaajat tekevät suunnitelmia kierroksiin. Nämä asiakirjat välitetään pelaajille synkronoitaessa, jonka jälkeen pelaajat voivat tarkastella niiden sisältöä. Peli ei tue pelaajien tekemiä muutoksia kyseisiin asiakirjoihin.

Pelin synkronointi tapahtuu käyttäjän toimesta. Synkronoitaessa joukkolista ja kaikki käyttäjän valmiiksi asettamat tilanteet viedään toisten käyttäjien saataville. Samalla haetaan muiden käyttäjien synkronoimat tilanteet ja päivitetään joukkolistaa. Synkronoinnin yhteydessä estetään valmiiksi asetettujen tilanteiden muokkaaminen ja joukkolistassa olleiden joukkojen poistaminen. Näiden toimintojen avulla varmistetaan, että myöhemmin tehtävät ratkaisut pelissä pohjautuvat samoihin tilanteisiin. Jos pelaaja jostain syystä haluaa muuttaa tekemiään ratkaisuja, tulee hänen pyytää tuomaria tekemään uuden kierroksen, jonka alkutilanne vastaa muutettavan tilanteen alkutilannetta.

4.2.2. Ohjelmointikieli

Peli toteutettiin käyttämällä ohjelmointikielenä Javaa. Java-ohjelmointikieli on yleiskäyttöinen, oliopohjainen ja vahvasti tyypitetty korkean tason ohjelmointikieli. Java on suunniteltu helposti omaksuttavaksi, turvalliseksi ja tuotantokäyttöiseksi kieleksi, minkä vuoksi siihen ei ole otettu mukaan uusia, testaamattomia ominaisuuksia. Java käännetään monista muista ohjelmointikielistä poiketen laiteriippumattomaksi tavukoodiksi, joka tulkitaan sellaisenaan tai muutetaan konekoodiksi ajonaikana. Tämän ominaisuuden vuoksi Javalla toteutettuja ohjelmistoja voidaan suorittaa erilaisissa ohjelmisto- ja laiteympäristöissä. (Gosling 2005, p1)

Koska ohjelmiston ajoympäristö oli tiedossa, voitiin ohjelmassa käyttää JNI:n (Java Native Interface) avulla käyttöjärjestelmäriippuvaisia elementtejä. Lisäksi voitiin käyttää tiedossa olevia käyttöjärjestelmäkutsuja käyttäen hyväksi Javan exec-metodia. Näiden ominaisuuksien käyttö tekee ohjelman riippuvaiseksi ohjelmistoympäristöstään ja erottaa tehdyn toteutuksen puhtaasta Java-sovelluksesta.

Ohjelmiston käännettiin Javan kääntäjän versiolla 1.6. Toimiakseen pelillä tulee olla käytettävissä Javan ajoympäristön (JRE, Java Runtime Environment) versio 1.6 tai suurempi. Pelin toteutusprojektissa käytettiin Eclipse-kehitysympäristön versiota 3.5.1.

4.2.3. Pelin ohjelmistokehys

Pelin toteutuksessa käytettiin hyväksi Spring-ohjelmistokehystä. Spring on ohjelmistokehys, jonka avulla Java-ohjelmistoihin voidaan saada yksinkertaisuutta, testattavuutta ja löyhiä kytköksiä (Walls 2007, p5).

Peli hyödyntää Springin tarjoamaa mahdollisuutta löyhiin kytköksiin käyttäen Dependency Injektionia. Käytettäessä Dependency Injektionia Springin avulla, tulee kytkettäville moduuleille luoda erillinen rajapinta ja toteutus. Moduulia käyttävät osat tuntevat vain rajapinnan ja rajapinnan toteuttava luokka on määritelty erillisessä xml-tiedostossa. Toteuttava luokka on näin helposti vaihdettavissa muokkaamalla xml-tiedostoa.

4.2.4. Tietojen esittäminen

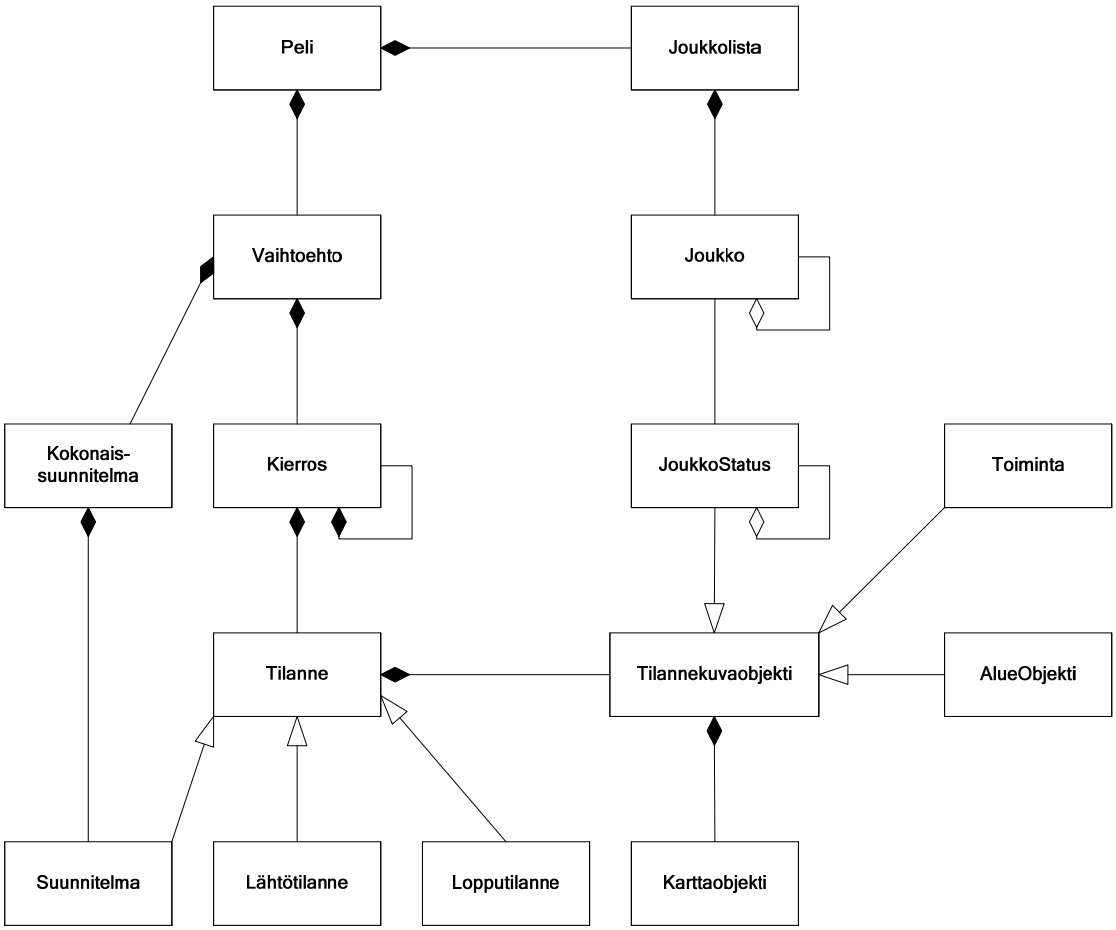
Pelin sisältämää informaatiota on tarkoitus käsitellä pääasiassa pelin kautta. Pelin avulla käsitellään kuitenkin myös tavallisia tekstiasiakirjoja, joiden käsittely myös pelin ulkopuolelta on tarpeellista. Tämän vuoksi tekstiasiakirjat tallennetaan kovalevylle normaaleina Microsoft Word -dokumentteina.

Yhtenevän esitystavan vuoksi myös muu pelin sisältämä informaatio tallennetaan helposti käsiteltävään muotoon, tässä tapauksessa xml-tiedostoiksi. Vaikka tietojen muokkaaminen ulkoa on näin tehty helposti mahdolliseksi, ei se ole suositeltavaa ja käyttäjän vastuulle onkin jätetty että tiedostojen nimet ja rakenne pysyvät oikeellisina.

4.2.5. Pelin tietomalli

Pelin tietomalli on esitetty kuvassa Kuva 4-2 Pelin tietomalli. Malli koostuu kolmen tyyppisestä elementistä:

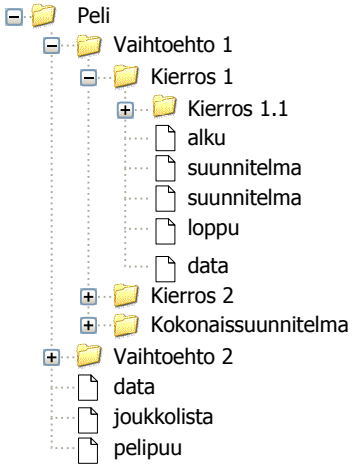
- Koostava elementti. Näillä elementeillä ei ole paljoa sisällöllistä tietoa, vaan ne toimivat pääasiassa säiliöinä. Tämän tyyppisiä elementtejä ovat peli, joukkolista, vaihtoehto, kierros, kokonaissuunnitelma ja tilanne.
- Erillisobjekti. Nämä objektit ovat sidottuja tiettyyn tilanteeseen ja niiden käyttö muussa yhteydessä ei ole järkevää. Tällaisia elementtejä ovat kaikki tilannekuvaobjektista perityt objektit.
- Yleisobjekti. Objekti, johon viittaaminen on mielekästä useasta elementistä. Näiden objektien muuttaminen vaikuttaa kaikkien niihin liittyvien objektien käyttäytymiseen. Joukkolistan joukot ovat tällaisia yleisobjekteja.



Kuva 4-2 Pelin tietomalli

Tietomallin kuvautuminen kovalevylle

Koostavat elementit voidaan tallentaa levylle joko tiedostoina tai hakemisto/tiedosto yhdistelmänä. Tiedostoina tallennetaan kaikki tilanteet sekä joukkolista. Hakemisto/tiedosto yhdistelminä tallennetaan peli, vaihtoehto, kierros ja kokonaissuunnitelma. Tiedostoista muodostuva hakemistorakenne on esitetty alla olevassa kuvassa (Kuva 4-3).



Kuva 4-3 Pelin tietomallin kuvautuminen kovalevyllä

Kuten luvussa 4.2.4 on esitetty, ovat tallennettavat tiedostot xml-muotoisia. Elementit, jotka on tallennettu hakemiston ja tiedoston yhdistelminä, muodostetaan elementin perustietoja sisältävästä data-tiedostosta sekä hakemiston muista tiedostoista ja kansioista saatavista elementeistä. Hakemistojen nimeäminen on tehty helpottamaan tietojen käsittelyä pelin ulkopuolelta, mutta näitä nimiä ei käytetä hyväksi elementtejä muodostettaessa.

5. TYÖN TOTEUTUS

Työ toteutettiin omaksi moduuliksi, joka otettiin pelissä käyttöön Spring-kehyksen avulla. Moduulin toteutuksessa pyrittiin monikäyttöisyyteen, jonka vuoksi rajapinnoissa pyrittiin välttämään pelin toteutukseen suoraan liittyviä luokkia tai parametreja. Lisäksi toteutuksesta pyrittiin tekemään mahdollisimman yksinkertainen ja käyttövarma, jotta sen käyttö ja muokkaaminen olisi helppoa.

5.1. Toteutusperiaatteiden valinta

Ennen toteutuksen aloittamista oli päätettävä toteutuksen yleisistä periaatteista. Näihin periaatteisiin kuuluivat toteutuksessa käytettävä arkkitehtuuri, synkronoitavan datan tallennusmuoto sekä käytettävä tiedonvaihtomuoto. Tehdyillä valinnoilla on huomattava vaikutus toteutuksen rakenteeseen ja erilaisten toimintojen toteuttamiseen.

5.1.1. Käytettävä arkkitehtuuri

Käytettäväksi arkkitehtuuriksi valittiin asiakas-palvelin-malli. Suurimpana syynä valinnalle oli verkon epävarmuus toimintaympäristössä, jonka vuoksi sovellusten välistä jatkuvaa kommunikaatioita ei voida varmistaa. Valitussa arkkitehtuurissa voidaan palvelin saattaa asiakkaan saataville käyttämällä verkosta riippumattomia ratkaisuja, jolloin verkon puute ei estä sovelluksen käyttöä. Lisäksi palvelinta voidaan käyttää tietomallien säilytyspaikkana, mikä parantaa tietomallien säilyvyyttä. Jotta palvelin voidaan sijoittaa helposti siirrettävään mediaan, käytetään raskas-asiakas mallia ja toteutetaan suurin osa toimintalogiikasta asiakasohjelmiston päässä.

Toisena arkkitehtuurivaihtoehtona ollut vertaisverkkomalli olisi tarjonnut mahdollisuuden sovellusten väliseen kommunikaatioon, vaikka jokin sovelluksista ei olisikaan yhteydessä toisiin. Tämän seurauksena olisi tietomalli kuitenkin helposti hajaantunut ja sovellusten näkemät tiedot poikkeaisivat toisistaan suuresti. Lisäksi mallissa olisi ollut haasteellista korvata sovellusten välinen verkkoyhteys jollakin toisella keinolla.

5.1.2. Synkronoitavan datan tallennusmuoto

Kappaleessa 3.3.2 todettiin synkronoitavan datan tallennusmuodon vaikuttavan mahdollisiin synkronointitapoihin ja suorituskykyyn. Tässä toteutuksessa tallennusmuoto vaikuttaa datan saatavuuteen, muuttuneen datan havaitsemiseen, synkronoitavan datan yhdistämisen helppouteen sekä suorituskykyyn. Edellä mainituista

asioista kolme viimeistä liittyvät yleiseen synkronointiin, kun taas ensimmäinen, datan saatavuus, liittyy synkronoitavan pelin vaatimuksiin.

Käytettäväksi tallennusmuodoksi valittiin tiedostorakenne. Tämän valinnan suurin etu oli mahdollisuus muokata ja tarkastella synkronoitavia tiedostoja myös ilman erillistä ohjelmistoa, mikä oli pelin tietorakenteen vaatimuksena. Siksi synkronoitavan datan rakenteena voitiin käyttää täsmälleen samaa rakennetta ohjelmiston oman tallennusmuodon kanssa, mikä poistaa datan muokkaamistarpeen synkronointi-operaation yhteydessä. Tiedostorakennetta on helppo käsitellä käyttöjärjestelmien omilla tiedosto-operaatioilla, mikä yksinkertaistaa toteutusta ja näin vähentää synkronoitaessa tapahtuvia mahdollisia virheitä. Koska ohjelmiston luonteeseen kuuluu vanhojen tietojen muuttumattomuus ja ohjelmiston tietomallin rakentuminen erillisiin tiedostoihin, ei synkronointioperaatiossa useinkaan ole tarvetta muuttaa kuin yksittäisiä tiedostoja. Tästä syystä tiedostorakenne tarjoaa riittävän suorituskyvyn synkronointiin. Suurimpana ongelmana tiedostorakenteessa on sen avoimuus, jonka seurauksena käyttäjällä on pelin dataa ulkokäsin tarkastellessaan mahdollisuus vahingossa sekoittaa synkronointipaikan rakenne.

Toisena mahdollisena vaihtoehtona synkronoitavan datan tallennusmuodoksi oli pakattu tiedostorakenne. Etuina valittuun rakenteeseen nähden pakattu tiedostorakenne vie vähemmän levytilaa ja se hankaloittaa synkronoitavan datan vahingossa tapahtuvaa sekoittamista käyttäjän toimesta. Haittoina pakatussa tiedostorakenteessa on pakkauksen ja purkamisen tuoma lisäkerros, joka vie prosessointiaikaa ja on mahdollisesti epäonnistuva operaatio. Lisäksi yksittäisen muutoksen vuoksi joudutaan koko pakattu järjestelmä päivittämään, toisin kuin pakkaamattomassa tiedostorakenteessa.

Kolmantena vaihtoehtona synkronoitavan datan tallennusmuodoksi oli tietokanta. Suurimpana etuna kahteen muuhun synkronointipaikkaan nähden oli muutosten hallinnan toteuttaminen; useimmat valmiit tietokannat tarjoavat toteutuksen transaktioiden hallintaa. Muutokset tietokantaan voidaan lähettää tietokannan ohjaussanomina, jolloin tietoa ei tarvitse siirtää suuria määriä. Ongelmina tietokannan käytössä ovat hankaluus tarkastella tietoja ohjelmiston ulkopuolelta, tarve muuttaa tiedot ohjelmiston tallennusformaatista tietokantamuotoiseksi sekä tietokantapalvelimen tarve.

5.1.3. Tiedonvaihto

Välitettävä data voi muodostua sanomista, tiedostoista tai pakatuista tiedostoista. Käytettäessä sanomia pysyisivät siirrettävät datamäärät suhteellisen pieninä. Tämä nopeuttaisi osaltaan synkronointitapahtumaa. Sanomien käyttö vaatii kuitenkin synkronoitavan datan sijoittamista palvelimelle, joka osaa tulkita sanomia ja muuttaa datan koostumusta niiden perusteella. Lisäksi sanomien välitykseen tarvitaan usein tietoliikenneportteja, joiden käyttö voi olla mahdotonta ympäristötekijöiden vuoksi.

Tiedostoja käytettäessä joudutaan koko tiedosto lähettämään uudelleen, mikäli sen sisältö on muuttunut. Tämän vuoksi voivat siirrettävät tietomäärät kasvaa suurestikin verrattuna sanomilla tapahtuvaan datan välitykseen. Synkronoitavan

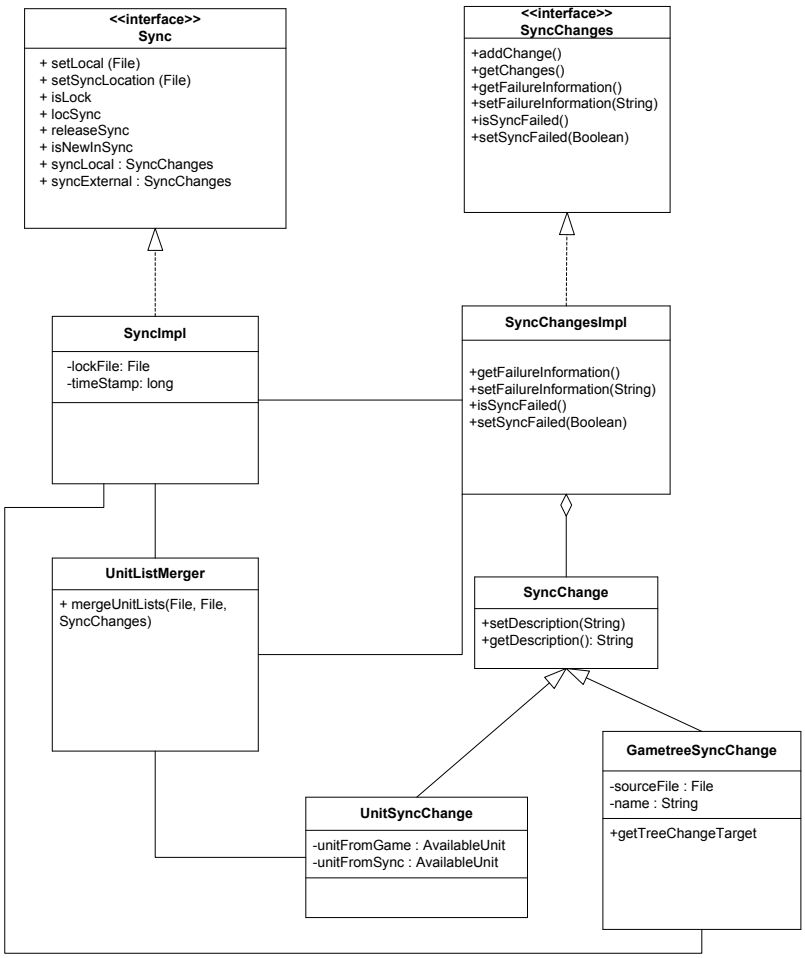
ohjelmiston tietomalli on kuitenkin rakenteeltaan sellainen, että suurin osa tiedostoista säilyy saman sisältöisenä koko pelin ajan. Tiedostoja voidaan lähettää useamman eri protokollan välillä ja niiden käsittelyyn on käyttöjärjestelmissä sisäänrakennettu tuki.

Datan koostuessa pakatuista tiedostoista ei siirrettävää dataa ole niin paljon kuin käsiteltäessä tiedostoja, mutta kuitenkin yleensä enemmän kuin sanomia käytettäessä. Koska synkronoitava data koostuu xml- ja tekstitiedostoista, on sen pakkautuvuus hyvä. Datan pakkaaminen ja purkaminen vaativat kuitenkin prosessointiaikaa sekä lisäävät mahdollisesti epäonnistuvan vaiheen synkronoinnissa.

Välitettävän datan muodoksi synkronoitaessa valittiin tiedostot, koska niitä voidaan käsitellä ilman erillistä palvelinta, muuttuvien tiedostojen määrä pysyy ohjelmiston normaalikäytössä maltillisena, synkronointioperaation kesto ei ole ongelmallinen ja tiedostojen käsittely on yksinkertaista. Tiedostoja voidaan välittää käyttäen käyttöjärjestelmän sisäänrakennettua tiedoston käsittelyä tai verkkoprotokollien avulla. Koska verkkoprotokollien käyttö saattaa olla estetty käyttöympäristön vaikutusten vuoksi, valittiin käyttöjärjestelmän sisäänrakennettu tiedostojen käsittely synkronoitavien tietojen välitystavaksi.

5.2. Toteutuksen rakenne

Toteutuksena lähtökohtana oli tehdä moduuli, joka vastaa pelin vaatimiin synkronointitarpeisiin, ja jonka rakenteita voidaan käyttää uudelleen myöhemmin vastaavien sovellusten yhteydessä vähällä vaivalla. Koska moduuli otettiin käyttöön käyttäen Spring-ohjelmistokehystä, muodostui rajapintaluokka ensimmäiseksi selkeäksi osaksi rakennetta. Tämän rajapinnan toteuttavan luokan suunnittelussa päätettiin tehdä yksi, sovelluksen tarpeisiin räätälöity, luokka. Toteuttava luokka olisi ollut mahdollista suunnitella täysin yleiskäyttöiseksi, mutta tällöin siihen olisi liittynyt suuri joukko erillisen, sovelluskohtaisen, toteutuksen vaatimia luokkia. Tämä ratkaisu ei suuresti lisännyt luokan uudelleenkäyttöarvoa, mutta olisi silti lisännyt toteutuksen monimutkaisuutta.



Kuva 5-1. Synkronointimoduulin luokkakaavio

Toteutuksen luokkakaavio on esitetty yllä olevassa kuvassa (Kuva 5-1). Toteutus jakautuu seuraaviin osiin: rajapinta, synkronointilogiikka, synkronoitavien tietojen yhdistämislogiikka ja synkronointitapahtuman muutokset. Rajapinnan muodostaa luokka *Sync*, synkronointilogiikan luokka *SyncImpl*, yhdistämislogiikan luokka *UnitListMerger*. Synkronointitapahtuman muutoksista vastaavat *SyncChanges*-rajapinnan toteuttava luokka *SyncChangesImpl* sekä *SyncChange*-luokasta laajennetut luokat *UnitSyncChange* ja *GametreeSyncChange.* Synkronointilogiikka toteuttaa rajapinnan ja tuottaa muutostapahtumia. Tarvittaessa synkronointilogiikka käyttää myös tietojen yhdistämislogiikkaa, joka tuottaa tehdyistä muutoksista joukkolistan muutostapahtumia.

5.3. Transaktioiden toteutus

Transaktiot toteutettiin ensin kopioimalla muokattava tiedostorakenne, tekemällä synkronointi kopioituun rakenteeseen, varmuuskopioimalla alkuperäinen rakenne, tuhoamalla alkuperäinen rakenne, kopioimalla muokattu rakenne alkuperäisen rakenteen tilalle ja tuhoamalla muokatun rakenteen alkuperäinen versio sekä alkuperäisen rakenteen varmuuskopio. Ratkaisun etuna on sen yksinkertaisuus ja tästä aiheutuva

onnistumisvarmuus. Ratkaisussa joudutaan tekemään paljon levyoperaatioita ja tämä näkyykin hitautena erityisesti käytettäessä hitaita levyjärjestelmiä. Operaation kokonaiskesto ei kuitenkaan muodostanut käyttöä haittaavaksi tekijäksi, minkä vuoksi pitäydyttiin toteutetussa käyttötavassa.

Vaihtoehtoisena toteutuksena muokatun ja alkuperäisen rakenteen kopioinnin ja poistamisen tilalle kokeiltiin uudelleennimeämistä. Ratkaisu olisi vähentänyt levyoperaatioita ja näin nopeuttanut transaktion toteutusta. Käytettäessä tätä ratkaisua havaittiin kuitenkin uudelleennimeämisen epäonnistuvan satunnaisesti. Tämä epäonnistuminen aiheutti tarpeetonta epävarmuutta synkronoinnin onnistumisesta, jonka vuoksi toteutuksen käytöstä luovuttiin.

Transaktion osa-alueet

Atomisuus on toteutettu säilyttämällä alkuperäinen versio tiedostorakenteesta synkronoinnin aikana. Jos synkronointioperaatio ei onnistu, tuhotaan yksinkertaisesti tiedostorakenteen kopio, johon muutokset kohdistuivat. Jos ongelma ilmenee muutettua tiedostorakennetta käyttöönotettaessa, eli varmuuskopioitaessa vanhaa rakennetta, tuhottaessa vanhaa rakennetta tai kopioitaessa muutettua rakennetta alkuperäisen rakenteen tilalle, otetaan käyttöön alkuperäinen rakenne, joko varmuuskopioista, tai jättämällä alkuperäinen rakenne paikalle. Epäonnistumisen jälkeen tuhotaan kaikki rakenteet alkuperäistä lukuun ottamatta.

Eheys transaktioiden jälkeen saavutetaan perustilanteissa sillä, että käsiteltävien tiedostojen sisältöä ei muokata ja niiden suhteellinen sijainti pidetään samana. Erikoistilanteen eheyden suhteen muodostaa joukkolistan yhdistäminen ja siitä johtuvat mahdolliset sivuvaikutukset. Yhdistämisoperaatiossa yhdistettävät tiedostot muutetaan objekteiksi, objektien tiedot yhdistetään käyttäen objektin tukemia operaatioita ja lopuksi tämä yhdistetty objekti muutetaan takaisin tiedostoksi.

Eristyvyys toteutuu lukkotiedoston avulla. Ennen synkronointioperaatioita luodaan tilapäistiedosto ja sille asetetaan lukko. Kun synkronointi on suoritettu, poistetaan lukko ja tuhotaan tilapäistiedosto. Jos toinen peli-instanssi pyrkii synkronoimaan samaa palvelinta, ei se pysty luomaan uutta lukkotiedostoa ja jatkamaan synkronointi-operaatiota ennen kuin ensimmäinen synkronointi on saatettu loppuun ja lukko vapautettu. Eristyvyys synkronoinnissa rajoittuu synkronointimoduulien eristämiseen. Asetettu lukko ei estä palvelimen käsittelyä ja muokkaamista muun ohjelmiston avulla.

Pysyvyys synkronointioperaatiossa toteutuu ilman erillistä käsittelyä, koska synkronointi perustuu levyoperaatioihin. Tehdyt operaatiot pysyvät tämän vuoksi voimassa vaikka ohjelmisto sammutetaan tai se kaatuu.

5.4. Tietojen yhdistäminen

Synkronoitavat tiedot ovat hajautuneet useisiin yksittäisiin tiedostoihin. Pääsääntöisesti tietojen yhdistäminen merkitsee sitä, että tiedostorakenteet saatetaan sisältämään samat tiedostot:

- Jos toisesta synkronoitavasta tiedostorakenteesta puuttuu tiedosto, lisätään tämä tiedosto.
- Jos molemmat tiedostorakenteet sisältävät saman tiedoston, tarkistetaan tiedostojen eroavaisuus MD5-tarkistussumman avulla. Jos tarkistussummat täsmäävät, ei kyseisen tiedoston kohdalla ole tarvetta synkronointiin. Tarkistussummien ollessa eriävät, mukaan otettava tiedosto riippuu sovelluksen moodista: sovelluksen ollessa tuomari-moodissa käytetään sovelluksen tiedostoa ja sovelluksen ollessa pelaaja-moodissa käytetään palvelimen tiedostoa.

Joukkotietojen yhdistäminen

Poikkeuksena tähän on pelin joukkolistan sisältävä tiedosto, jonka tapauksessa myös tiedostojen sisällöt yhdistetään. Tämän tiedoston käsittely aloitetaan samoin kuin kaikkien muidenkin tiedostojen käsittely, mutta tilanteessa, jossa tarkistussummat eivät vastaa toisiaan, toteutetaan joukkolistojen yhdistäminen. Yhdistämisen hoitaa *UnitListMerger*-luokka, jolle annetaan joukkolistojen lähde- ja kohdetiedostot. Kun peliä käytetään tuomari-roolissa, toimii lähdetiedostona palvelimen tiedosto ja kohdetiedostona sovelluksen tiedosto. Muulloin lähde- ja kohdetiedostot asetetaan päinvastoin. Luokka tekee saamistaan tiedostoista joukkolistaobjektit, joita sitten käsitellään. Lähdetiedostosta muodostetun joukkolistan joukkojen yksilöllistä Universally Unique Identifier (UUID) -tunnistetta verrataan kohteen joukkojen tunnisteisiin. Mikäli tunniste löytyy, jatketaan seuraavan joukon käsittelyllä. Jos tunnistetta taas ei löydy, lisätään joukko myöhemmin käsiteltäväksi. Kun kaikki lähteen joukot on näin käyty läpi, tehdään vastaava operaatio kohteen joukoille. Löydettyjä, uusia joukkoja, verrataan keskenään ja tunnistetaan joukkojen kaksoiskappaleet. Tämän jälkeen lähteen uudet, ei kaksoiskappaleiksi tunnistetut, joukot liitetään kohteeseen. Tunnistetuista uusista kohteista muodostetaan muutosobjektit, jotka välitetään synkronointioperaation paluuarvon osana kutsuvalle ohjelmistolle. Lopuksi kohde- ja lähdetiedostot päivitetään kirjoittamalla muokattu kohteen joukkolista vanhojen tiedostojen päälle.

Pelin käyttämä tiedostorakenne saattaa sisältää tiedostoja, joiden synkronointi ei ole pelin toimintalogiikan kannalta toivottavaa. Nämä tiedostot tunnistetaan joko niiden sijaintikansion *tmp*-päätteestä, tai xml-tiedoston elementin *published*-arvosta.

5.5. Ristiriitojen käsittely

Synkronoitavat tiedot on sijoitettu erillisiin tiedostoihin, jolloin myös mahdolliset konfliktit kohdistuvat yksittäisiin tiedostoihin. Mahdollisia konfliktitilanteita ovat kahden käyttäjän yhtäaikainen uuden tiedoston lisäys ja yhtäaikainen tiedoston

muuttuminen. Tiedoston muuttuminen voi aiheutua tuomarin tekemästä muutoksesta peliin kuuluvaan tekstiasiakirjaan tai pelin joukkolistan muokkaaminen jonkun käyttäjän toimesta. Tiedostojen poistaminen ei ole pelin toimintalogiikalla mahdollista, joten se ei voi aiheuttaa konfliktitilannetta.

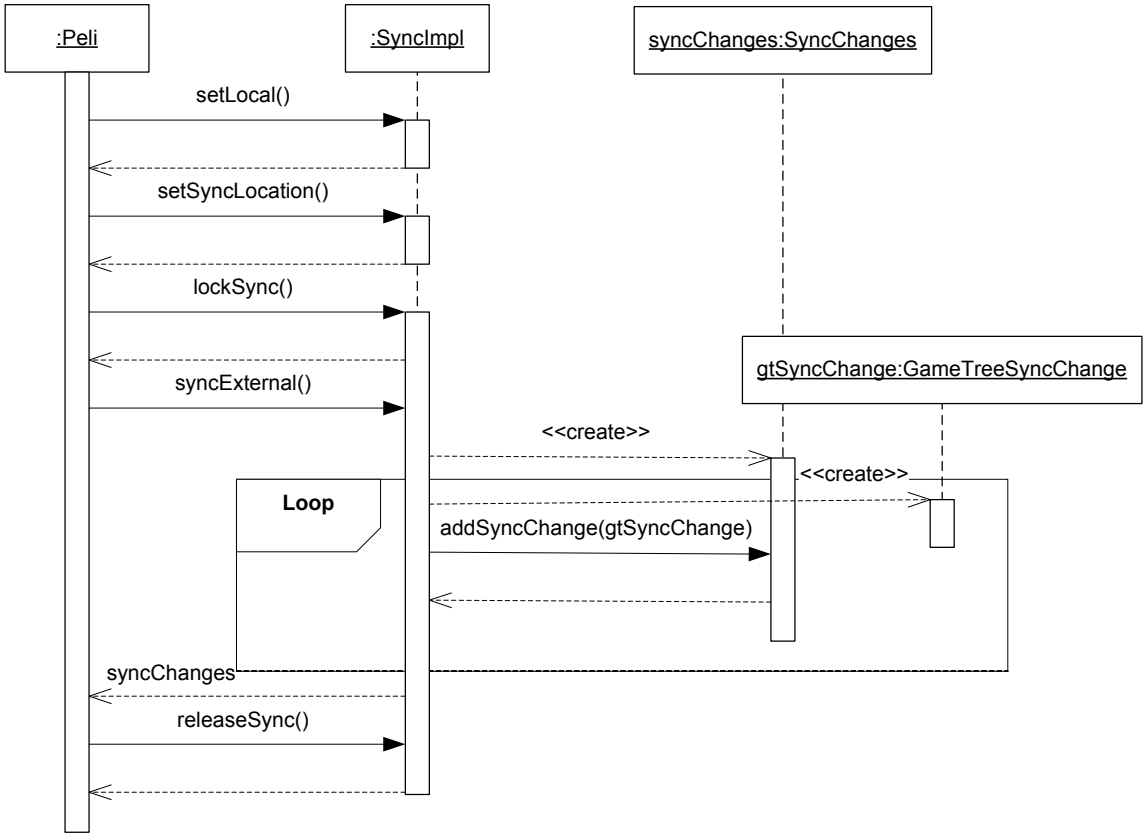
Mikäli kaksi käyttäjää on ohjelmistoa käyttäessään luonut samaksi tiedostoksi kuvautuvan tilanteen, moduuli suosii ensimmäisenä palvelimelle saatettua tiedostoa. Käyttäjä, joka myöhemmin pyrkii synkronoimaan omaa versiotaan tiedostosta, saa ilmoituksen olemassa olevasta päällekkäisyydestä. Samalla käyttäjältä kysytään jatketaanko synkronointia. Käyttäjän halutessa jatkaa synkronointia, korvataan käyttäjän versio tiedostosta palvelimella olevalla tiedostolla. Jos käyttäjä keskeyttää synkronoinnin, ei sovellus saa mitään palvelimen sisältämistä uusista tiedostoista. Mikäli käyttäjä haluaa saada kyseisen palvelimen tiedot, tulee hänen sallia omien tiedostojensa päällekirjoitus. Ennen tämän toteuttamista on käyttäjän mahdollista tallentaa kopio olemassa olevasta tilanteestaan tai tehdä pelin sisällä kopio ristiriidassa olevista tiedoista toiseen paikkaan.

Tilanteessa, jossa palvelimen ja pelin tiedostot eroavat toisistaan ilman juuri luotua tiedostoa ja kyseessä ei ole joukkotietoja sisältävä tiedosto, otetaan synkronointipaikan tiedosto käyttöön. Jos tiedostot sisältävät joukkotietoja, toimitaan luvussa 5.4 kuvatulla tavalla. Poikkeuksen käytäntöön tekee tapaus, jossa peli on tuomari-moodissa, jolloin käytetään pelin tiedostoa. Tällä toimintatavalla saadaan palvelimen tiedostoja muutettua, mutta niiden muuttuminen tapahtuu hallitusti yhden moodin avulla.

5.6. Käyttöskenaarioita

Synkronoitavan pelin saattaminen palvelimelle

Kun peli on vasta aloitettu, ei siihen liittyvää dataa ole saatavilla palvelimelta. Ensimmäinen synkronointi vastaakin tämän vuoksi suuresti peilaamista, jossa tiedon siirto tapahtuu vain yhdensuuntaisena. Sekvenssikaavio tapahtumasta, kuvattuna synkronointimoduulin näkökulmasta, on esitetty alla olevassa kuvassa (Kuva 5-2).

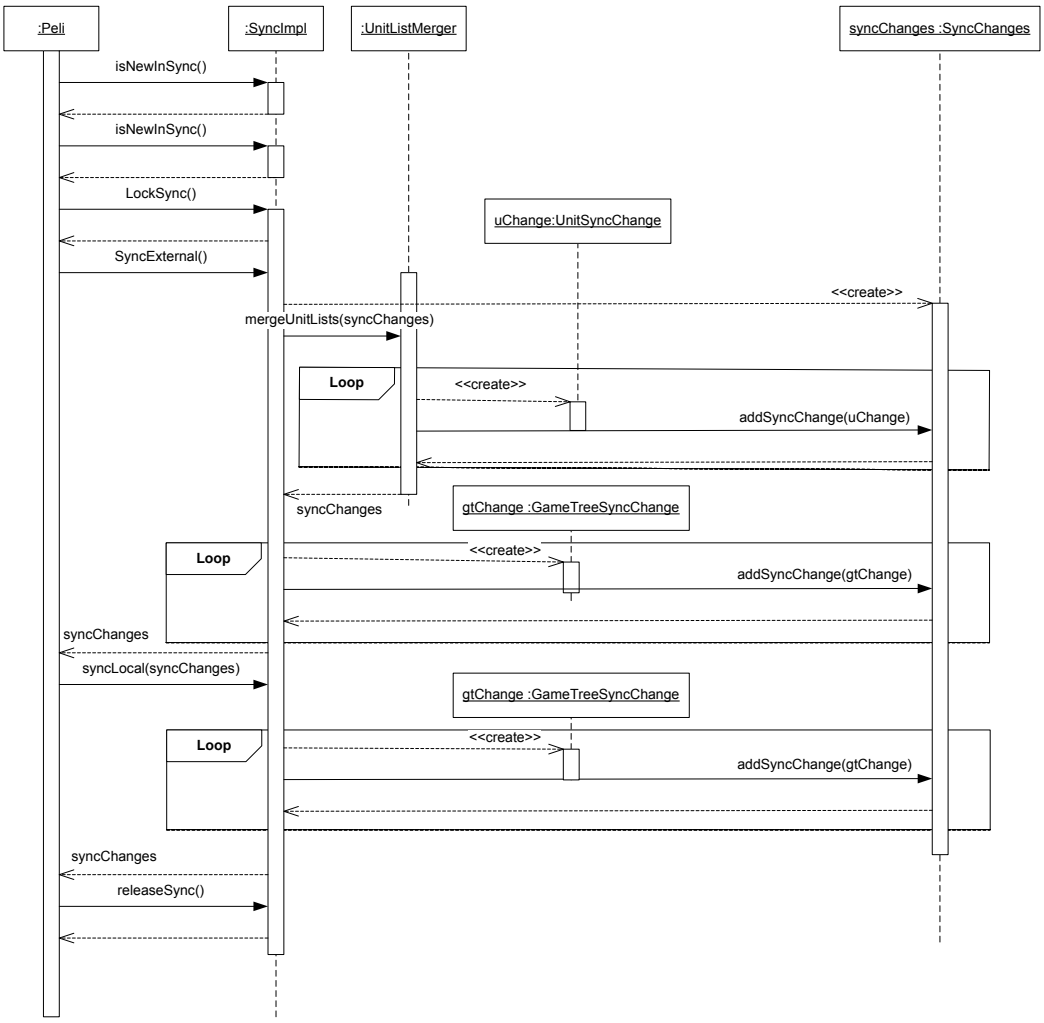


Kuva 5-2. Synkronointimoduulin ja palvelimen alustuksen sekvenssikaavio.

Sekvenssikaavio on laadittu kuvaamaan normaalia käyttäytymistä, jossa kaikki kutsut palauttavat hyväksyttävät vastaukset ja synkronointi voi edetä suunnitellusti. Kaaviosta on myös poistettu funktioiden parametreja ja paluuarvoja selkeyden saavuttamiseksi. Ensimmäisessä vaiheessa peli asettaa synkronointimoduulin tarvitsemat tiedot pelin sisäisen datan ja palvelimen sijainnista. Seuraavaksi peli pyytää synkronointimoduulia lukitsemaan palvelimen omaan käyttöönsä. Kun palvelin on lukittu moduulille, peli pyytää synkronointimoduulia siirtämään sisäisen datan palvelimelle (*syncExternal()*). Ennen varsinaista siirto-operaatioita synkronointimoduuli luo *SyncChanges*-tyyppisen olion, johon kerätään synkronoinnin aikana tapahtuvat muutokset. Varsinaisessa synkronoinnissa käydään läpi kaikki pelin sisäisen datan muodostaman hakemistorakenteen tiedostot ja kopioidaan ne palvelimelle vastaavaan hakemistorakenteeseen. Kopioitua tiedostoa kohden luodaan *GameTreeSyncChange*-tyyppinen olio, joka kuvaa synkronoinnissa tapahtunutta muutosta, tässä tapauksessa siis tiedoston lisäystä palvelimelle. Nämä luodut oliot annetaan säilytettäväksi *syncChanges-oliolle*. Kun kaikki tiedostot on kopioitu palvelimelle, palautetaan muutokset sisältävä *syncChanges* kutsuvalle pelille. Pelin vastuulle jää esittää tarvittaessa muutokset käyttäjälle. Lopuksi peli antaa moduulille luvan vapauttaa palvelimen lukitus.

Pelin synkronointi palvelimen kanssa

Useimmin synkronointi suoritetaan tilanteessa, jossa peliä on jo pelattu ja palvelin sisältää hieman eriävää tietoa pelaajan tietoihin verrattuna. Tapahtuman sekvenssikaavio on esitetty alla olevassa kuvassa (Kuva 5-3).



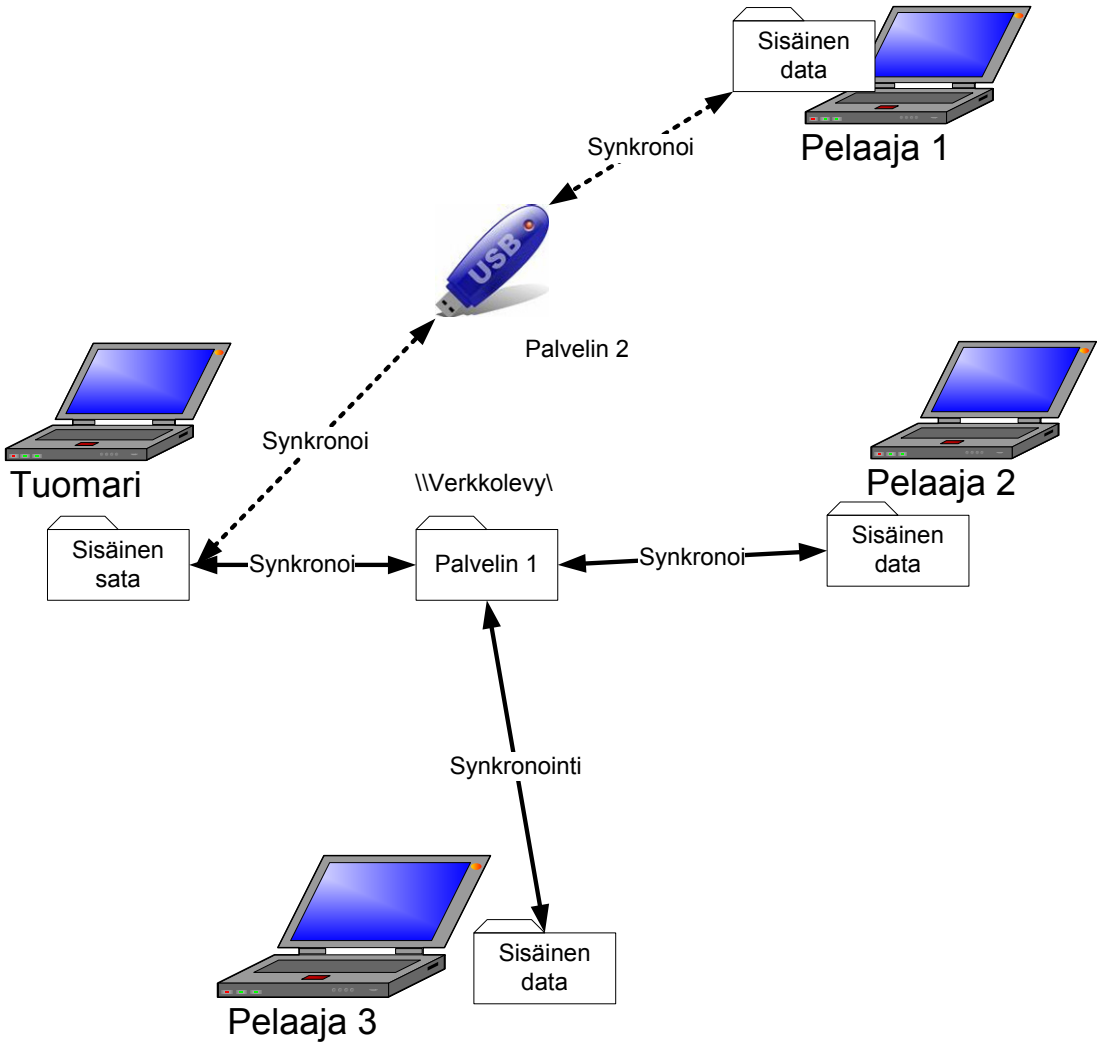
Kuva 5-3. Synkronoinnin perustapauksen sekvenssikaavio

Tiedon eriävästä sisällöstä peli saa kutsumalla synkronointimoduulin *isNewInSync*-metodia. Kun peli kutsun paluuarvosta havaitsee uutta dataa olevan saatavilla, pyytää peli moduulia lukitsemaan palvelimen itselleen. Kutsun seurauksena synkronointimoduuli luo *syncChanges*-olion, jonne kerätään tiedot synkronointi-tapahtumasta. Seuraavaksi synkronointimoduuli yhdistää pelin ja palvelimen joukkolistat *UnitListMerger*-luokan avulla. Jokaisesta joukkolistan muutoksesta tehdään *UnitSyncChange*-olio, joka annetaan *synchChanges*:n säilytettäväksi. Joukkolistojen käsittelyn jälkeen synkronointimoduuli lähettää pelin sisältämät uudet tiedot palvelimelle. Muokattua/lisättyä tietoa kohden luodaan *GametreeSyncChange*-olio, joka annetaan *syncChanges*:n säilytettäväksi. Kun palvelimen tiedot on saatu päivitettyä, palautetaan *syncChanges* kutsuvalle pelille. Seuraavaksi peli kutsuu moduulin *syncLocal*-metodia antaen sille parametriksi edellä saamansa *syncChanges*:n. Tällöin muutosten säilöntään käytetään annettua oliota uuden luomisen sijasta. Moduuli hakee palvelimen tiedot ja asettaa pelin tiedot vastaamaan niitä. Operaation aikana tehtävistä

muutoksista luodaan *GametreeSyncChange*-oliot, jotka annetaan *syncChanges*:n säilytettäväksi. Kun tiedot on saatettu vastaamaan toisiaan, palautetaan pelille muutokset sisältävä *syncChanges*. Saatuaan muutostiedot, antaa peli synkronointimoduulille luvan vapauttaa palvelin muiden instanssien käyttöön. Muutosten esittäminen käyttäjälle jää pelin vastuulle.

Pelien yhteistoiminta

Tuomarin ja usean pelaajan samanaikaisesti pelaama yksittäinen peli on tilanne, jota varten synkronointimoduuli on rakennettu. Lisäksi synkronointimoduulissa on huomioitu, että pelaajat eivät välttämättä voi luottaa verkkoyhteyden toimintaan yhteistoiminnassaan. Alla olevassa kuvassa (Kuva 5-4) on esitetty tilanne, jossa tuomari ja kaksi pelaajaa ovat verkkoyhteydessä toisiinsa ja käyttävät verkkolevyllä olevaa palvelinta synkronointiin. Yhdellä pelaajista ei ole mahdollisuutta verkkolevyllä olevan palvelimen käyttöön yhteysongelmien vuoksi, joten hän käyttää synkronointiin USB-levyllä olevaa palvelinta. Tuomari toimii linkkinä palvelinten välillä.



Kuva 5-4. Pelien sijainti toisiinsa ja palvelimiin nähden yhteiskäyttötilanteessa

Pelin kulku edellä kuvatussa tilanteessa on seuraava:

1. Tuomari luo lähtökohdat pelille ja synkronoi pelin molempien palvelinten kanssa.
2. Pelaajalle 1 toimitetaan palvelimen 2 sisältämä USB-levy ja hän synkronoi pelinsä sen kanssa.
3. Pelaajat 2 ja 3 synkronoivat pelinsä palvelimen 1 kanssa.
4. Pelaajat suunnittelevat toimintansa.
5. Pelaajat synkronoivat pelinsä palvelimen kanssa.
6. Tuomarille toimitetaan palvelimen 2 sisältämä USB-levy ja hän synkronoi pelinsä sen kanssa.
7. Tuomari synkronoi pelinsä palvelimen 1 kanssa.
8. Tuomari määrittelee lopputulokset pelaajien suunnitelmista, luo lähtökohdat pelin jatkamiselle ja synkronoi pelin molemmille palvelimille.
9. Palataan kohtaan 2.

Yllä esitetyssä pelin kulussa on pelaajilla yhtenevät tiedot ainoastaan kohdan 3 jälkeen. Tästä kohdata eteenpäin pelaajat muokkaavat omia tietojaan ja palvelinten sisältämät tiedot eroavat pelaajien tiedoista. Kohta 5 ei vielä tuo edes pelaajien tietoja yhteneviksi, sillä pelaaja 1 käyttää eri palvelinta ja vaikka pelaajat 2 ja 3 käyttävät samaa palvelinta, ei ensimmäisenä näistä synkronoinut saa jälkimmäisen tietoja. Tämä ei ole kuitenkaan pelin toiminnan kannalta ongelma, sillä pelaajilla ei yleensä ole tarvetta saada tietoonsa toisten tekemiä suunnitelmia. Jos tällainen tarve kuitenkin ilmenee, voivat samaa palvelinta käyttävät saattaa tietonsa samaan tilaan tekemällä synkronoinnin uudelleen kun kaikki kyseistä palvelinta käyttävät ovat suorittaneet synkronoinnin.

Esitetyssä pelin kulussa myös palvelinten tiedot ovat epäyhtenevät kohtien 5 ja 8 välillä. Jotta palvelinten tiedot on mahdollista saattaa yhteneviksi, on tärkeää että juuri tuomarilla on yhteys molempiin palvelimiin. Tämä siksi, että tuomari-moodissa oleva peli suosii sisäistä dataa synkronoitaessa. Näin tuomarin tekemä kahden palvelimen peräkkäinen synkronointi saattaa niiden tiedot yhteneviksi.

6. TOTEUTUKSEN ARVIOINTI

6.1. Käyttövarmuus

Käyttövarmuuden arvioimiseksi ei synkronointimoduulille suoritettu erillisiä yksikkötestejä, josta olisi saatu tilastollista dataa käyttövarmuudesta. Moduulin käyttövarmuutta arvioitiin kuitenkin toteutustyön ohessa tuotetuilla yksikkötesteillä, integrointivaiheessa tehdyillä rasitustesteillä sekä normaaleja käyttötilanteita seuraamalla.

Toteutustyön ohessa tehdyillä yksikkötesteillä varmistettiin joukkolistojen yhdistämisen sekä tiedostojen oikea käsittely. Joukkolistojen yhdistämistestauksessa käytettiin vain muutamia joukkoja, joihin oli tehty tarpeellisia muutoksia yhdistämisen onnistumisen havaitsemiseksi. Tiedostojen käsittelyn testauksessa testit suoritettiin käyttämällä muutamaa tiedostoa, pelin tietomallia vastaavan tiedostorakenteen sijasta. Näin toteutetulla testauksella havaittiin nopeasti virheet yhdistämisen ja tiedostojen käsittelyn perustoiminnoissa ja testin onnistuttua oli mielekästä jatkaa integrointitestauksella.

Integrointivaiheen testauksessa synkronointimoduulin toimintaa testattiin käyttämällä pelin tarjoamia rajapintoja. Testauksessa synkronointia suoritettiin normaalitilannetta suuremmilla tietomäärillä ja sekvensseillä sekä vaihtelevilla palvelimilla. Testeissä ajettiin myös useaa sovellusta samalla koneella, jolloin käytettävä synkronointipalvelin sijaitsi samalla kovalevyllä synkronoitavien pelien kanssa. Integraatiovaiheen testien aikana havaittiin luvussa 5.3 mainittu satunnaisesti esiintyvä ongelma ensimmäisen transaktiototeutuksen toiminnassa. Vastaavaa ongelmaa ei saatu uusittua toteutuksen muuttamisen jälkeen. Integrointivaiheen testauksessa voitiin myös selvittää usean sovelluksen yhtäaikaisen palvelimen käytön mahdollisia ongelmia. Näiden testien aikana vain yksi sovellus pystyi vaikuttamaan palvelimen tiedostoihin tietyllä ajanhetkellä, eli synkronointimoduulin toteuttama palvelimen lukitus toimi moitteetta. Sen sijaan muutaman kerran synkronointi ei onnistunut vaikka ympäristössä ei pitänyt olla mitään estettä toiminnalle. Koska virheitä ei kuitenkaan saatu toistettua, ei moduulin tehty muutoksia näiden takia.

Normaalissa käyttötilanteessa seurattiin peliä, johon osallistui tuomari sekä kolme pelaajaa. Synkronointipalvelin oli sijoitettu verkkoon jaettuun kansioon, johon kaikilla käyttäjillä oli käyttöoikeus. Pelin aikana jokainen käyttäjä suoritti noin kymmenen synkronointia, joissa synkronoitavan datan määrä kasvoi jokaisessa synkronoinnissa. Pelin aikana ei havaittu synkronoitavan datan korruptoitumista tai synkronointipalvelimen lukituksen epäonnistumista. Lisäksi synkronointipalvelimella oleva data oli käytettävää myös pelin loputtua.

Eri vaiheessa tehtyjen testausten perusteella on synkronointimoduulin käyttövarmuus hyvä. Normaalissa käytössä ei moduulin käytössä havaittu virheitä ja rasitustestit paljastivat ainoastaan satunnaisia virheitä, joiden syytä ei voitu johtaa moduuliin, ohjelmistoympäristöön tai laitteistoon. Lisäksi toteutuksen tukena käytetyt yksikkötestit lisäsivät varmuutta moduulin oikeasta toiminnasta.

6.2. Suorituskyky

Kuten ei käyttövarmuutta, ei myöskään suorituskykyä mitattu menetelmillä, joista olisi saatavilla tilastollista dataa. Suorituskykyä ei myöskään tarkkailtu missään ennalta määritellyssä ympäristössä, vaan havaintoja suorituskyvystä tehtiin useassa erilaisessa ympäristössä, erilaisilla tietomäärillä ja eri käyttöintensiteeteillä.

Suorituskykyä havainnoitaessa synkronoinnin kestossa havaittiin suuria vaihteluita synkronoitavan tilanteen ja käyttöympäristön muuttuessa. Käyttöympäristön pysyessä samana synkronointiaikaa nosti eniten joukkolistaan kahden käyttäjän toimesta lisätyt suuret joukkomäärät. Seuraavaksi merkittävin hidastava tekijä oli uusien tiedostojen huomattava määrä, mikä yleensä johtui ensimmäisestä synkronoinnista. Muut muutokset synkronoitavissa tiedoissa eivät aiheuttaneet merkittäviä muutoksia synkronoinnin kestoon. Tarkastelussa havaittiin myös synkronoinnin kestävän aina selkeästi havaittavan ajan, vaikka synkronoitavaa ei kyseisellä hetkellä ollut.

Kun synkronoitava tilanne pidettiin samana ja käyttöympäristöä muutettiin, havaittiin synkronointiajan olevan riippuvainen käytettävästä koneesta sekä synkronointipalvelimen sijainnista. Suorituskyky oli suurimmillaan käytettäessä tehokasta konetta, jossa oli nopea kiintolevy, synkronointipalvelimen sijaitessa samalla kiintolevyllä. Kun palvelin sijoitettiin UBS-levylle tai verkon yli käytettävälle kiintolevylle havaittiin suorituskyvyssä selkeä pudotus. Suorituskyvyn heikkeneminen havaittiin myös, kun peli sijaitsi koneella, jossa oli hidas kiintolevy.

Suorituskyvyn tarkastelussa tehtyjen havaintojen perusteella suorituskykyyn vaikuttavat neljä osatekijää: transaktioiden toteuttaminen, joukkolistojen yhdistäminen, synkronointitarpeen päätteleminen ja synkronointipalvelimen sijainti.

Toteutustavasta johtuen muodostuivat transaktiot suurimmaksi yksittäiseksi hidastavaksi osaksi suorituskyvyssä. Transaktioissa tehdään täydelliset kopiot synkronoitavista tiedostorakenteista, joten niihin tarvitaan useita levyoperaatioita. Erityisesti hitailla kiintolevyillä tämä nostaa synkronointioperaation kestoa melkoisesti. Transaktioiden suorituskykyä heikentävää ominaisuutta pyrittiin poistamaan luvussa 5.3 mainitulla uudelleennimeämistoiminnolla, mutta siitä luovuttiin käyttövarmuuden ylläpitämiseksi.

Joukkolistojen yhdistämisessä kahden toisistaan eroavan xml-tiedoston elementit yhdistetään. Elementit eivät ole tiedostoissa missään vakioidussa järjestyksessä, jolloin muuttuneet ja lisätyt elementit tulee tunnistaa yksitellen. Jos molemmat tiedostot sisältävät useita lisättyjä elementtejä, kestää niiden yhdistämisessä

kymmeniä sekunteja normaalitehoisella koneella. Normaalissa pelitilanteessa tapaus on kuitenkin harvinainen, eikä se tapahdu useita kertoja saman pelin aikana. Tämän vuoksi ei erillistä optimointia joukkolistojen yhdistämiseen tehty.

Tiedostojen synkronointitarve päätellään laskemalla samannimisten tiedostojen MD5-tarkistussumma. Koska summa lasketaan jokaiselle tiedostolle erikseen synkronoinnin yhteydessä, vie tämä prosessointiaikaa jokaisella kerralla erikseen. Pelin normaalissa käytössä tämä aika ei ole merkittävä, mutta pelin sisältäessä suuren joukon tiedostoja, nostaa tämä toiminto synkronoinnin kokonaiskestoa.

Synkronointipalvelimen ja pelin keskinäinen sijainti vaikuttaa suorituskykyyn lähinnä tiedostojen siirron osalta. Koska kaikki muuttuneet ja uudet tiedostot siirretään kokonaisuudessaan synkronoinnin yhteydessä, heikentää hidas tietoyhteys suorituskykyä. Erityisesti tämä näkyy ensimmäisissä synkronoinneissa, jolloin kaikki tiedostot ovat uusia ja ne tulee kopioida palvelimelta peliin. Tiedon siirtonopeuden lisäksi palvelimen sijainti vaikuttaa tiedostojen luku- ja kirjoitusnopeuksiin. Tämän vuoksi palvelimen sijainti vaikuttaa paitsi tiedostojen siirtonopeuteen myös MD5-tarkistussumman laskentanopeuteen.

6.3. Käytettävyys

Synkronointimoduulin käytettävyys voidaan jakaa käytettävyyteen käyttävän ohjelmiston osalta sekä moduulin käytettävyyteen käyttäjän kannalta osana ohjelmistoa. Näistä molemmat ovat olennaisia moduulin käyttöä varten.

Käyttävän ohjelmiston osalta synkronointimoduulin käyttäminen on suoraviivaista. Moduulissa on ainoastaan 8 julkista rajapintafunktiota, joiden avulla moduulia käytetään. Moduulin käyttöönotossa pitää ensin kutsua pelin ja palvelimen sijainnin asettavia funktioita. Näiden jakaminen erillisiksi funktioiksi heikentää osaltaan käytettävyyttä, sillä kutsuvan ohjelman pitää asettaa molemmat erillisellä kutsulla. Valinta kuitenkin myös lisää käytettävyyttä, koska toisen sijainnin vaihtuessa, ei molempia tarvitse asettaa uudelleen. Synkronoinnin toteuttaminen moduulin avulla vaatii kutsuvalta ohjelmalta usean peräkkäisen kutsun. Nämä ovat palvelimen lukitseminen, synkronointi palvelimelle ja/tai palvelimelta ja lukituksen poisto. Nämä funktiot olisi voitu yhdistää toisiinsa, jolloin kutsuvan ohjelman ei tarvitsisi muistaa lukituksen käsittelyä synkronoinnin yhteydessä. Funktioiden erottaminen toisistaan antaa kuitenkin kutsuvalle ohjelmistolle mahdollisuuden pitää palvelin lukittuna haluamansa ajan, ja mahdollisesti näin ajaa useita synkronointeja peräkkäin. Moduulin käytettävyys kutsuvan ohjelmiston kannalta on kaikkiaan yksinkertaista ja antaa mahdollisuuksia vaihtelevaan käyttöön, mutta samalla kutsuvan ohjelmiston tulee muistaa kutsua useaa metodia toimintojen suorittamiseksi.

Moduulin käytettävyyteen osana muuta ohjelmistoa vaikuttaa luonnollisesti käyttävän ohjelmiston toteutus. Synkronoinnin aikana tapahtuvista muutoksista moduuli tekee yhdistelmän, joka välitetään kutsuvalle ohjelmalle. Tällä ohjelmalla on siten mahdollisuus näyttää käyttäjälle tapahtuneet muutokset sopivaksi katsomallaan tavalla.

Moduulia voidaan käyttää siten, että vain harvoissa tilanteissa käyttäjältä vaaditaan reagointia synkronointitapahtuman jatkamiseksi. Ainoa tilanne, jossa moduuli vaatii käyttäjän huomioita, liittyy tilanteeseen, jossa käyttäjän tekemät muutokset ovat korvautumassa palvelimella olevilla tiedoilla. Muissa tapauksissa synkronointi ei vaadi käyttäjän toimenpiteitä, mikä parantaa käytettävyyttä huomattavasti.

6.4. Kehitysmahdollisuudet

Moduulin kehittäminen tehtiin varsin nopealla tahdilla, minkä seurauksena monissa kohdissa valittiin käyttövarmoja ja yksinkertaisia ratkaisuja. Näiden valintojen muuttaminen tehokkaammiksi vaatii monimutkaisempia ratkaisuja ja siten ratkaisujen syvällistä testausta. Lisäksi moduulissa havaittiin mahdollisuuksia toiminnallisuuden laajentamiseen ja joidenkin toimintojen kehittämiseen.

Uudesta datasta ilmoittamisen automatisointi

Saadakseen tiedon palvelimella olevasta muuttuneesta tai uudesta tiedosta tulee moduulia käyttävän ohjelmiston säännöllisesti tiedustella moduulilta mahdollisia muutoksia. Tämän tyyppisten turhien tarkistelujen välttämiseksi tulisi moduulin olla tarkkailtavissa tarkkailija-mallin mukaisena objektina. Helpoiten tämä on toteutettavissa niin, että moduuli itsenäisesti tarkkailee säännöllisesti palvelimella olevan datan tietoja. Tämä ratkaisu kuitenkin ainoastaan siirtäisi turhien tarkastelujen sijaintia, eikä poistaisi niitä. Tulevassa Javan kehitysversiossa 7 on esitelty metodit tiedostojen muutosten tarkkailuun, joita hyödyntämällä voitaisiin resursseja vievä tiedostojen tarkkailu poistaa (Heiss & Zakhour 2009). Nämä metodit hyödyntävät mahdollisuuksien mukaan tiedostojärjestelmien omia metodeja, jolloin käyttö on tehokkaampaa kuin tiedostojen tietojen jatkuva kysely.

Suodatus ilmoituksille uudesta datasta

Moduuli ei tunnista muuttuneen datan lähdettä, jonka vuoksi se ilmoittaa aina havaitessaan muutoksen datassa. Koska peli sisältää tietoja jotka ovat vain toisen pelaajan käsiteltävissä, voi käyttäjä saada ilmoituksen muuttuneesta datasta joka ei kuitenkaan tule näkyviin synkronoinnin yhteydessä. Tämä lisää synkronointi-tapahtumien määrää ja pitää palvelimen lukittuna turhaan varsinkin sellaisissa tapauksissa, joissa pelaajia on useampi kuin yksi puolta kohden. Moduulia olisi mahdollista laajentaa siten, että se tietää käyttäjän roolin pelissä ja ilmoittaa ainoastaan roolissa näkyville saatavista muutoksista.

Tilannetietojen yhdistäminen

Pelin versio, jota varten synkronointimoduuli tehtiin, ei tukenut usean käyttäjän yhtäaikaista tilanteiden muokkaamista tai luomista. Myöhempää käyttöä varten olisi kuitenkin hyödyllistä mahdollistaa kaikkien tilannetiedostojen yhdistäminen nykyisen joukkolistojen yhdistämisen lisäksi. Tällaisen toiminnallisuuden lisääminen vaatii

mahdollisesti lisää interaktiota käyttäjän kanssa, jotta tilanteeseen liittyvien elementtien oikeat versiot tulevat valituiksi. Oikeat, mukaan otettavat, versiot elementeistä voitaisiin myös päätellä muiden tilanteiden avulla, mutta näin tehtyjen valintojen oikeellisuudesta ei voitaisi olla täysin varmoja.

Synkronoitavan datan muoto

Synkronoitavan datan muodoksi valittiin pelin käyttämä tiedostomuoto. Vaikka tällä valinnalla saavutettiin riittävä tehokkuus synkronointiin, on siinä nähtävissä selviä kehitysmahdollisuuksia. Yksinkertainen kehitystapa on lisätä synkronoitavien tiedostojen rinnalle listaus tiedostoista versioineen ja tarkistussummineen. Listoja vertaamalla saadaan tehokkaammin selville muuttuneet ja synkronoimista vaativat tiedostot. Samalla pystytään vähentämään turhaa tiedostojen siirtämistä tarkistussummien laskentaa varten. Listojen käyttäminen mahdollistaisi myös palvelimen tietojen laajemman päivittämisen muidenkin kuin tuomari-moodissa olevien käyttäjien osalta, koska tiedoista on olemassa erillinen historia.

Transaktioiden suorituksen tehostaminen

Moduulissa transaktioiden suoritus on suurin suorituskykyä alentava tekijä. Transaktioissa tehdään suurelle tiedostojoukolle monta kopiointioperaatiota, joiden avulla varmistetaan transaktioiden onnistuminen. Kopiointioperaatiot on toteutettu yksinkertaisella silmukkatoteutuksella, jota optimoimalla pystytään nopeuttamaan käsittelyä. Osa kopiointioperaatioista voidaan korvata uudelleennimeämisellä, mikäli voidaan varmistaa uudelleennimeämisen onnistuminen. Moduulia toteutettaessa tähän varmistukseen ei pystytty, mutta myöhemmin todettiin käyttöjärjestelmän palveluiden olleen uudelleennimeämisen esteenä. Tästä johtuen transaktioiden toteutuksessa on syytä yrittää uudelleennimeämistä uudestaan.

Funktioiden esiehdot

Nykyisessä toteutuksessa synkronoinnin toteuttavat funktiot vaativat synkronointipalvelimen lukitsemisen pyytämistä toteuttavan ohjelman puolesta. Vaikka kyseinen toiminto lisää moduulin käyttömahdollisuuksia, se samalla heikentää käytettävyyttä kutsuvan ohjelmiston näkökulmasta. Esiehto olisi mahdollista poistaa, jos lukitseminen tapahtuisi moduulin toimesta synkronointifunktion yhteydessä. Jotta nykyinen toimintatapa olisi silti mahdollinen, tulee funktioiden huomioida ohjelman asettama lukko, jotta tätä lukkoa ei tarpeettomasti vapauteta funktion päätteeksi.

6.5. Vertailu Diplomacy-pelin synkronointitoteutusten kanssa

Synkronoitavan pelin toteutus oli ensimmäinen laatuaan, joten tehtyä synkronointia ei ole mahdollista vertailla pelin edellisten versioiden kanssa. Peli kuitenkin muistuttaa

yleiseltä toimintalogiikaltaan suuresti alun perin lautapelinä ilmestynyttä Diplomacya, jonka vuoksi vertailu Diplomacyn synkronointitoteutusten kanssa on mielekästä.

Diplomacy on peli, jossa kahdesta seitsemään pelaajaa hallitsee Euroopan suurvaltoja pyrkimyksenään hallita suurinta osaa pelialueesta. Pelin toiminta koostuu vuoroista, jotka jakaantuvat suunnittelu- ja toteutusvaiheeseen. Suunnitteluvaiheessa pelaajat kommunikoivat keskenään ja suunnittelevat vuoron aikana tehtävät siirrot. Suunnitellut siirrot kirjataan paperille ja toteutusvaiheessa kirjatut siirrot toteutetaan mahdollisuuksien mukaan. (Sharp 1979)

Synkronoidussa pelissä pelaajien tekemät suunnitelmat vastaavatkin Diplomacyn suunnitelmia ja tuomarin tekemä tuomarointi ja mahdollinen uuden alkuasetelman luominen vastaavat Diplomacyn kierroksen toteutusvaihetta. Diplomacyn alkuperäinen version on lautapeli, jolloin pelaajien tulee olla fyysisesti paikalla pelaamassa. Peliä voidaan pelata myös postin tai sähköpostin välityksellä, joita helpottamaan on kehitetty erilaisia apukeinoja. Lisäksi pelistä on tehty useita tietokoneversioita, joita voidaan pelata joka yksin, internetissä, lähiverkossa tai samaa tietokonetta käyttäen.

Lautapeli

Diplomacyn alkuperäisessä lautapeliversioissa ei ole varsinaisesti tarvetta synkronoinnille, koska kaikki tapahtumat ovat keskitettyjä. Synkronointina voidaan kuitenkin pitää pelaajien suunnitelmien kokoamista toteuttamisvaihetta varten sekä toteutumisvaiheen tulosten välittymistä takaisin pelaajille. Suurimpana erona toteutettuun pelin synkronointiin on ylimääräisen tilatiedon puute, jonka vuoksi kaikki muutokset tehdään yhteen tietomalliin. Lisäksi pelaajilla ei ole erillistä näkymää tietomalliin, vaan kaikki tarkkailevat samaa mallia, saman näkymän kautta. Tämä synkronointitapa on erittäin luotettava ja epäherkkä virheille, mutta samalla ei ole yhtä joustava kuin toteutettu synkronointimoduuli.

Postipelaaminen

Lautapelin tuomat rajoitteet pelaajien fyysisestä läsnäolosta poistuivat Diplomacyn postin kautta tapahtuvalla pelaamisella (play-by-post). Tässä pelin versiossa pelaajat ilmoittautuvat pelinjohtajalle halukkuudestaan liittyä peliin kirjeitse. Kun tarvittava määrä pelaajia on koossa, lähettää pelinjohtaja osallistujille tiedot muista pelaajista, mitä maata kukin pelaaja hallitsee ja koska ensimmäisen vuoron siirtojen tulee olla toimitettuna pelinjohtajalle. Näiden tietojen pohjalta pelaaja suunnittelee siirtonsa, jotka hän postittaa pelinjohtajalle. Kun pelinjohtaja on saanut kaikkien suunnitelmat, hän toteuttaa mahdolliset siirrot ja postittaa näistä muodostuneen tilanteen pelaajille. Usein tilanteet postitetaan julkaisussa, joka sisältää useamman pelin tapahtumat. (Sharp 1979)

Tämä pelin muoto vastaa synkronoinnin osalta lautapelin ja toteutetun pelin välimuotoa: Muutokset tehdään yhteen tietomalliin, mutta pelaajat voivat halutessaan muodostaa oman tietomallin saamiensa tietojen perusteella. Lisäksi pelinjohtajan

postittamat tilanteet toimivat ylimääräisenä rajapintana pelin tietomalliin. Synkronointi tässä versiossa on huomattavasti hitaampaa, kuin toteutetussa pelissä. Tosin hitauden myötä yhtäaikaisuus ei ole ongelma. Lisäksi postilla lähetetyt suunnitelmat saattavat kadota tai viivästyä kriittisesti. Näistä ongelmista ei pelaaja edes välttämättä saa tietoa toisin kuin toteutetussa synkronointimoduulissa. Koska pelaajat joutuvat muodostamaan kuvan tilanteesta saamiensa tietojen perusteella, on myös tässä vaiheessa mahdollisuus virheelliseen synkronointiin.

Sähköpostipelaaminen

Diplomacya on mahdollista pelata myös sähköpostin välityksellä (play-by-email). Tällöin peli voi vastata postin välityksellä tapahtuvaa pelaamista, mutta pelin muuttuminen sähköiseksi mahdollistaa myös uusia ominaisuuksia. Huomattavin tällainen ominaisuus on tietokonepelinjohtajan käyttö. Mikäli tietokonetta käytetään pelinjohtajana, rekisteröityvät pelaajat pelinjohtajaominaisuudet toteuttavalle palvelimelle lähettämällä määrämuotoisen sähköpostin palvelimen sähköposti-osoitteeseen. Kun peliin on rekisteröitynyt tarvittava määrä pelaajia, lähettää palvelin pelin tiedot pelaajille. Palaajat lähettävät siirtonsa palvelimelle, joka sitten toteuttaa ne yhtäaikaisesti ja ilmoittaa tuloksen pelaajille. Palvelimen avulla pelaajat voivat myös kommunikoida keskenään tuntematta toistensa sähköpostiosoitteita. (Brennan 2006)

Synkronointi tässä pelitavassa voi olla nopeampaa kuin toteutetussa pelissä, mutta sen onnistuminen vaatii toimivan verkkoyhteyden. Mikäli palvelin ei lähetä kuittausta lähetetyistä suunnitelmista, on synkronoinnin onnistuminen epävarmempaa kuin tehdyssä toteutuksessa. Kuittauksen lähettämiseen voidaan usein vaikuttaa yksinkertaisella palvelimen parametroinnilla. Lisäksi, kuten postin välityksellä tapahtuvassa pelaamisessa, joutuvat pelaajat itse muodostamaan kuvan pelin tapahtumista, mikä mahdollistaa virheellisen synkronoinnin.

Pelaaminen internetissä

Diplomacya voidaan pelata internetissä käyttäen tavallisia selaimia. Tämän pelitavan esiversioita olivat sähköpostin avulla tapahtuvien pelien karttakuvien esittäminen selaimella. Jo näissä versioissa monet pelin käskyistä voitiin antaa sähköpostin lisäksi selaimen avulla. Myöhemmissä versioissa ei sähköpostia tarvita lainkaan, vaan kaikki toiminta on mahdollista toteuttaa selaimella. Yksi tällaisen pelaamisen mahdollistava toteutus on webDiplomacy (http://webdiplomacy.net).

Synkronointi tässä pelitavassa on nopeaa ja käyttäjät voivat havaita antamiensa käskyjen onnistumisen suoraan selaimistaan. Verrattuna tehtyyn toteutukseen, koostuu synkronointi lähinnä malliin annettavista muutoskäskyistä. Koska malleja on ainoastaan yksi ja pelaajat näkevät vain siitä muodostetun näkymän, on synkronointi yksinkertaisempaa tehtyyn toteutukseen verrattuna. Haittana tässä pelitavassa on verkkoyhteyden tarve sekä useamman kopion puuttuminen.

Tietokonesovellukset

Diplomacysta on tehty useita yksittäisiä sovelluksia, jotka mahdollistavat pelaamisen tietokonetta vastaan sekä moninpelin. Yksinpeleissä ei synkronointi ole tarpeen ja erillisillä koneilla tapahtuvat moninpelit on toteutettu käyttäen vastaavia metodeja kuin sähköpostilla pelaamisessa tai selaimilla pelattaessa. Uuden näkökulman synkronoinnin osalta muodostaa samalla koneella tapahtuva moninpelaaminen (hotseat). Tässä pelitavassa pelaajat käyttävät peliä lähinnä virtuaalisena pelilautana ja syöttävät omat siirtonsa käyttäen vuorotellen samaa käyttöliittymää. Alkuperäiseen lautapeliin verrattuna tässä on se hyöty, että laittomia siirtoja ei voi esittää. Muutoin synkronointi vastaa lautapelin synkronointia.

7. YHTEENVETO

Työn tavoitteena oli toteuttaa synkronointimoduuli vuoropohjaiselle pelille, jonka toimintaympäristössä ei voida taata verkkoyhteyttä eri pelien välillä. Tällaista datan synkronointia on tutkittu paljon viime aikoina, mutta useimmat ratkaisut käyttävät hyväkseen verkkoa synkronoinnin toteuttamiseksi. Toteutuksia, joissa kahden tietokoneen tiedostot synkronoidaan ulkoisen levyn avulla, on myös saatavilla, mutta näissä ratkaisuissa synkronoitavia järjestelmiä on tarkalleen kaksi. Näitä toteutettuja synkronointiratkaisuja pystyttiin käyttämään pohjana toteutuksen suunnittelulle.

Moduulin toteutuksessa keskityttiin ensisijaisesti synkronoinnin perustoiminnan toimintavarmuuden varmistamiseen. Toteutukseen käytettävän ajan puitteissa moduuliin lisättiin ominaisuuksia synkronointitapahtumasta informoimiseen sekä optimoitiin suorituskykyä. Moduulin toteutuksessa huomioitiin mahdollinen uudelleenkäyttö siten, että synkronoitavaan peliin liittyvät erityispiirteet piilotettiin abstraktiotason avulla. Lisäksi moduulin toteutuksessa huomioitiin käytettyjen periaatteiden siirtämisen mahdollisuus muihin vastaaviin synkronointimoduuleihin.

Toteutettu moduuli vastasi käyttötarkoitustaan eli sen avulla pystyttiin välittämään pelin tapahtumia pelaajien kesken. Moduulin toiminta oli myös luotettavaa, pelin tiedot säilyivät synkronoinnin yhteydessä oikeina ja synkronointi myös onnistui, kun se suoritettiin. Moduuli tarjosi mahdollisuuden näyttää pelaajille tapahtuneet muutokset, jolloin pelaajilla säilyi hyvä käsitys pelin tilasta. Moduulin suorituskykyyn jäi huomattavan paljon parannettavaa, koska suorituskykyä parannettaessa olisi moduulin monimutkaisuus lisääntynyt huomattavasti. Tämä lisääntynyt monimutkaisuus olisi vaatinut laajaa testaamista käyttövarmuuden saavuttamiseksi, ja tämän suorittamiseen ei ollut käytettävissä tarvittavaa aikaa. Moduulissa havaittiin myös monia kehitysmahdollisuuksia, jotka parantavat käytettävyyttä ja mahdollistavat pelin joidenkin rajoitteiden poistamisen. Näihin kehitysmahdollisuuksiin ei kuitenkaan työn toteutuksessa tartuttu aikarajoitteiden vuoksi.

LÄHTEET

Androutsellis-Theotokis, S., Spinellis, D. 2004. A Survey of Peer-to-Peer Content Distribution Technologies. ACM Computing Surveys Dec2004, Vol. 36 Issue 4. pp: 335 – 371.

Balasubramaniam, S., Pierce, B. C. 1998. What is a file synchronizer? MobiCom '98: Proceedings of the 4th annual ACM/IEEE international conference on Mobile computing and networking. pp: 98 – 108.

Brennan, J. 2006. A Newbie's Guide to Play By E-Mail Diplomacy. [WWW][viitattu 7.5.10] Saatavissa: http://www.diplom.org/Email/newintro.html

Collins-Sussman, B., Fitzpatrick, B. W., Pilato, C. M. 2008. Version Control with Subversion, Second Edition. O'Reilly Media. 432p.

Cox, T. 2000. Online and multiplayer gaming — An overview. Virtual Reality Volume 5, Number 4 / December, 2000. Springer London.

Crichlow, J. M. 2001. Hajautetut tietojärjestelmät. IT Press. 186s.

Gosling, J., Joy, B., Steele, G. & Bracha G. 2005. The Java™ Language Specification. 3rd Addison-Wesley. 649 p.

Heiss, J. J., Zakhour, S. 2009. The Java NIO.2 File System in JDK 7. [WWW] [viitattu 14.4.10]. Saatavissa: http://java.sun.com/developer/technicalArticles/javase/nio/.

Jokinen, J. 2010. Socket-ohjelmointi [WWW] [viitattu 14.4.10]. Saatavissa: http://www.cs.tut.fi/~hajap/luennot/socket.pdf.

Kon, F. 1996. Distributed File Systems Past, Present and Future. A Distributed File System for 2006. Technical Report, University of Illinois at Urbana-Champaign.

Mullender, S. 1993. Distributed Systems, Second Edition. Addison-Wesley. 601p.

O'Brien, L. 1997. Multiplayer Math. [WWW][viitattu 12.5.10]. Saatavissa: http://www.gamasutra.com/view/feature/3221/multiplayer_math.php

Reese, G. 2000. Distributed Application Architecture. Database Programming with JDBC and Java, Second Edition. pp: 126-145.

Rizzo, T. 2006. Database Mirroring. Pro SQL Server 2005. Apress. pp: 567-607.

Sharp, R. 1979. The Game of Diplomacy. Arthur Barker. 192p.

Walls, C., Breidenbach, R. 2007. Spring in Action, Second Edition, MEAP Edition. Manning Early Access Program, Manning Publications. 705p.

Wierzbicki, J. 1999. The Essentials of Multiplayer Games. [WWW] [viitattu 5.5.10]. Saatavissa: http://www.gamedev.net/reference/articles/article722.asp