Metropolia Ammattikorkeakoulu
Tietotekniikan koulutusohjelma

Ari Rautio

Muistipeli

Insinöörityö 27.2.2009

Ohjaaja: koulutuspäällikkö Markku Karhu

Metropolia Ammattikorkeakoulu **Insinöörityön tiivistelmä**

Tekijä Otsikko Sivumäärä Aika	Ari Rautio Muistipeli 48 sivua 27.2.2009
Koulutusohjelma	tietotekniikka
Tutkinto	insinööri (AMK)
Ohjaaja	koulutuspäällikkö Markku Karhu

Insinöörityössä tehtiin graafinen Javalla toimiva muistipeli. Työn tarkoitus oli tutustua ohjelmistontuotannon eri vaiheisiin käytännönläheisen projektin avulla. Taustalla oli halu oppia uusia asioita ohjelmoinnista.

Työ aloitettiin pelissä vaadittavien toimintojen ja komponenttien kartoittamisella. Toimintojen perusteella luotiin pelin alustava ulkoasu ja luokkakaavio. Näiden jälkeen aloitettiin varsinaisen ohjelman tekeminen. Koodaamisen yhteydessä tehtiin testausta jo kahden ensimmäisen luokan Kortti ja Pelilauta luomisen yhteydessä. Kun nämä saatiin toimiviksi, luotiin Kehysluokassa peliin valikot ja alivalikot sekä näihin liittyvät toiminnot. Viimeisenä luotiin tietokanta, joka toteutettiin kahden luokan avulla, jotka olivat PelitulosTietue ja Tulostaulu. Näiden luokkien testaus ei ollut pelin kannalta kovinkaan kriittistä ja ne testattiinkin kaikkien muiden luokkien kanssa. Syy miksi näin tehtiin, oli se, että testaaminen ilman muiden luokkien läsnäoloa oli mahdotonta. Näiden kahden luokan saaminen toimimaan muiden luokkien kanssa olikin työn vaikein osuus.

Ohjelmointityö tehtiin NetBeans IDE 6.1-version avulla. Ohjelmalla saatiin aikaiseksi graafinen peli. Työn vaiheina olivat määrittely, suunnittelu, toteutus ja testausvaiheet. Työ eteni samaan tyyliin kuin palapelin kasaaminen eli ohjelmistoon luotiin yksi uusi ominaisuus kerrallaan. Tämän jälkeen ohjelmaa testattiin lähinnä pelaamalla ja tarkkaillen, toimiiko lisätty toiminto oikealla tavalla. Lopputuloksena saatiin aikaiseksi peli, joka toteutti kaikki määrittelydokumentissa asetetut tavoitteet. Hankaluutena työssä oli se, ettei ollut kokemuksia graafisten ohjelmistojen teosta. Näin ollen ohjelmointityön aikana joutui opettelemaan paljon uusia asioita. Tämä vaati asioihin paneutumista ja aikaa itse ohjelmointiin kuluikin oletettua enemmän. Tärkeimmät vaiheet työssä olivat määrittely- ja suunnitteluvaiheet. Näihin vaiheisiin tulee panostaa kunnolla, koska muuten ei tällainen ohjelmointityöprojekti onnistuisi.

Hakusanat	Java, ohjelmistotuotanto, graafinen käyttöliittymä

Helsinki Metropolia University of Applied Sciences **Abstract**

Author Title Number of pages Date	Ari Rautio Memory game 48 27 February 2009
Degree Programme	Information Technology
Degree	Bachelor of Engineering
Instructor	Degree Programme Director, Markku Karhu

The goal of this thesis was to create a graphic Java game. The ultimate aim was to become acquainted with the various stages of a new programming paradigm through a practice-oriented project.

The work was started by exploring what kind of actions and components were needed. Next, the first game layout and class diagram model were built, followed by programming and associated testing. At first, two classes, the Card class and the Game-Board class, were created. Once they had been found to work properly, a class named Frame was built to implement the menus and submenus as well as the actions needed.

A database was finally built by means of two classes named GameResultRecord and ResultTable. Testing of individual classes one by one was not feasible in this case, because the functionality of the classes was so intricately intertwined. In such circumstances the final testing is the only way to determine whether the program is working correctly. The most challenging part of this project was thus the interplay of the GameResultRecord and Result classes with the rest of the classes of the game.

The programming was done using NetBeans 6.1 software. The software engineering processes for this project included: specification, design and implementation as well as testing. Because the
project was intended for home users, there is no maintenance responsibility involved.

The work proceeded feature by feature very much like solving a jigsaw puzzle. The game was tested by playing it as well as by establishing that the individual features worked correctly, till all the inserted features in the project were found to work appropriately together and thus meet the software specifications made. The difficulties encountered were to a great extent due to lack of experience with a graphical user interface (GUI). The need to study a number of new things made the programming more time-consuming than expected.

Keywords	Java, software engineering, GUI

Sisällys

1 Johdanto

Tämän insinöörityön aiheena on laatia Javalla toteutettu muistipeli. Pelin graafinen käyttöliittymä toteutetaan NetBeans IDE 6.1 -version avulla. NetBeans IDE -kehitysympäristö on Sun MicroSystemsin sponsoroima avoimeen lähdekoodiin perustuva ilmainen ohjelmisto, joka tukee monia ohjelmointikieliä. Ohjelman avulla voidaan tuottaa koodia, kääntää ja jäljittää virheitä siitä. [1.] Suunnitteluvaiheessa sillä voidaan piirtää kaavioita (esimerkiksi käyttötapaus- ja luokkakaaviot) sekä rakentaa alustava käyttöliittymä valmiiden objektien avulla. Ohjelmassa on myös mahdollista generoida piirretystä käyttöliittymästä elementeistä suoraan koodin luuranko. Luurankokoodi on perusta, joka sisältää tärkeimmät toiminnot ja jonkin verran koodia, mutta varsinaisen toiminnallisuuden joutuu itse kirjoittamaan. Tämä vaatii kuitenkin luurankokoodin täydellistä ymmärtämistä, jotta sitä voitaisiin soveltaa omassa projektissa. Tässä työssä koodi kirjoitetaan kuitenkin käsin. Projekti alkaa tutustumalla Netbeans-organisaation sivulla [2] oleviin tutoriaaleihin.

Työn tavoitteena on laatia yksinkertainen ja helppokäyttöinen toimiva graafinen peli, jonka tekemisessä tulisi käytyä läpi kaikki ohjelmiston tuotannossa olevat tärkeimmät prosessit eli määrittely, suunnittelu, toteutus ja testausvaiheet [3, s.73]. Vaiheet on jaettu työssä omiin lukuihinsa, jotta työn lukija löytää tarvittavan tiedon mahdollisimman helposti. Jako helpottaa myös työn mahdollista jatkokehittelyä tulevaisuudessa.

Pelin ideana on kääntää jokaisella kierroksella kaksi pelikorttia ja yrittää muistelemalla edellisellä kierroksella näkemiään kääntöjä löytää kaksi samaa korttia. Pelin tarkoituksena on kehittää muistia ja toimia samalla koko perheen ajanvietteenä. Peliä voi pelata vain yksi henkilö kerrallaan.

Pelilauta koostuu korteista, joiden alla esiintyy numero, joka satunnaisgeneraattorilla arvotaan. Korttien selkäpuoli on kaikissa korteissa samanlainen. Jaettavat kortit sekoitetaan ja asetetaan pelilaudalle nurinpäin. Pelissä on tarkoitus etsiä pareja, jotka ovat numeroltaan samanlaisia. Jokaisella vuorolla on mahdollista valita kaksi pelilaudalla ole-

vaa korttia ja yrittää näin löytää sopivia pareja. Valitut kortit käännetään näkyviin. Jos käännetyt numerot eivät ole samat, käännetään ne takaisin nurinpäin ja numero poistuu samalla näkyviltä. Mikäli kortit olivat samanlaiset, jäävät ne kuvapuoli alaspäin pelilaudalle ja muuttuvat väriltään harmaiksi, jolloin myös numero katoaa näkyviltä. Tämä kertoo sen, ettei näitä kortteja voi enää pelata. Käyttäjä voi aloittaa uuden pelin milloin tahansa valitsemalla *Tiedosto*-valikosta *Uusi peli*. Uuden pelin valinnan yhteydessä on mahdollista määrittää myös vaikeustaso. Kun peli käynnistyy, samalla käynnistyy ajastin, joka laskee pelissä kuluvaa aikaa. Jos läpäisee pelin riittävän nopeasti, pääsee kymmenen parhaimman pelaajan listalle. Parhaita tuloksia voidaan tarkastella milloin tahansa valitsemalla *Tiedosto*-valikosta *Parastulos*-toiminto.

2 Määrittely

Määrittelyvaiheessa kuvataan muistipelille asetettavia toiminnallisia vaatimuksia. Prosessia voidaan myös kutsua työn esitutkintavaiheeksi. Tässä luvussa pyritään selvittämään, mitä vaatimuksia ohjelmoitava peli vaatii. Vaatimusten kartoittamisen tuloksena syntyy määrittelydokumentti, joka on pohjana suunnitteludokumentille ja samalla myös karkea kuvaus käyttöliittymästä ja sen perustoiminnoista, joita voidaan tarkentaa myöhemmin projektin aikana toteutusdokumentissa. Määrittelyn kirjoittamisen tukena on käytetty Jaakko Nenosen ohjetta määrittelydokumentin kirjoittamisesta [4].

Työn määrittely aloitettiin tekemällä alustava järjestelmän kartoitus eli mietittiin, mitä erilaisia toimintoja peli vaatii. Huomattiin, että kaikki valikoissa olevat toiminnot ovatkin niitä käyttötapauksia, joissa käyttäjä ja muistipeli ovat vuorovaikutuksessa keskenään. Myös tietokoneohjelman itsensä suorittamat käyttötapaukset selvitettiin. Näiden pohjalta saatiin aikaan kaikki pelin käyttötapaukset, joista piirrettiin kaavio NetBeans-ohjelman avulla.

Tämän jälkeen suunniteltiin alustava käyttöliittymän ulkoinen asu, joka on hyvin karkea kuvaus käyttöliittymän toiminnoista eli tässä vaiheessa ns. ensimmäinen prototyyppi järjestelmän ulkoisesta rakenteesta. Käyttöliittymän piirrosta tullaan hyödyntämään työn toteutusvaiheessa eli koodausvaiheessa. Määrittelyvaiheessa ei vielä lyödä tarkasti lukkoon ohjelman tarvitsemia tietorakenteita eikä lopullista ulkoista asuakaan, koska ne saattavat vielä muuttua suunnitteluvaiheessa. Määrittelyvaihe on projektin kannalta erittäin tärkeä vaihe, ja se ratkaisee hyvin pitkälti sen, onnistuuko projekti vai ei. Määrittelyvaiheessa priorisoidaan työn kannalta keskeiset työvaiheet, jotka tulee ensimmäisessä versiossa vähintään olla ja mitkä voidaan toteuttaa myöhemmin.

2.1 Järjestelmän kartoitusvaihe

Kartoitusvaiheessa pohditaan, mitä ominaisuuksia ja toimintoja peli ensisijaisesti tarvitsee ja kirjoitetaan ne auki mahdollisimman tarkasti. Varsinaisiin tietorakenteisiin ei tässä vaiheessa vielä puututa. Toimintoja ja ominaisuuksia voidaan lisätä myöhemmin, jos siihen on tarvetta.

Pelilauta tulee muodostumaan korteista, joiden alla voi esiintyä numero. Pelissä on tarkoitus etsiä pareja, jotka ovat numeroltaan samanlaisia. Pelaaja kääntää jokaisella kierroksella kaksi korttia ja yrittää muistelemalla edellisellä kierroksella näkemiään kääntöjä löytää kaksi samannumeroista korttia. Mikäli kaksi korttia eivät ole samat, käännetään ne takaisin, kun napsautetaan seuraavaa korttia. Mikäli kortit olivat samanlaiset, poistetaan kortit pelilaudalta. Pelistä toteutetaan vain yhden pelaajan versio.

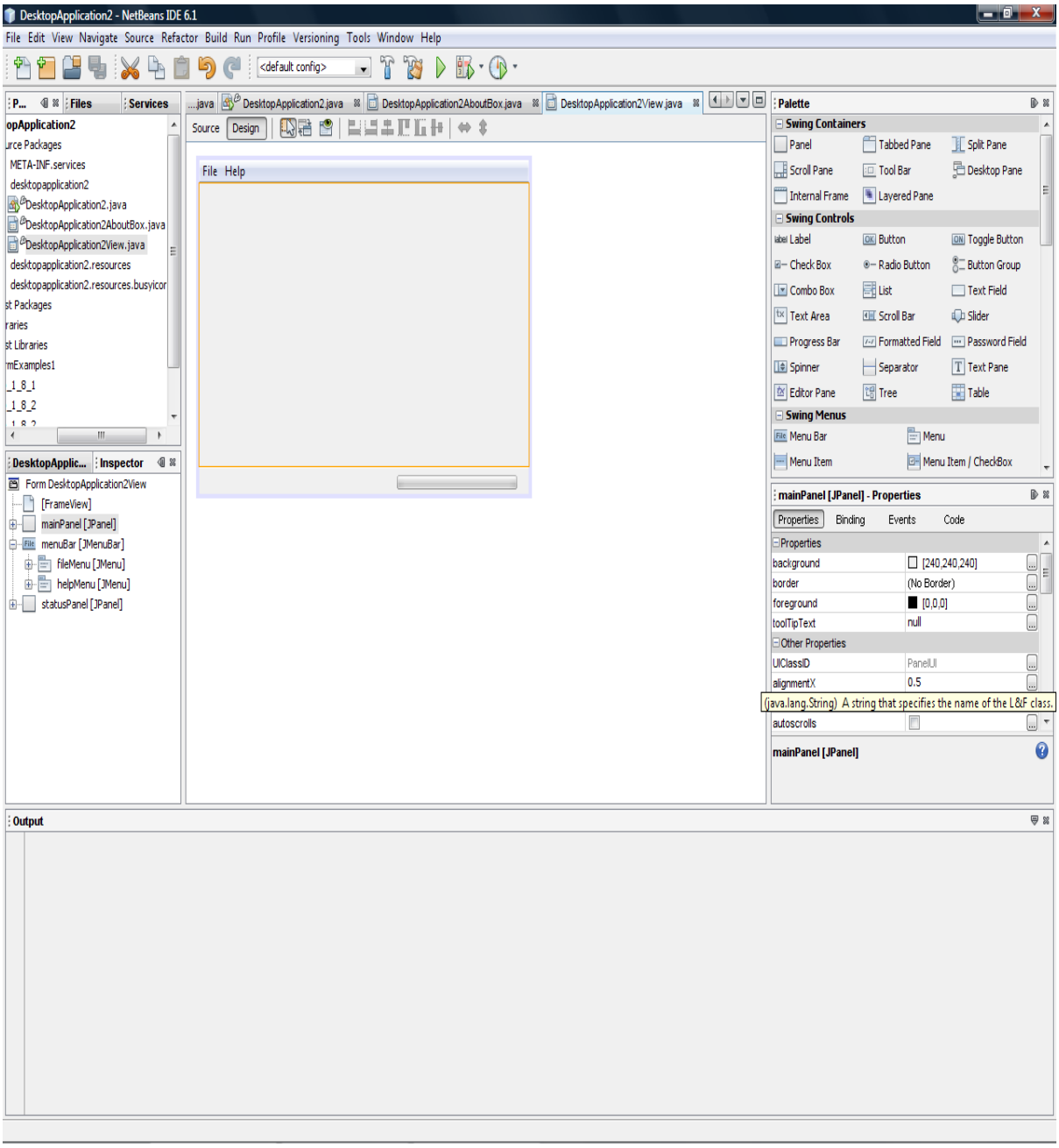
Pelissä rajoituksen tulee asettamaan näytön resoluutio, joka määrittää, montako korttia pelialustalle mahtuu ilman, että näyttöä tarvitsee vierittää. Minimiresoluutioksi suositellaan 1280 x 800 pikseliä tai suurempaa. Jotta peliä pystyisi mahdollisimman moni pelaamaan, on korttien maksimimäärälle asetettu maksimiksi 8 x 8 ruudukko (64 korttia). Näin kortit saadaan mahtumaan ruudulle. Näytölle asetettava minikoko on 15 tuuman näyttö, jonka tulee kyetä suoriutumaan edellämainituista resoluutiovaatimuksista. Keskusmuistimäärälle ei tarvitse asettaa rajoituksia, koska ohjelma tulee vaatimaan vain vähän muistia toimiakseen. Nykypäivän koneet suoriutuvat kaikista yllämainituista vaatimuksista, paitsi pienellä alle 15 tuuman näytöllä varustetut kannettavat. Näissä näytön resoluutio ei välttämättä riitä näyttämään korttien numeroita ja pelaaminen on tällöin mahdotonta.

Kokonaista korttipakkaa ei tässä työssä ole tarkoitus valokuvata eikä kuvien piirtämistä tässä työssä myöskään tehdä, koska siinä kuluisi liikaa aikaa. Numerot, joita korteissa käytetään, generoidaan satunnaisgeneraattorin avulla. Se arpoo jokaisessa pelissä kaksi samalla numerolla varustettua pelikorttia. Korttien kääntöpuolelle kuva otetaan digitaalikameralla (kuva 1) ja skaalataan se sopivan kokoiseksi myöhemmin pelin toteutusvaiheessa.

Kuva 1. Kortin kuva.

Alustava suunnitelma siitä, mitä valikkoja lopullisessa versiossa voisi olla, toteutettiin generoimalla alustava käyttöliittymä suoraan NetBeansin avulla. Generointi tapahtui valitsemalla valikosta uusi *Java Desktop Application*, joka luo automaattisesti kuvassa 2 näkyvän käyttöliittymän luurangon. Generoitua käyttöliittymää voidaankin jo tässä vaiheessa tarkastella ja myös jatkokehittää ohjelman paletissa esiintyvillä työkaluilla. Samalla voi tutustua myös NetBeansin generoimaan koodiin, vaikkei olisi koodannut vielä ainuttakaan riviä ohjelmasta. Tämä on erittäin hyvä ominaisuus päästä tarkastelemaan, miltä käyttöliittymä voisi näyttää. Apua tästä on varsinkin ensikertalaisille, jotka laativat omia sovellutuksia.

Kuvan 2 englanninkielisten valikoiden tekstejä muokkaamalla päädyttiin käyttöliittymäsuunnittelussa sitten kuvan 4 mukaiseen kuvaan. *Tiedosto*-valikkoon kuuluvat toiminnot voisivat olla uuden pelin valinta, josta voitaisiin haarautua vielä vaikeustason valinta toimintoihin. *Tiedosto*-valikossa voisi löytyä myös paras tulos ja pelin lopetustoiminnot. *Ohje*-valikossa olisivat tiedot ohjelmaversiosta ja pelin käyttöohje.



Kuva 2. NetBeansin generoima käyttöliittymä

2.2 Pelissä käsiteltävä tieto

Pelissä on pidettävä yllä korttien määrää ja niiden sijaintitietoja. Myös löydettyjen parien määrä ja pelikorttien kääntökerrat lasketaan laskureiden avulla. Pelikorttien käytössä olevat numerot tulee olla taulukoituina jossakin luokassa. Aktiivisia viimeksi käännettyjä kahta korttia voidaan vertailla aputaulukon avulla.

Pistemäärän laskenta voitaisiin toteuttaa siten, että se on sidoksissa kääntökertojen ja löydettyjen parien määrään sekä mahdollisesti vielä vaikeusasteeseenkin. Tämä on mahdollista toteuttaa esimerkiksi jonkin matemaattisen kaavan avulla. Parhaimmat pistemäärät ovat tallessa tiedostossa, jonne tallentuu tulos, vain jos saatu tulos on parempi kuin edellinen tulos. Pistemäärien tarkistamisen yhteydessä käytetään aputaulukkoa, jonka avulla pistemäärien vertaaminen tapahtuu. Vertailussa tarkistetaan, onko edellisen kerran paras pistemäärä huonompi vai parempi kuin syntynyt tulos. Jos tulos on suurempi kuin nykyinen paras pistemäärä, se lisätään parhaimpien pisteiden listalle ja vanha paras tulos siirtyy listalla yhden pykälän alaspäin. Jos listalle halutaan esimerkiksi kymmenen henkilön pisteet, on vertailuja tehtävä huomattavasti enemmän. Korkeimman pistemäärän pääsee näkemään valitsemalla *Tiedosto*-valikosta *Parastulos*.

2.3 Pelin lähettämät viestit käyttäjälle

Pelissä käyttäjä liikuttaa hiirtä pelialustalla ja napsauttaa sitä valitessaan kaksi korttia. Käännetyt kortit, joissa on sama numero, viestivät käyttäjälle, onnistuiko parin valinta vai ei. Peli näyttää viestiosassa käyttäjälle käännettyjen parien määrän ja mahdollisesti pelissä siihen saakka kertyneen pistemäärän ja peliajan sekunteina. Käyttäjä näkee löydetyt parit, kun ne kääntyvät pelialueella nurinpäin ja muuttuvat samalla harmaiksi eikä niitä pysty enää kääntämään. Vain jäljelle jääneitä kortteja voidaan yrittää valita pelilaudalta. Käyttäjä näkee koko ajan käytetyn peliajan, ja se pysähtyy, kun peli päättyy.

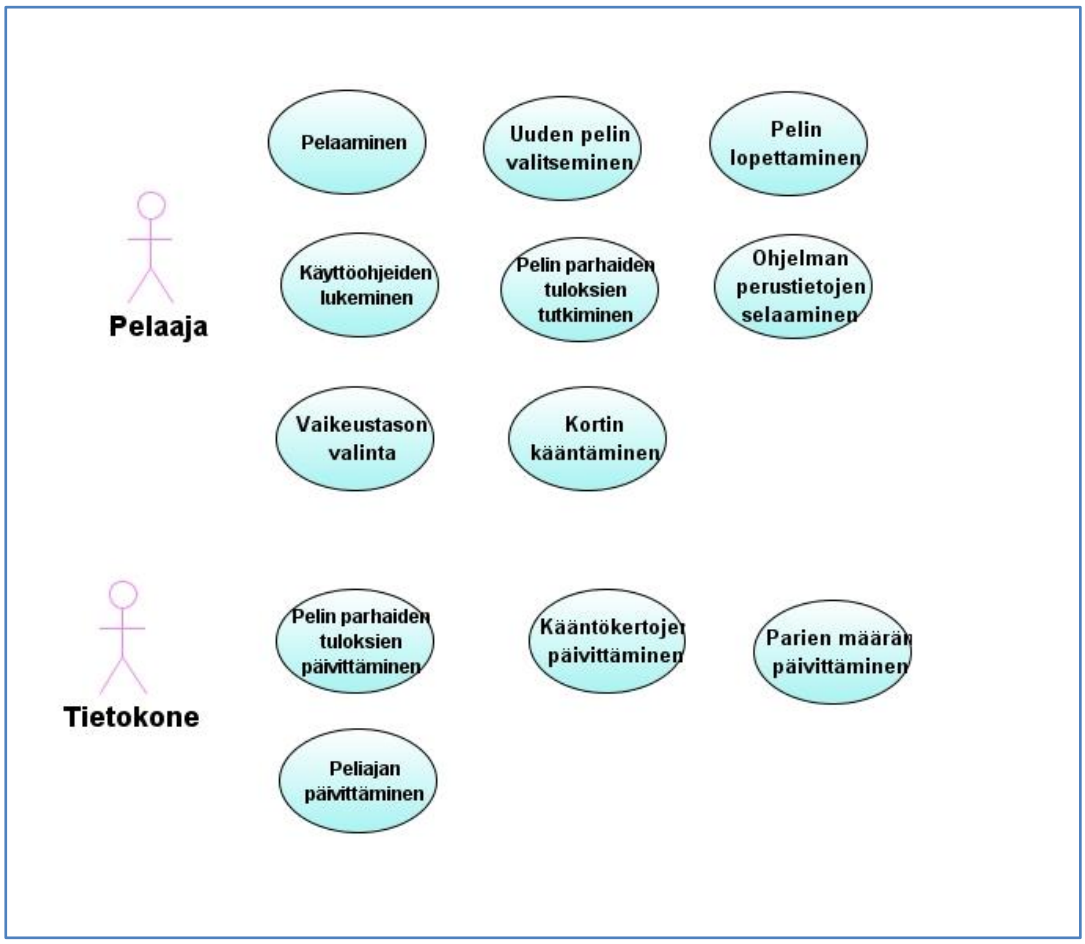
2.4 Käyttötapausten kuvaus

Pelin tärkein käyttötapaus on muistipelin pelaaminen. Pelin aloitus tapahtuu valitsemalla *Tiedosto*-valikosta *UusiPeli* ja tämän jälkeen käyttäjälle avautuu alivalikko, josta hän voi valita haluamansa vaikeustason *Helppo*, *Keskivaikea* tai *Vaikea*. Helpossa vaikeustasossa pelattavien korttien määrä olisi 4 x 4, Keskivaikeassa 6 x 6 ja Vaikeassa 8 x 8 korttia. Vaikeustason valitsemisen jälkeen kortit sekoitetaan ja ne jaetaan pelipöydälle ympäri käännettyinä ja pelaaja pääsee tämän jälkeen aloittamaan pelin pelaamisen. Pelaajan tehtävänä on yrittää löytää korteista samaa numeroa vastaavia pareja kääntelemällä niitä pelialueella.

Kääntäminen tapahtuu hiiren avulla napsauttamalla jotain korttia pelialueelta. Pelaaja näkee vain kaksi korttia kerrallaan käännettyinä. Jokaisesta kortin kääntökerrasta pidetään kirjaa, samoin kuin jo löydetyistä pareista. Kun kaikki parit pelilaudalta ovat löytyneet, peli päättyy. Uuden pelin voi aloittaa koska tahansa valitsemalla *Tiedosto*-valikosta *UusiPeli*.

Mikäli pelissä halutaan paras pistemäärä talteen, tarvitaan ohjelmassa tiedostoon tallennus. Pisteiden tarkistamisen jälkeen, jos tulos oli parempi kuin edellinen paras, aukeaa käyttäjälle dialogi, josta käyttäjälle näytetään parhaiden pelaajien lista. Tallennuksessa voidaan kysyä myös pelaajan nimi, jolloin se näkyy tarkasteltaessa parasta pistemäärää. Samoin pelaamiseen käytetty aika sekunteina tallennetaan. Pelin aikana tulee olla mahdollista tutkia myös, mikä on pelin paras tulos pelihetkellä. Tämä toiminto tulee *Tiedosto*-valikkoon.

Käyttöohjeiden lukeminen pelin aikana on myös yksi käyttötapaus. Käyttöohje löytyy klikkaamalla *Ohje*-valikosta *KäyttöOhje*. Ohje-valikosta voidaan myös napsauttaa kohtaa *Tietoja Ohjelmasta*, jolloin näytölle tulee näkyville ohjelman perustiedot, kuten ohjelmaversion numero ja tekijän tiedot. Ohjelman sulkeminen tapahtuu *Tiedosto*-valikon kautta valitsemalla *Lopeta* tai Windows-ohjelmien perinteisellä tavalla eli oikeasta yläkulmasta rastia painamalla (kuva 12, sivu 41). Pelaajan ja tietokoneen käyttötapaukset esitetään kuvassa 3.

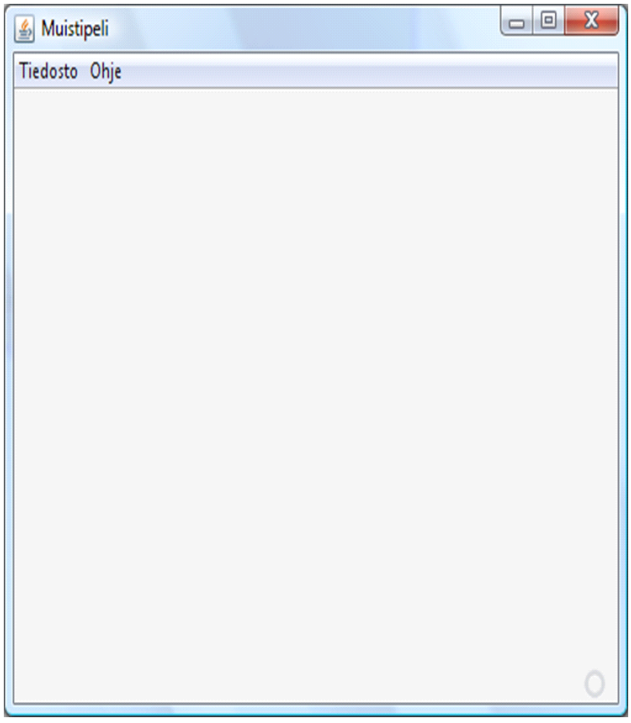


Kuva 3. Käyttötapausdiagrammi

2.5 Pelin käyttäytymisen kuvaaminen

Muistipelissä käytetään pelilautaa, jossa käännettävät pelikortit sijaitsevat. Pelikortteja käännetään ympäri napsauttamalla haluttua korttia. Kortteja voi olla käännettyinä vain kaksi vuorollaan. Mikäli paria ei löytynyt, kortit kääntyvät selkäpuoli ylöspäin napsauttamalla seuraavaa korttia. Jos ensimmäisen kortin valinnan jälkeen napsautus kohdistuu jo kääntyneenä olevaan korttiin ja sitä napsautetaan uudelleen, ei tapahdu mitään. Käyttäjän tulee valita kaksi eri korttia hiirellä, jotta ne voidaan hyväksyä yrityksiksi. Vain kun molemmat kortit ovat kääntyneinä, napsautettaessa mitä tahansa korttia molemmat kortit kääntyvät selkäpuoli ylöspäin. Pelialueella on pelitilanteesta kertova tietoalue, josta käyttäjä voi seurata pelin kannalta keskeisiä tietoja.

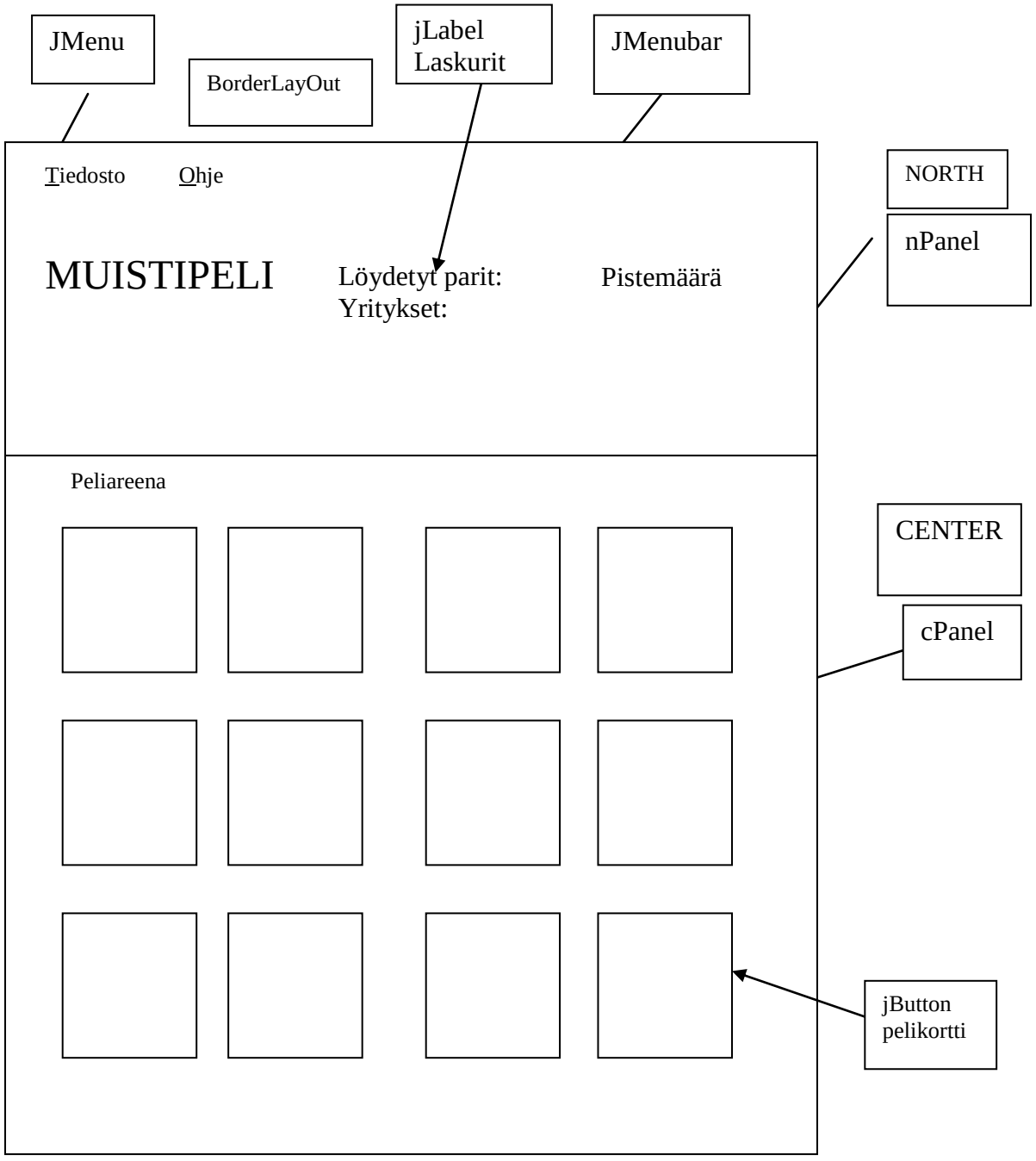
Ikkunan ylälaidassa on kaksi valikkoa. *Tiedosto*-valikosta voidaan valita peliin liittyvät yleisimmät perustoiminnot, kuten uuden pelin valinta, pelin lopettaminen ja paras pistemäärä. Kun uusi peli valitaan, avautuu käyttäjältä alivalikko, josta voidaan valita vaikeustaso ennen pelin käynnistämistä. Tämä määrittää sen, kuinka monta korttia pelilaudalle jaetaan. Toinen valikoista on *Ohje*-valikko, jota käytetään pelin ohjeistukseen ja antamaan tietoja itse ohjelmasta. *Ohje*-valikosta löytyy käyttöohjeistus, jossa on lähinnä pelaamiseen tarvittavia ohjeita eli miten peliä voidaan pelata. Tietoja ohjelmasta -valinta avaa uuden ikkunan, joka kertoo pelin tekijän tietoja ja muistipeliohjelman versionumeron. Käyttöohjeistus on tarkoitus toteuttaa siten, että käyttäjälle avautuu uusi ikkuna HTML-muodossa ruudulle, josta hän voi sitten hakea tarvitsemiaan ohjeita pelin pelaamiseen. Käyttöliittymän suunnitelma on esitetty kuvassa 4. Ohjeita voidaan joutua tarkentamaan myöhemmin suunnittelu- ja toteutusvaiheessa.



Kuva 4. Käyttöliittymän suunnitelma

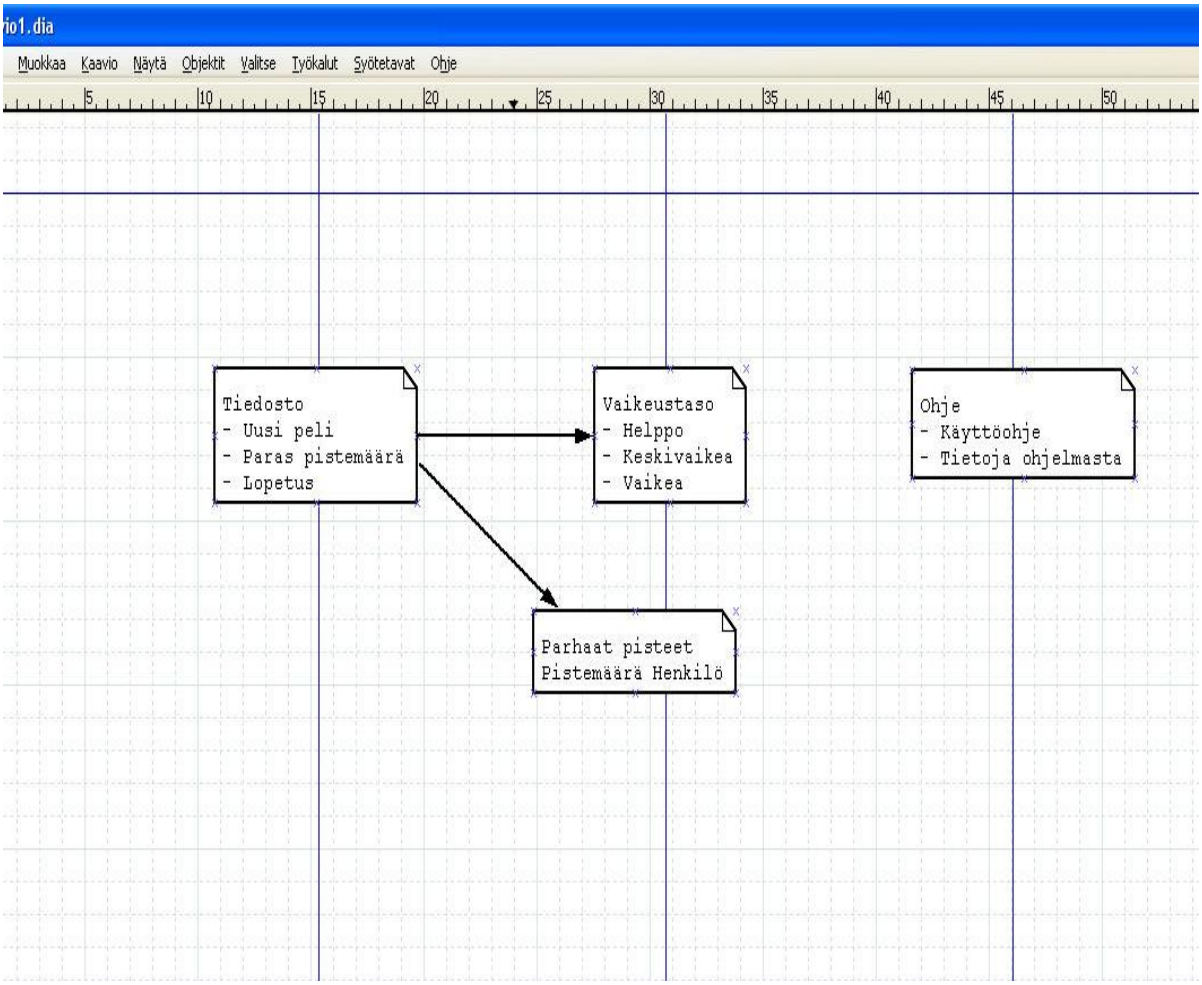
Kuvassa 5 on piirretty ensimmäinen käyttöliittymäsuunnitelma, josta näkee, miten valikot ja objektit tulevat sijoittumaan ruudulle sekä mistä komponentista ne mahdollisesti periytyvät. Pelikortti toteutetaan periyttämällä se *jButtonista*. Laskurit ja pistemäärä

ovat *jLabelista* periytettyjä komponentteja. Sijoittelumanagerin pohjapaneelina käytetään BorderLayouttia, jolla paneeli voidaan jakaa North-, South- ja Center-osiin.



Kuva 5. Käyttöliittymän komponentit ja rakenne

Kuvassa 6 esitetään kaikki ohjelman valikot ja toiminnot. *Tiedosto*-vali-kosta voidaan valittaessa *UusiPeli* haarautua alivalikkoon, jossa voidaan määrittää pelin vaikeustaso. Vastaavasti *Parastulos* -toiminnosta avautuisi *Parhaat tulokset* -dialogi-ikkuna, josta voisi tarkastella senhetkisiä parhaita tuloksia. Ohjevalikosta voitaisiin valita *Käyttöohje* ja *Tietoja Ohjelmasta* -toiminnot. Käyttöohje toteutettaisiin html-tiedoston avulla sekä Tietoja ohjelmasta -dialogi-ikkunan avulla.



Kuva 6. Ohjelman valikot ja toiminnot

2.6 Ohjelman toteutuksen priorisointi

Toteutuksessa määritellään ohjelman teon kannalta merkittävimmät prioriteetit, eli miten kiireellisesti mitäkin ohjelman toimintoja tulee toteuttaa. Tämä auttaa ohjelman tekemisessä. Parhaiden pistemäärien listan toteutuksen prioriteetti on matala, ja sen toteuttaminen on vaikeusasteeltaan hyvin vaativa, koska se vaatii tietokannan eikä sitä toteuteta ainakaan ohjelman ensimmäisessä versiossa. Pelattavien korttien määrän eli vaikeusasteen prioriteetti on korkea ja sen toteutus helppo, koska se ei vaadi suurtakaan lisätyötä. Käyttöohjeen tekeminen on yksi keskeisimmistä asioista, joita jokaisessa pelissä on oltava. Sen toteutus on myös luultavasti helppo. Kaikki muut aiemmin määrittelyvaiheessa kuvatut asiat tullaan kokonaisuudessaan toteuttamaan eikä niitä erikseen tarvitse priorisoida.

3 Suunnittelu

Ohjelman suunnitteluvaiheen tarkoitus on täsmentää määrittelyvaiheessa esitettyjä järjestelmän määrittelyyn liittyviä käyttötapauksia ja toimintoja. Tässä vaiheessa suunnitellaan myös luokkakaavio ja ohjelman varsinaiset tietorakenteet. Käytännössä suunnittelu- ja toteutusvaiheet kulkevat työssä rinnakkain, koska käytännössä on mahdotonta suunnitella ohjelmistoa niin hyvin, että se pystyttäisiin sellaisenaan koodaamaan alusta loppuun ilman, että suunnitteludokumenttiin tarvitsisi tehdä muutoksia. Myös tässä luvussa on hyödynnetty Jaakko Nenosen sähköisiä dokumentaatio-ohjeita. [5, 6.]

Ohjelman suunnitteluvaiheessa tuotetusta dokumentista on yleensä hyötyä jollekin toiselle ohjelmoijalle, joka tulee projektiin myöhäisemmässä vaiheessa mukaan tai jopa sen päättymisen jälkeen. Hänellä tulee olla mahdollisuus tutustua paremmin ohjelman toimintaan ja päästä siihen sisälle. Tarkasta dokumentaatiosta on hyötyä myös tulevaisuudessa, kun ohjelmistoa aletaan ylläpitää. Suunnitteludokumentissa tulee kuvata ohjelman tekemisen ratkaisuperiaatteet, käytetyt tietorakenteet ja ohjelman toiminnan pääpiirteet. Tässä työssä lisäyksiä tietorakenteisiin tehtiin vielä version 2 valmistumisen myötä ja niitä käsitellään toteutusluvussa.

3.1 Käyttöliittymän suunnittelussa huomioonotettavia seikkoja

Käyttöliittymän tulee olla selkeä, eikä liikaa ominaisuuksia tai valikoita saa olla muistettavana. Toimintojen tulee perustua tavanomaiseen Windows-ohjelmista jo tuttuihin ominaisuuksiin ja on pyrittävä pitämään käyttäjäkynnys mahdollisimman matalana. Mitä helpommin ohjelma on opittavissa ja mitä miellyttävämpi käyttäjäkokemus on, sitä paremmin ohjelman suunnittelussa on onnistuttu. Jos käyttöliittymän toimintojen etsimiseen menee käyttäjällä liikaa aikaa, on ohjelmisto liian vaikea käytettävyydeltään ja se menettää hyvin helposti mielenkiinnon. Ohjelman teossa on tärkeää kuunnella loppukäyttäjiä jo testattaessa ensimmäistä alfaversiota eli prototyypin syntyvaiheessa.

Käyttöliittymän ulkoasun tekemiseen kannattaa panostaa, koska se on ensimmäinen asia, johon asiakas kiinnittää huomionsa, kun ohjelma julkaistaan. Useimmiten isoissa projekteissa palkataan visuaalinen suunnittelija suunnittelemaan käyttöliittymän ulkoinen asu. Ulkoasun teossa voidaan käyttää myös Javan Look-And-Feel-suunnitteluohjenuoria [7], joiden avulla on mahdollista saada tuotteesta aikaan sellainen, joka ei poikkeaisi merkittävästi jo olemassa olevista Windows ohjelmista.

3.2 Yleiskuvaus

Muistipelin tekemiseen tarvitaan vähintään kolme luokkaa. Yksi luokka tarvitaan kortille, yksi pelilautaa varten sekä yksi käyttöliittymän ikkunaa ja valikoiden luomista varten. Tässä suunnitteludokumentissa kuvataan niiden pääperiaatteet, eli mitä kukin luokka tulee tekemään. Toteutusdokumentissa kerrotaan, miten suunnitelmat todellisuudessa toteutuivat ja pitikö tehdä vielä muutoksia.

Korttiluokka tullaan periyttämään *Jbuttonista*. Korttiluokan keskeisimpiä metodeja ovat kortin toiminnan asettaminen näkyviin, pois näkyviltä sekä pois pelattavista. Kun kortti käännetään näkyville, se saa numeron arvoksi. Kun kahden valitun kortin numero on sama, käännetään kortit pois pelattavista eli estetään niiden käyttö. Jos korttien numero ei ole sama, käännetään ne vain pois näkyviltä. Korttiluokan ratkaisussa on huomioitava myös vaikeustasot, joissa korttien määrä on jokaisella tasolla eri. Käyttäjän tulee voida valita vaikeustaso uuden pelin valinnan yhteydessä *Tiedosto*-valikosta.

Ohjelman rakenteen tulee olla mahdollisimman joustava, eikä se saa asettaa liikaa rajoituksia ohjelman toteuttamiselle. Korttiluokalle valittiin tietorakenteeksi rajoittamaton taulukko, koska taulukon alkioiden määrä muuttuu vaikeustasojen valinnan yhteydessä. Näin ollen rajoittamattoman taulukon käyttö tuntui ainoalta järkevältä tietorakenteelta, jonka avulla korttiluokka voitaisiin toteuttaa. Graafisten kuvien lisääminen kortteihin olisi vaatinut suuren määrän kuvien piirtämistä tai kuvien ottamista. Näin ollen vain kortin kääntöpuolelle lisättiin itse otettu kuva kortista. Toiselle puolelle ko. kuvaa ei

lisätty, koska toisella puolella käytetään sekoittamisen yhteydessä satunnaisgeneraattorin avulla luotua kortin numeroa. Kortin numero tulee näkyviin napsautettaessa korttia.

Pelilautaluokassa tarvittiin myös korttien sekoittajametodi (*Shuttle)*, joka löytyi Javan luokkakirjastosta. Pelilautaluokka luo pelilaudan korteille, ja kortin sekoittaja sekoittaa kortit ja asettelee ne satunnaisesti sekoitusalgoritmin avulla pelilaudalle. Pelilautaluokassa sijaitsee myös itse pääohjelma eli main-metodi, joka käynnistyessään luo ilmentymän kehysluokkaan, joka vuorostaan asettaa käyttöliittymän ikkunan näkyville.

Voidaankin sanoa, että ohjelman varsinainen käyttöliittymä toteutetaan pelilauta- ja kehys- ja korttiluokan yhteistyönä. Kehys-luokassa luodaan käyttöliittymän ulkoinen asu kuten kehykset ja menut. Näiden lisäksi siellä asetetaan ohjelmaan liittyvät tapahtuman kuuntelijat, jotka reagoivat käyttäjän tekemiin valintoihin menuissa ja niiden perusteella tapahtuma sallitaan ja lopuksi suoritetaan jokin toiminto. Ohjelman luokkia voidaan käyttää myös toisessa luokassa, mutta niistä pitää muodostaa aina ilmentymä, koska on kyse olio-ohjelmoinnista. Muodostamalla ilmentymä toiseen luokkaan, se mahdollistaa, että voidaan toisesta luokasta käyttää kaikkia sen julkisia metodeja ja välittää parametreja myös niiden välillä. Esimerkiksi muodostettaessa kehysluokan ilmentymä voidaan kehysluokkaa käyttää pelin ulkoisten puitteiden luomiseen, pelikentän tietojen päivittämiseen ja aikalaskurin päivittämiseen.

3.3 Luokkarakenne

Pelilautaluokka on pelin pääluokka, joka vastaa pelin käynnistymisestä ja graafisen käyttöliittymän luomisesta. Se sisältää ohjelman päämetodin main, jonka käynnistyessä luodaan kaikki pelin graafiseen käyttöliittymään liittyvät toiminnot. Aluksi main-metodista luodaan ilmentymä kehysluokkaan, jossa sijoitellaan kaikki ruudulle tulevat pelin näkyvät osat, kuten kehys, siihen liittyvät valikot, tilatiedot ja pelikortit. Lisäksi samassa luokassa luodaan kortti-ilmentymän välityksellä myös pelissä käytettävät kortit. Tämä luokka sisältää myös korttiluokasta luodun Arraylist-mallisen taulukon, joka pitää yllä korttien numerot. Korttien määrä saadaan muodostettua kaavalla (rivit x sa-

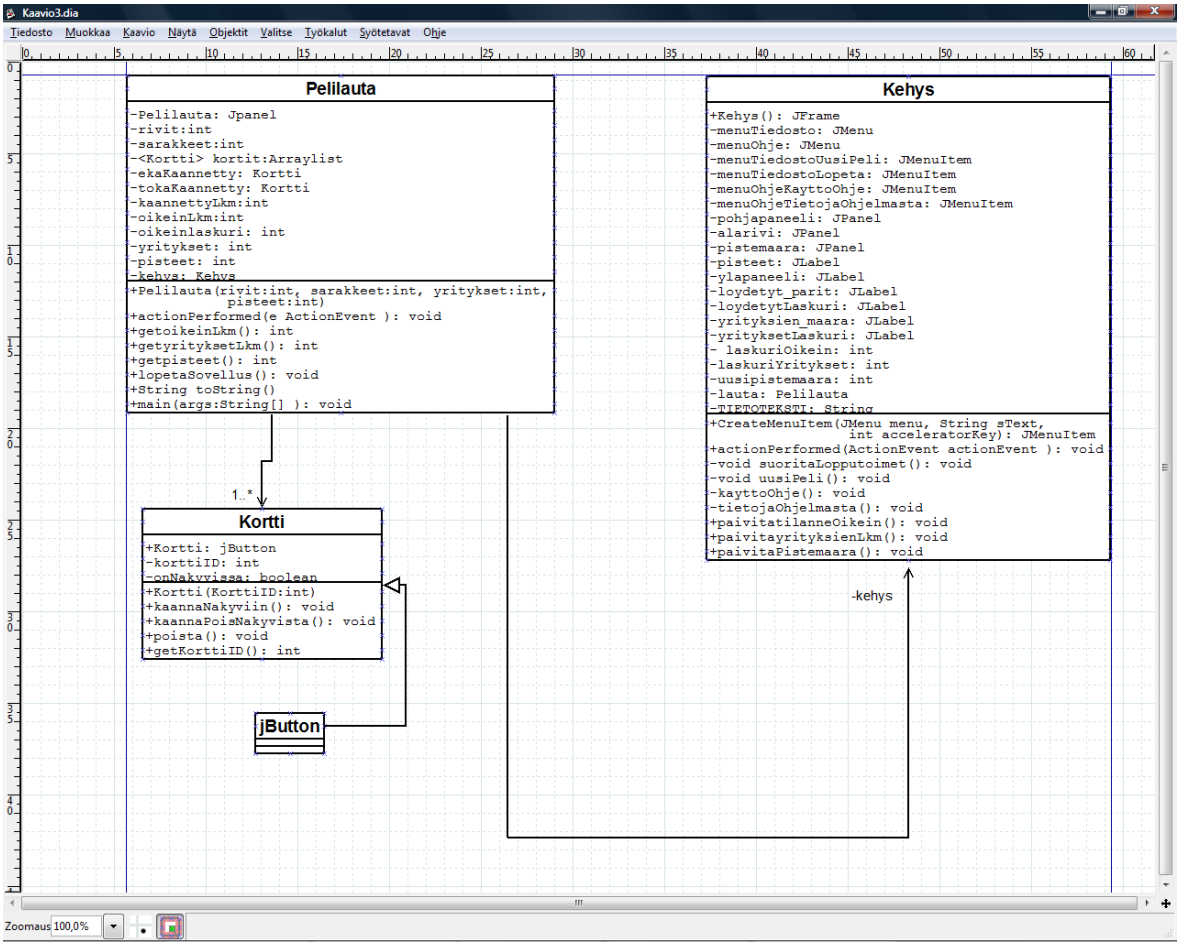
rakkeet)/2. Kun rivien ja sarakkeiden määrä jaetaan kahdella, on mahdollista luoda kaksi eri ilmentymää olioviitteen avulla ja saada aikaan parillinen määrä samaa numeroa vastaavia kortteja. Voitaisiinkin ajatella, että käytettäisiin kahta eri korttipakkaa. Rivien ja sarakkeiden määrän parametrien arvot asetetaan aina pelin vaikeustason valinnan yhteydessä tai kun ohjelma ensimmäisen kerran käynnistetään. Oletuksena ensimmäisessä pelissä on helpoin vaikeustaso, joka ladataan, kun peli ensimmäisen kerran käynnistetään. Tämän jälkeen voi vaikeustasoa muuttaa milloin tahansa valitsemalla *Tiedosto*-valikosta *UusiPeli*, jolloin rivien ja sarakkeiden parametrit asettuvat sen mukaisesti, minkä tason käyttäjä valitsee. Edellämainittu kaava mahdollistaa, että molemmat korttipakat ovat aina yhtä suuret ja jaetuista korteista voidaan löytää täsmälleen kaksi samannumeroista korttia. Pelilautaluokka on toteutettu täysin irrallaan muusta toteutuksesta, jolloin ohjelman tekijällä on mahdollisuus vaihtaa käyttöliittymän ulkoasua tulevaisuudessa. Pelilautaluokka kommunikoi Kehys- ja Kortti-luokan kesken olioviitteiden välityksellä.

Korttiluokan tehtävänä on käännellä kortteja edestakaisin. Jos kaksi samaa korttia löytyy, kortin numero katoaa näytöltä eikä niitä voi enää pelata. Korttiluokka palauttaa käännetyn kortin numeron, jotta sitä voidaan verrata toisen kortin numeroon. Kun käyttäjä valitsee kolmannen kortin eivätkä aiemmin valitut kaksi korttia olleet samat, käännetään ne, jolloin numero poistuu samalla näkyviltä. Korttiluokka ei ole tietoinen muista luokista ja sen dataa käsitellään Pelilautaluokasta vain olioviitteen avulla. Käytännössä korttiluokasta muodostetaan rakenteeltaan Arraylist-taulukko.

Kehysluokka luo ohjelman käyttöliittymän ikkunan ja lisää siihen paneelit ja valikot. Kehysluokan sisällä menujen asettelu tuntui luontevimmalta ratkaisulta, koska siellä luodaan käyttöliittymän ikkuna. Menuvalintoja varten luotiin oma pieni apumetodi, jonka tarkoitus on nopeuttaa uusien menujen tekemistä. Ohjelmointityön rakenne haluttiin pitää kuitenkin mahdollisimman yksinkertaisena eikä menu- ja kehysluokista tehty erillisiä luokkia.

Luokkakaavio esitetään kuvassa 7. Kaavio muodostuu määritteistä ja operaatiosta. Määritteet ovat luokan omia muuttujia, joita käytetään ja operaatiot luokan metodeja, joiden

avulla luokkaa käytetään. Määritteet sijaitsevat kuvassa keskiviivan yläpuolella ja metodit keskiviivan alapuolella. Lohkoissa olevat etumerkinnät kertovat, ovatko muuttujat tai metodit yksityisiä (- -merkki) vai julkisia (+ -merkki). Kaaviosta voidaan myös päätellä jokaisen luokan tehtävä ohjelman kokonaisuuden kannalta sekä luokkien väliset yhteydet.



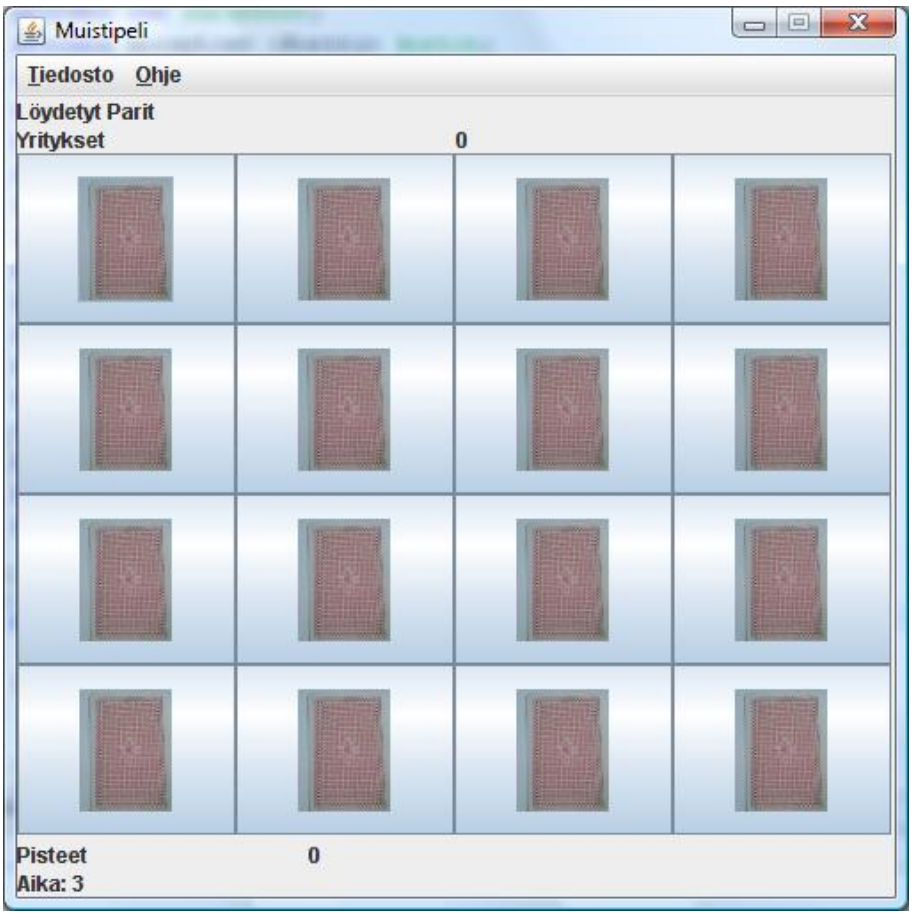
Kuva 7. Luokkakaavio

3.4 Ohjelman toiminta

Kun peli käynnistetään, muodostuu pelilauta, jossa pelikortit ovat selkäpuoli ylöspäin. Käyttäjän tehtävänä on löytää kaksi samanlaista numeroa mahdollisimman vähillä hiiren napsautuksilla. Kun käyttäjä valitsee kaksi samannumeroista korttia, ne käännetään pois näkyvistä eikä niitä voi sen jälkeen enää pelata. Pelissä on mahdollista valita kolmen eri vaikeustason väliltä, joista helpoimmassa pelilaudalle muodostetaan 16 korttia, keskivaikeassa 36 ja vaikeimmassa 64 korttia. Löydettävien parien määrä pelissä on puolet korttien kokonaismäärästä, joka saadaan kaavalla, josta jo kerrottiin luvussa 3.3.

Peli päivittää koko ajan löydettyjen parien ja yritysten lukumäärää, peliaikaa sekä pistemäärää. Ne näkyvät näytön tietoalueilla käyttäjälle, ja ne ovat *Jlabelilla* toteutettuja. Kun peli päättyy, ilmestyy ruudulle viesti, että peli päättyi, ja samalla myös peliaika pysähtyy. Kuluva peliaika lähtee päälle heti, kun peli käynnistetään tai valitaan uudelle pelille vaikeustaso. Peliaika esitetään ruudulla sekunteina. Pelikortit on toteutettu *Jbuttonista* periyttämällä sen ominaisuudet kortille. Varsinainen pelilauta toimii ohjelman keskeisimpänä luokkana, jossa pääohjelma sijaitsee ja josta muita luokkia kutsutaan ilmentymien avulla.

Käyttöliittymä ja sen elementit esitellään kuvassa 8. Ohjelmasta löytyy kaksi valikkoa, *Tiedosto* ja *Ohje*, joista käyttäjä voi valita haluamansa toiminnon. Valittavia toimintoja *Tiedosto*-valikossa ovat *UusiPeli*-valinta, josta avautuu alivalikko, josta voidaan valita vaikeustaso. Tämän lisäksi voidaan tutkia myös pelin parhaita tuloksia. Menun viimeisenä toimintona on Lopeta, jonka valitsemalla peli sulkeutuu. *Ohje*-valikosta löytyvät peliin liittyvät käyttöohjeet sekä tietoja muistipelistä. *Tietoja Ohjelmasta* avaa oman dialogi-ikkunan, joka sisältää pelin versionumeron sekä tietoja pelin tekijästä. Käyttöohje sijaitsee omassa HTML-tiedostossa, joka avautuu näytölle, kun se valitaan. Pelissä on myös pikanäppäimet, joilla peliä voidaan ainakin valikoiden osalta käyttää. Varsinainen pelaaminen vaatii kuitenkin hiiren.



Kuva 8. Käyttöliittymä ja sen elementit

4 Toteutus

Seuraavaksi käsitellään hieman sitä, miten koodausvaihe toteutettiin. Työssä käytettiin NetBeans IDE 6.1 -versiota, johon ladattiin kaikki mahdolliset päivitykset. Ohjelmassa olevia valmiita Swing-käyttöliittymäkirjastoja hyödynnettiin mahdollisimman paljon tehtäessä käyttöliittymäelementtejä. Koodaustyö toteutettiin käsin kirjoittamalla, koska valmiiksi NetBeansilla generoituja luokkia ei osattu hyödyntää liian vähäisen ohjelmointikokemuksen takia. Ne antoivat toki vinkkejä, miten ohjelmointi kannattaisi tehdä. Varsinainen ohjelmointityö toteutui kuitenkin määrittely- ja suunnitteludokumentaatiossa kuvatulla tasolla, vain luokkakaavioon täytyi tehdä muutoksia. Muuttujia ja metodeja jouduttiin lisäämään sitä mukaa kuin ohjelmointityö eteni. Viimeisin luokkakaavio, jossa on mukana kaikki viisi luokkaa, esitellään liitteessä 1. Kaavio antaa hieman vertailupohjaa, miten työ muuttui toteutusvaiheessa.

Koodi pyrittiin pitämään siistinä ja ylimääräisiä ohjelmointikikkailuja pyrittiin välttämään. Koodia kommentoitiin sitä mukaa kuin sitä kirjoitettiin, jotta ajatukset pysyivät kasassa ja muistaminen helpottuisi. Tässä luvussa käsitellään, mitä toteutusvaiheessa saatiin aikaiseksi sekä käydään läpi muutama ohjelman keskeinen rakenne.

4.1 Yleistä graafisesta ohjelmoinnista

Javassa käyttöliittymä toteutetaan olioiden avulla. Apuna käyttöliittymän teossa voidaan käyttää esimerkiksi Swingin luokkakirjastoa, jonka avulla on mahdollista saada aikaan näyttäviä ohjelmistoja. Graafiset ohjelmat perustuvat aina erilaisiin tapahtumiin (event based). Kun hiirellä painetaan jotain komponenttia tai valitaan se valikosta, suoritetaan aina jokin tapahtuma. Työssä päädyttiin graafiseen ohjelmaan, koska nykyisin lähes kaikki ohjelmat tehdään graafisina sovelluksina. Graafisten ohjelmien käyttö on helpompaa kuin merkkipohjaisten ohjelmistojen, koska graafisessa kaikki voidaan valita hiiren avulla, kun taas merkkipohjaisessa kaikki käskyt pitää kirjoittaa komentokehotteessa. Graafisten ohjelmien tekeminen on kuitenkin huomattavasti vaikeampaa, koska pitää huomioida kaikki mahdollinen toiminta, mitä hiirellä tai näppäimistöllä voidaan

tehdä. Ohjelma toimii siihen saakka, kunnes käyttäjä päättää sulkea ohjelman joko valikon kautta tai painamalla ohjelman yläkulmassa olevaa rastia. Ohjelma ei etene suoraviivaisesti alusta loppuun, kuten se eteni joskus komentokehoteperustaisissa ohjelmistoissa. Näin ohjelman koodaaminen ja toimintojen seuraaminen on paljon monimutkaisempaa kuin ei-graafisissa ohjelmistoissa. Ohjelma käyttää paljon olioilmentymiä, ja niiden ymmärtäminen on aluksi hankalaa. Samoin laaja Swing-kirjasto tuottaa aluksi vaikeuksia, kun pitää osata valita omaan ohjelmaan sopiva komponentti.

Erilaiset ohjelmassa mahdollisesti sattuvat virhetilanteet pitää osata käsitellä jo koodausvaiheessa poikkeusten käsittelyn avulla, muuten kääntäjä ei hyväksy koodia. Ohjelman toipumisen erilaisista ennalta-arvattavista tai arvaamattomista virhetilanteista toteutetaan try-catch -rakenteiden avulla. Onneksi nykyiset ohjelmointityökalut ohjaavat ohjelmoijaa tekemään koodiin korjauksia antamalla pieniä vinkkejä ohjelmaa kirjoitettaessa. Tämä edellyttää, että ohjelman asetuksista on laitettu oikeat rastit päälle, jotta vinkit tulevat näytölle. Mikäli osa vinkeistä jää ratkaisematta, ohjelma saattaa kuitenkin toimia. Tällöin on usein kyseessä vain suositus tehdä jokin asia uuden Java-version edellyttämällä tavalla tai koodissa on vain pieni kosmeettinen vika.

4.2 Työssä käytetyt sijoittelumanagerit

Javassa komponenttien sijoitteluun voidaan käyttää olemassa olevia valmiita sijoittelumanagereita. Niiden tehtävänä on helpottaa käyttöliittymässä tarvittavien komponenttien sijoittelua. Sijoittelumanagereita on mahdollista sijoittaa myös sisäkkäin, jolloin niillä voidaan asemoida komponenttien paikat entistä tarkemmin. [8, s. 321.]

Työssä käytettiin BorderLayout-manageria yhdessä GridLayout-managerin kanssa. BorderLayout -managerilla luotiin pohjapaneeli. Tämän jälkeen GridLayoutilla voitiin tarkentaa sijoittelua BorderLayoutin sisällä. Näiden sijoittelumanagerien avulla saatiin komponentit järjesteltyä ruudulle. GridLayoutilla jaettiin pelilauta ruudukoksi, jossa kaikki ruudut ovat samankokoisia. GridLayout metodin parametrit suluissa kertovat rivien ja sarakkeiden määrän, joiden avulla lasketaan tarvittavien pelikorttien määrä. BorderLayoutin avulla komponentit saadaan järjestettyä eri ilmansuunnittain. Esimerkki 1 havainnollistaa paneeleiden layoutia ja asettelua. Kun luodaan Layout, voidaan samassa yhteydessä määrittää ja välittää parametrit, jotka vaikuttavat komponentin asemointiin ruudulla. Esimerkiksi (1,1) tarkoittaa yhtä riviä ja yhtä saraketta, eli kyseinen tila varataan tässä tapauksessa alapaneelia varten ja alapaneeli asetetaan BorderLayoutin avulla etelää kohti (SOUTH).

Esimerkki 1. Paneelien layout ja asettelu.

```
 // Tämä koodi suoritetaan kehysluokan alustamisessa
    private JPanel pohjapaneeli=new JPanel(new BorderLayout());
    private JPanel alapaneeli=new JPanel(new GridLayout (1,1));
    private JPanel ylapaneeli=new JPanel(new GridLayout(2,2));

//Asetetaan paneelit oikeisiin paikkoihin, kehysluokan konstruktorissa
oleva koodi
    pohjapaneeli.add(alapaneeli, BorderLayout.SOUTH);
    pohjapaneeli.add(ylapaneeli, BorderLayout.NORTH);
    pohjapaneeli.add(lauta, BorderLayout.CENTER);
    this.add(pohjapaneeli);
```

4.3 Säikeet

Javassa moniajon käyttö onnistuu käyttämällä säikeitä. Usein niitä käytetään, jos ruudulla tapahtuu jotain muutaman millisekunnin välein ja halutaan estää ohjelman mahdollinen kaatuminen. Tässä työssä ajastin on sijoitettu säikeen sisälle. Yleensä säikeet mahdollistavat myös useiden eri tehtävien ajamisen rinnakkain. [9.]

Tässä projektin ohjelmassa on käytössä vain yksi säie, joka luotiin lähinnä harjoitusmielessä. Sen hyötynäkökohta on se, että vaikeustasoa voidaan pelattaessa muuttaa ilman, että se aiheuttaa ohjelman käytössä ongelmia. Ilman säiettä ohjelma voisi kaatua jossain

tilanteessa, kun samaa muuttujaa pääsee samaan aikaan operoimaan jokin toinen olio, mutta tässä se on estetty säikeen avulla. Säie ei päästä toista oliota samaan aikaan käsittelemään samaa oliota tai muuttujaa, koska muuten voi ohjelma lukkiutua ja kaatua. Tässä yhteydessä puhutaan ohjelman lukkiutumisesta (”deadlock-ilmiöstä”). Säikeiden käyttöä tukevat nykyiset moniprosessorilla varustetut tietokoneet. Myös vanhemmissa laitteissa on mahdollista käyttää säikeitä, mutta todellista moniajoa niillä ei tietenkään voida suorittaa. Tässä työssä on käytetty yhtä säiettä, joka on käynnissä koko ohjelman suorituksen ajan ja sen toteuttaa Runnable-interface. Luokan olennaisin metodi on run(), josta säikeen käynnistäminen tapahtuu. Se on korvattava omalla koodilla, mikäli haluaa sen suorittavan jotain. Toinen mahdollisuus, jolla säie on mahdollista toteuttaa, on periyttää se suoraan Thread-luokasta. [9.]

Ohjelmassa halutaan, että yksi säie on vuorollaan suorituksessa ja sen vuoksi käytetään metodia *synchronized.* Tämä mahdollistaa sen, että säiettä ei voi käyttää kuin yksi olio kerrallaan. Tässä työssä synkronointi kohdistuu ainoastaan main-metodiin. Säie käynnistetään run() -metodin avulla (katso esimerkki 2), ja se on toiminnassa koko ohjelman suorittamisen ajan. Säikeet huolehtivat siitä, että samojen olioiden ollessa käytössä ei mitään virhetilanteita pääsisi muodostumaan. Jos samaa säiettä yrittää käyttää jokin toinen olio, olio joutuu jonotuslistalle. Se pääsee suoritukseen vasta, kun toinen olio on suorittanut omat tehtävänsä. [9.]

Esimerkki 2. Säikeistys main() metodissa.

```
  Runnable runner = new Runnable () {
    // säie alkaa
    public void run() {
Runnable runner = new Runnable () {
    // jonotuslista
       EventQueue.invokeLater(runner);
```

4.4 Tapahtumankuuntelijat

Ohjelmassa tarvitaan erilaisia tapahtumankuuntelijoita, joiden avulla voidaan tarkkailla käyttäjän valitsemia toimintoja ja reagoida niihin oikealla tavalla. Yleensä tarvitaan metodi, jota kuunneltava komponentti kutsuu, kun ohjelmassa tapahtuu jotain. Tapahtumankuuntelijametodi on muotoa addActionListener(new ActionListener()). Tapahtu-

mankuuntelijametodien lisäksi tarvitaan tapahtumankäsittelymetodeja. Niitä tarvitaan kaikissa tapahtumissa joihin ohjelman halutaan reagoida. Tapahtumankuuntelija reagoi esimerkiksi, kun käyttäjä valitsee jonkin toiminnon valikosta tai napsauttaa jotain komponenttia ohjelman ikkunasta. [10, s.236, s.239.]

Tapahtumankäsittelymetodi ottaa vastaan kaikki tapahtumat ja reagoi niihin suorittamalla jonkin sille määrätyn koodinpätkän [10, s.239]. Tämä voidaan toteuttaa esimerkiksi public void actionPerformed(ActionEvent e) -metodin avulla. Seuraavaksi tarkastellaan esimerkkiä 3, jossa käyttäjä valitsee valikosta uuden pelin. Tähän tapahtumaan reagoi tapahtumankuuntelija, joka kutsuu tapahtumankäsittelijä-metodia, joka puolestaan kutsuu uusiPeli()-metodia. Näin käynnistyy uusi peli.

Esimerkki 3. Valikon tapahtumankuuntelija

```
menuTiedostoUusiPeliHelppo.addActionListener(new ActionListener() {
//Lisätään tapahtuman kuuntelijaolio ja sen käsittelijämetodi
        public void actionPerformed(ActionEvent e) {
          uusiPeli(); }}
```

4.5 Käytetyt tietorakenteet

Tässä työssä käytettiin etupäässä valmiita luokkakirjastoja, koska ei ole järkevää keksiä jo keksittyjä asioita. Sanotaankin, että pyörää ei kannata keksiä uudelleen. Ensimmäinen tietorakenne korttiluokan käyttö toteutettiin periyttämällä se *Jbuttonista*, koska se tuli ensimmäisenä mieleen. Paremmin tämän olisi voinut toteuttaa *Jlabelilla*.

Korttien luomisessa käytettiin rajoittamatonta taulukkoa, joka toteutettiin luokan *ArrayList<Kortti>* avulla, jossa alkioiden indeksointi tapahtuu automaattisesti. Tämä rakenne mahdollistaa sen, että alkioiden määrää voidaan kasvattaa joustavasti ohjelmassa tarpeen mukaan. Tätä rakennetta pystyttiin hyödyntämään korttien määrän kasvatuksessa vaikeustasojen valinnan yhteydessä. Tallennettavan alkion tyyppi on Kortti-olio. Rajattoman taulukon luomisesta kertoo esimerkki 4, jossa aluksi luodaan tyhjä Arraylist-olio, jonka alkioksi kelpaavat vain Kortti-oliot. Sitten luodaan pelikortit kahteen kertaan, jotta saadaan kaksi samalla numerolla varustettua korttia. Korttien kokonaismäärä saadaan pelilaudan luomisen yhteydessä, kun käyttäjä valitsee vaikeustason eli kaavalla

rivit x sarakkeet/2. Tämän jälkeen kortit sekoitetaan *Shuttle*-metodin avulla ja ne lisätään peliin kortti kerrallaan ja asetetaan GridLayoutin mukaisesti ruudulle. Samalla korteille asetetaan tapahtumankuuntelijat. Toistettava algoritmialkio saa for-lauseessa kaikki kortit-taulukon alkioiden arvot yksitellen.

Esimerkki 4. Rajaton taulukko ja korttien sekoitus.

```
    ArrayList <Kortti>kortit=new ArrayList<Kortti>();
//korttienmäärä silmukalla
    for(int i=0; i<korttienIDMaara; i++) {
//Luodaan kaksi korttioliota, jotta saadaan parit muodostettua
      kortit.add(new Kortti(i));
      kortit.add(new Kortti(i));
// korttien ID numero kehykseen
      kehys.add(kortit.get(i));
    }
// korttien sekoitusmetodi
    Collections.shuffle(kortit);
// korttien lisäämissilmukka
    for(Kortti alkio:kortit) {
// tapahtuman kuuntelijan lisääminen
      alkio.addActionListener(this);
// kortin lisääminen
      this.add(alkio);
….};
```

Parhaiden pistemäärien tietorakenne toteutettiin vastaavanlaisella ArrayList-rakenteella kuin pelikortin tietorakenne. Tässä rakenteessa esitellään tulostaulun läpikäyntimetodi esimerkki 5, jossa tulosten läpikäynnistä huolehtii iteraattori.

Toinen esimerkki tietorakenteesta, esitetään esimerkissä 6. Se on metodi *tallennaRivitTiedostoon*. Tämä metodi huolehtii PeliTulosTietueen tallennuksesta suoraan tiedostoon. Jos tiedostoa ei vielä ole olemassa, se luodaan ensimmäisen pelin yhteydessä.

Esimerkki 5. Tulostaulun läpikäynti.

```
public Iterator<PelitulosTietue> iterator() {
Ar-
rayList<PelitulosTietue>PelitulosTietueet=newArrayList<PelitulosTietue
>();
//Luodaan pelitulostietue-oliota varten luodaan taulukko
        for(int i = 0; i < taulukko.length; i++) {
            PelitulosTietueet.add(taulukko[i]);
           //lisätään tietueet taulukkoon [i]
        }
        return PelitulosTietueet.iterator();
    }
```

Esimerkki 6. Tallenna rivit tiedostoon.

```
public boolean tallennaRivitTiedostoon() {
        try {
            FileWriter tiedosto = new FileWriter(new
File(tiedostonNimi));
//luo uuden tiedoston kirjoittamista varten tai avaa olemassa olevan
tiedoston kirjoittamista varten
            BufferedWriter kirjoittaja = new BufferedWriter(tiedosto);
//Tietojen tallennus puskuroituna, kytketty FileWriter-olioon
            for(int i = 0; i < taulukko.length; i++) {
//Kirjoittaa taulukon PelitulosTietue-oliot tiedostoon, yksi per rivi.
                if(taulukko[i] != null) {
                    kirjoittaja.write(taulukko[i] + "");
/Tietueet kirjoitetaan stringinä
                    kirjoittaja.newLine();
//seuraavalle riville siirtäminen
                }
            }
//Kun kaikki tarvittava tieto on kirjoitettu, kirjoittaja suljetaan.
            if(kirjoittaja != null) {
                kirjoittaja.close();
            }
        }
        catch(IOException poikkeus) { //poikkeuksen käsittely
            return false; //kirjoittaminen epäonnistui
        }
        return true; //Kirjoittaminen onnistui
    }
```

4.6 Versiot ja lopullinen luokkakaavio

Työstä tehtiin kaksi eri versiota. Ensimmäisessä versiossa oli mukana ajastin ja vaikeustasojen valinta. Helpommalla tasolla on pelattavissa 16 korttia, keskivaikeassa 36 korttia ja vaikeimmassa tasossa 64 korttia. Ainoana puutteena ohjelmassa oli, ettei parhaiden pelaajien tuloksia voinut tallentaa tiedostoon, ja eikä niitä voinut myöskään tarkastella pelin jälkeen. Tällä versiolla pelattavuus kärsi, koska tuloksen joutui pitämään mielessä tai kirjaamaan muistilapulle.

Toiseen versioon luotiin tietorakenne, joka mahdollisti parhaiden aikojen tallennuksen ja niiden näyttämisen. Parhaat ajat ja pelaajan nimi kerätään tietokantaan, jota käyttäjä voi tarkastella myöhemmin. Näin pelistä saatiin aikaiseksi miellyttävämpi pelikokemus, koska se mahdollisti pelikilpailun järjestämisen vaikkapa perheen tai ystävien kesken. Näin kaikki ohjelmalle määrittelydokumentissa asetetut tavoitteet saavutettiin version 2.0 myötä. Ainostaan parhaiden pistemäärien tallennuksesta luovuttiin, koska pelkkä ajanlaskenta riittää parhaiden tulosten määrittämisessä. Tämä ei huononna ohjelmaa

kuitenkaan millään tavalla. Ohjelman luokkakaavioon tehtiin paljon lisäyksiä ja siihen lisättiin kaksi uutta luokkaa. Luokkakaavion viimeinen versio esitetään liitteessä 1.

5 Testaus

Ohjelman testaus tapahtui rinnakkain ohjelman teon aikana. Löytyneet virheet korjattiin sitä mukaa kuin niitä löytyi kääntäjän suorittaman yksikkötestin toimesta tai peliä pelattaessa. Kun virheet saatiin korjattua, oli mahdollista rakentaa peliin lisäominaisuuksia pala kerrallaan. Graafisen ohjelman testaus jokainen luokka erikseen ilman muiden luokkien läsnäoloa olisi täysin mahdotonta. Pelissä käyttöliittymän näytölle muodostaminen tapahtuu kehysluokan avulla. Ilman sitä ei näytöllä nähtäisi mitään ja testaaminen ilman näkyviä osia olisi mahdotonta. Lisäksi luokat kommunikoivat hyvin aktiivisesti toisten luokkien kanssa eikä niiden toimintaa voida erikseen tarkastella. Korttiluokan testaaminen yksinään on täysin tarpeetonta, koska se sisältää vain perusmetodeja ja niiden toiminta voidaan testata samalla, kun pelilautaluokkaa testataan.

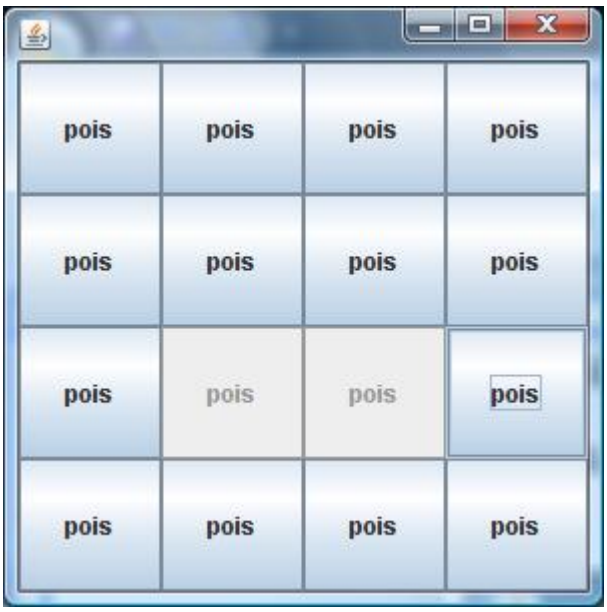
Koodaamisen yhteydessä NetBeans tarjoaa mahdollisuuden myös luoda ja tallentaa *JUnit* -testejä jo ohjelman teon alusta alkaen. Näin ollen hyvinkin monimutkaisia rakenteita pääsee heti alusta alkaen testaamaan. Kun testiä ajetaan aina uudelleen ja uudelleen, huomataan, että virheet ovat vähentyneet ja lopulta päädytään aivan yllättäen toimivaan ohjelmaan. Ohjelman valmistuessa testit eivät löydä enää virheitä.

5.1 Kehysluokka

Kehysluokkaa testattiin perinteisellä keinolla lisäämällä yksikkötestausta varten koodin loppuun *main*-metodi. Kaikki kehysluokassa luodut valikot ja niihin liittyvät toiminnot sekä *Jlabeleiden* päivitykset testattiin. Muiden luokkien aiheuttamiin toimintoihin ei tässä kohtaa kiinnitetty huomioita, vaikka osa toiminnoista vaatii toisen luokan läsnäoloa varsinkin, kun se käyttää toisen luokan metodeja ilmentymien välityksellä. Kortteja voidaan testissä napsautella ja havaita, että napsauttamalla kahta korttia päivittyvät yritysten määrä, löydettyjen parien määrä sekä pistemäärä. Tällä testillä saatiin selville se, että kaikki kehysluokan toiminnot toimivat odotetulla tavalla. Myöhemmässä vaiheessa lisättiin vielä ajanotto *Jlabeliin*. Ajanottoa testattiin ohjelman kaikkien luokkien kanssa.

5.2 Pelilauta- ja Korttiluokat

Seuraavaksi Pelilauta- ja Kortti-luokkia testattiin yhtä aikaa. Testi syntyi aivan ohjelman teon ensi versiossa, jossa ei vielä kehysluokkaa ollut olemassa. Versiossa ladattiin vain Pelilauta ja aseteltiin kortit sinne. Testaus tapahtui pelaamalla peliä ja valitsemalla hiirellä kaksi satunnaista korttia pelilaudalta. Tässä esiversiossa laatan *Jlabelissa* olevat toiminnot olivat teksti näkyvillä, pois teksti ei näkyvillä ja viimeisenä teksti pois on harmaa. Kuva 9 havainnollistaa edellä mainittuja seikkoja. Pois-teksti kuvaa kortin selkäpuolta ja pois teksti harmaa sitä, että kortti on poistettu pelistä, ja numero näkyvillä sitä, että kortti on käännettynä numeropuoli ylös. Tämä oli ensimmäisiä versioita, joissa vielä voidaan havainnollistaa testaaminen näiden kahden luokan välillä. Myöhemmät testit tapahtuivat, kun ohjelmaan lisättiin kehysluokka, jolloin kolmea luokkaa testattiin periaatteessa vain kääntäjän toimesta.



Kuva 9. Pelilaudan testiversio

Pelilaudassa *main*-metodi kutsuu pelilaudan konstruktoria ja samalla sille välitetään parametreina rivien ja sarakkeiden määrä. Näiden perusteella kontruktori osaa luoda pelissä tarvittavat kortit. Pääohjelma asettaa ikkunan näkyville konstruktorissa, jolloin myös kortit asetetaan asemoinnin mukaisesti ruudulle näkyviin. Pelilaudan kortit lasketaan kontruktorissa kaavalla ja niitä saadaan 16 kappaletta sen helpoimmalle tasolle. Ne

ovat eri numerolla varustettuja. Jotta saataisiin kaikkien korttien numerot esiintymään kahdesti, luodaan kortti-ilmentymä kahteen kertaan ja lisätään siten saadut kortit *ArrayList*-taulukkoon. Näin saadaan täsmälleen kaksi samalla numerolla varustettua korttia toteutettua.

Testin tuloksena havaittiin, että ikkunan koon pienentäminen piilotti aluksi numerot eikä pelaaminen siksi onnistunut. Ongelma ratkaistiin estämällä ikkunan koon muuttaminen kokonaan käyttäjältä. Toisen ongelman aiheutti näytön resoluutio. Tähän ei ollut muuta ratkaisua kuin kehottaa käyttäjää muuttamaan näytön resoluutioksi vähintään 1280 x 800 pikseliä. Kannettavilla minitietokoneilla ei resoluutio riitä, joten peliä ei suositella pelattavaksi niillä lainkaan. Kolmannen ongelman aiheutti se, että napsauttamalla samaa korttia useamman kerran häipyi se pelattavista. Virhe löytyi korttiluokan switch-lauseesta ja se korjattiin. Muita virheitä ei tässä vaiheessa havaittu.

5.3 Pelilauta-, kortti- ja kehysluokat, tulostaulu ja pelitietue

Tulostaulun ja pelitietueen testaaminen toteutettiin ainoastaan Java-kääntäjän toimesta. Kun kääntäjä ei löytänyt niistä virheitä, suoritettiin niiden integroiminen muihin luokkiin. Lopuksi testattiin ohjelma kaikkien luokkien kanssa ja tehtiin tarvittavat muutokset koodiin.

Aluksi virheitä integroinnin jälkeen löytyi paljon, mutta tulostaulun ja pelitietueen toimimattomuus ei onneksi haitannut itse pelin suoritusta, vaan näitä kahta luokkaa tarvittiin vasta, kun peli oli pelattu loppuun. Näiden kahden luokan tarkoituksena oli tallentaa pelin aika, pelaajan nimi sekä päivämäärä ja kellonaika, jolloin peliä pelattiin. Tässä vaiheessa tietueen tallennusvaiheessa syntyi paljon ajonaikaisia virheitä, joiden korjaaminen vaati monia muutoksia ohjelman muihin luokkiin. Tulostaulun ja pelitietueen toimintaa ei olisi voitu varmistaa ilman muiden luokkien läsnäoloa.

Peliä testattiin kaikkien luokkien kanssa pelaamalla ohjelmaa läpi. Kokeiltiin kaikkia toimintoja ja samalla käynnisteltiin myös muita Windows-ohjelmia, jotta nähtäisiin,

kaatuisiko peliohjelma vai ei. Myös eri resoluutioilla kokeiltiin ja päädyttiin samaan kuin edellisen kappaleen testausvaiheessa eli resoluutio tulee asettaa 1280 x 800 pikselin suuruiseksi. Toisaalta, jos haluttaisiin pelin toimivan myös pienellä näytöllä, ei ikkunan koon muuttamista saisi estää koodista. Tosin todennäköisempää on se, etteivät kaikki käyttäjät osaa muuttaa pelialueen ikkunan kokoa ennen pelin aloittamista. Niinpä ikkunan koko jätettiin estetyksi ohjelman lopullisessa julkistettavassa versiossa 2.0. Lopputuloksena saatiin toimiva ohjelma, joka täytti määrittely- ja suunnitteluvaiheessa esitetyt vaatimukset.

6 Käyttöohje

Käyttöohje kertoo ohjeet pelaajalle, miten ohjelmaa käytetään. Ohje kertoo pelin käyttötarkoituksen, kuinka peli käynnistetään sekä laitteistolle asetettavat vaatimukset. Ohjetta pääsee tarkastelemaan milloin tahansa pelin aikana. Ohjeessa kuvataan pelin kannalta keskeisiä toimintoja. Tämän luvun runko perustuu Jaakko Nenosen [6] laatimaan ohjelmoinnin harjoitustyön ohjeistukseen.

6.1 Pelin käyttötarkoitus

Muistipeli on graafinen älypeli, jonka tarkoituksena on kehittää ihmisen muistia ja toimia ajanvietteenä niin aikuisille kuin lapsille. Tässä pelissä ei ole pelaamiselle asetettu mitään ikärajoja.

6.2 Ohjelman käynnistäminen

Ohjelmisto toimitetaan kokonaisena pakettina, jonka nimi on *Muistipeli.jar*. Ohjelmistopaketti sisältää kaikki pelissä tarvittavat luokkatiedostot eli *Pelilauta.class, Kehys.class, Kortti.class, PelitulosTietue.class* ja *Tulostaulu.class*. Mukana tulee myös käyttöohjetta varten oma tiedosto, joka on nimeltään *ohje.html* sekä korteissa käytetty kuvatiedosto *Kortti.jpg*. Edellä mainittujen tiedostojen tulee sijaita samassa hakemistossa pelitiedostojen kanssa.

Peli voidaan käynnistää graafisesta käyttöliittymästä (Windows tai Linux) tuplanapsauttamalla *jar*-pakettia, jolloin peli aukeaa. Komentokehotteesta ohjelma voidaan käynnistää komennolla *java –jar Muistipeli.jar*. Esimerkiksi komentokehotteeseen Windows XP:stä pääsee valitsemalla Käynnistä/Start ja Suorita/Run ja kirjoittamalla avautuvaan ikkunaan *cmd*-komento, jolloin komentokehoteikkuna avautuu. On huomattavaa, ettei peli käynnisty Windows Vista käyttöjärjestelmässä muuten kuin suorittamalla se ko-

mentokehotteesta järjestelmänvalvojan oikeuksilla. Haluttaessa lisätietoa komentokehotteen käytöstä, tutustu oman käyttöjärjestelmän ohjetoimintoon.

6.3 Laitteistovaatimukset

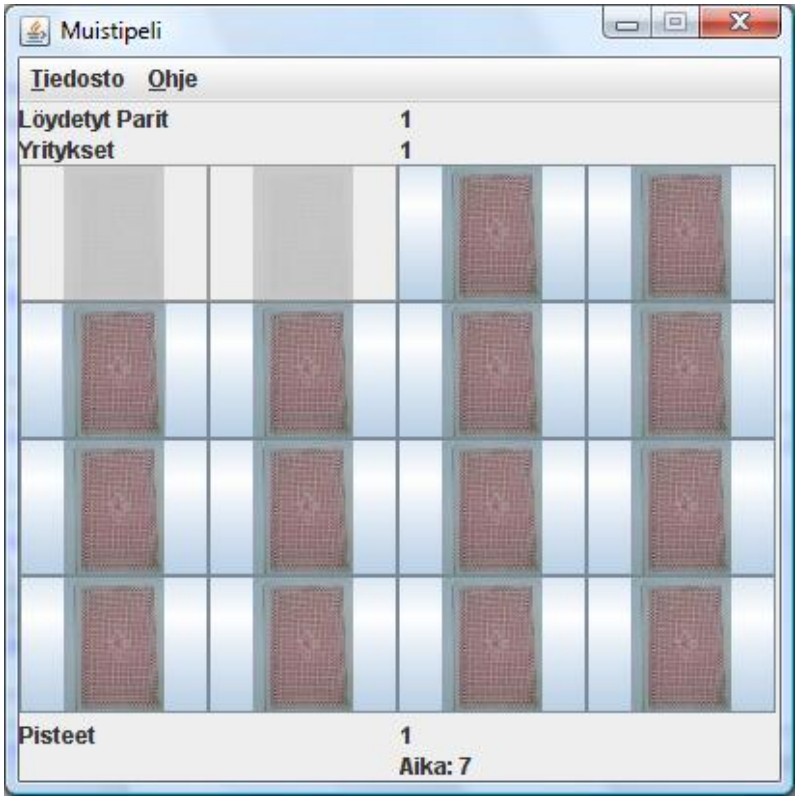
Muistipeli vaatii graafisen käyttöjärjestelmän, esimerkiksi. Windowsin tai Linuxin. Koneeseen tulee olla asennettuna Java Standard Edition 6.0 (SE) tai uudempi versio. Tämän voi ladata Sunin kotisivuilta ilmaiseksi osoitteesta http://www.sun.com/. Näyttöasetuksissa riittää 16-bittiset värit. Näyttöasetuksissa suositellaan resoluution tarkkuudeksi vähintään 1280 x 800 kuvapistettä tai suurempi, jotta peli näkyisi ruudulla kokonaisuudessaan. Näytöksi sopii parhaiten 15 tuuman näyttö tai suurempi. Pelikoneeksi kelpaa tietokone, jossa käyttöjärjestelminä on vähintään Windows XP, Windows Vista tai Linux. Ohjelmaa testattiin kaikilla edellä mainituilla käyttöjärjestelmillä. Pienillä näytöillä varustetuissa tietokoneissa ja liian pienellä resoluutiolla havaittiin, ettei korttien numero tule näkyville kaikilla vaikeustasoilla. Asia korjautuu muuttamalla resoluutiota suosituksen mukaiseen arvoon.

6.4 Pelin käyttöliittymä ja toiminnot

Kuva 10 näyttää, millaiselta peliohjelman graafinen käyttöliittymä näyttää. Keskellä sijaitsee varsinainen peliareena, jossa pelikortit sijaitsevat. Pelissä kortit reagoivat käyttäjän hiiren napsautuksiin kääntämällä kortteja näytöllä vuoroin esille tai piiloon. Pelin käyttöliittymä muodostuu pelialueesta ja kahdesta tilatietoalueesta. Toinen tilatietoalue on pelin ylälaidassa ja toinen alalaidassa. Molemmilla tilatietoalueilla näkyy tietoja pelin tilasta. Pelin tilarivi pitää käyttäjän ajan tasalla pelitilanteesta kertomalla löydettyjen parien määrän, yritykset, senhetkisen pistetilanteen sekä pelissä sillä hetkellä käytetyn peliajan.

Kun pelaaja on kääntänyt kaksi korttia, katsotaan ne ensimmäiseksi yritykseksi löytää pari. Samalla periaatteella myös löydettyjen parien määrä lasketaan kasvavaksi, mikäli oikea pari löytyy. Alapuolisella tilatietoalueella näkyy puolestaan kertynyt pistemäärä.

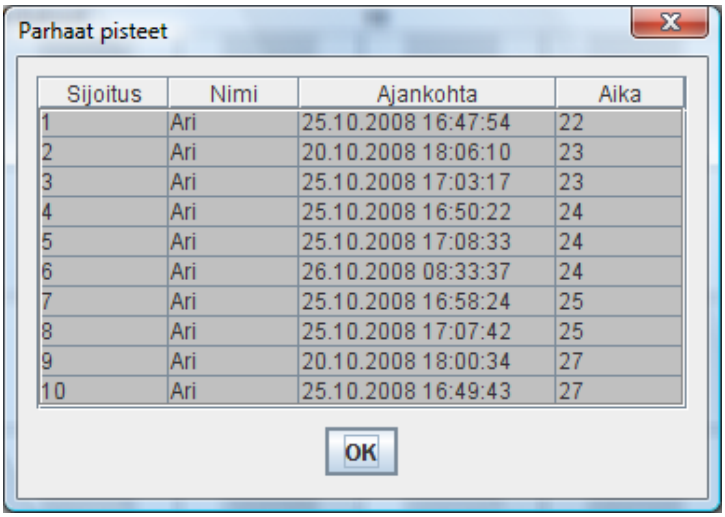
Alkupistemääräksi on asetettu nolla pistettä, ja se kasvaa aina yhden pisteen sitä mukaa, kun parin arvauksissa onnistutaan. Keskellä olevalle pelialueelle ohjelma sijoittaa pelattavat kortit ja niiden tilaa päivitetään sitä mukaa kuin peli etenee. Löydetyt parit näkyvät pelissä harmaana eikä pelialueella olevia kortteja voi enää pelata.



Kuva 10. Pelin käyttöliittymä

Valikoiden toimintoihin pääsee hiiren lisäksi myös näppäimistön avulla tai käyttäen pikanäppäimiä. Pikanäppäimet näkyvät valikoissa esiintyvistä alleviivauksista. Ohjelmassa on kaksi valikkoa eli *Tiedosto* ja *Ohje*. Tiedosto-valikkoon pääsee hiirellä tai näppäinyhdistelmällä *Alt+t*. Tiedostovalikosta voidaan valita *UusiPeli*, jos halutaan pelata peli uudelleen. Uuden pelin valinnan yhteydessä tulee valita myös vaikeustaso. Valittavia tasoja on kolme eli *helppo, keskivaikea* tai *vaikea*. Kun uusi pelilauta luodaan, sekoitetaan pelikortit ja jaetaan ne pelialueelle. Samalla nollataan löydettyjen parien, yritysten, ajan sekä pistemäärän tilanne. Pelissä olevia parhaita pisteitä pääsee tarkastelemaan valitsemalla *Tiedosto*-valikosta *Parastulos*, jolloin avautuu kuvan 11 mukainen ikkuna ruudulle. Se kertoo pistemäärien sijoituksen, pelaajan nimen, ajankohdan,

jolloin peliä pelattiin, ja ajan, kuinka monta sekuntia kului pelaamisessa. Painamalla *OK*-painiketta ikkuna sulkeutuu. *Tiedosto*-valikon *Lopeta*-valinnalla voidaan koko ohjelman suoritus päättää.



Kuva 11: Parhaat pisteet

Ohje-valikosta löytyvät pelin *Käyttöohje* ja *Tietoja Ohjelmasta*. Näitä molempia voidaan tarkastella milloin tahansa pelin aikana. Käyttöohjetta tarkasteltaessa voidaan tekstiä vierittää ruudulla näkyvästä vierityspalkista. *Tietoja Ohjelmasta* -valinta avaa viesti-ikkunan, jossa on tietoja pelintekijästä ja sen versionumerosta. Painamalla *OK*-painiketta viesti-ikkuna sulkeutuu. Oikeassa yläkulmassa sijaitsevat tyypilliset Windows-ohjelman painikkeet (kuva 12). Ensimmäinen on pienennä-painike, keskimmäinen suurenna-painike ja oikeanpuoleisin sulje-painike. Pienennä-painikkeesta peli menee käyttöjärjestelmän tehtäväpalkkiin pieneksi ikoniksi, josta napsauttaen se aukeaa takaisin pelattavaksi. Oikealla puolella olevaa rastia painamalla peli voidaan myös lopettaa. Keskimmäisen painikkeen käyttö pelissä on estetty, eikä käyttäjä voi muuttaa pelin kokoa lainkaan.

Kuva 12: Pienennys-, palautus- ja sulje-painikkeet

6.5 Pelaaminen

Muistipelissä esiintyvät pelattavat kortit sijaitsevat keskellä pelialuetta. Pelaaja valitsee napsauttaen jotain haluamaansa korttia, jolloin se kääntyy ympäri ja piilossa oleva numero tulee näkyville. Sen jälkeen käyttäjä valitsee toisen kortin vastaavasti. Kun pelaaja on löytänyt kaksi samaa numeroa, poistuvat kortit pelattavista. Ne muuttuvat harmaiksi eikä niitä voi enää pelata. Mikäli löytyi pari, se näkyy ruudulla kohdassa löydetyt parit. Kun kaikki parit ovat löytyneet, peli päättyy. Pelin päättyessä ruudulle tulee ilmoitus, että peli päättyi, ja peliaika pysähtyy. Tämän jälkeen suoritetaan pistemäärän tarkistaminen. Jos pelitulos oli riittävän hyvä, on mahdollista päästä parhaiden pelaajien listalle. Silloin avautuu ikkuna, jonne käyttäjä voi syöttää nimensä, jonka jälkeen näytetään päivitetty parhaiden pelaajien lista (katso kuva 11). Jos tulos ei riittänyt listalle pääsemiseen, ei ruudulle avaudu ikkunaa.

Kortteja voi olla vain kaksi käännettyinä kerrallaan. Kun käyttäjä klikkaa kolmatta, edelliset käännetyt kortit kääntyvät nurinpäin ja näkyvillä ollut numero katoaa. Täten tehtävä vaatii pelaajalta tarkkaavaisuutta ja näkyvillä olevien numeroiden paikkojen mieleen painamista. Kun peli päättyy, voi *Tiedosto*-valikosta aloittaa uuden pelin haluamallaan vaikeustasolla tai lopettaa pelin.

6.6 Virheilmoitukset

Taulukossa 1 esitetään mahdollisesti esiintyviä virhetilanteita ja niiden ratkaisuja.

Taulukko 1. Virheilmoitukset

Virhetilanne	**Selitys**	**Ratkaisu**
Käyttöohjeistus ei avaudu.	Peli ei voinut avata käyttöohjetiedostoa tai ohjeet.html -tiedosto puuttuu tai se ei sijaitse pelihakemistossa.	Pura Muistipeli.jar-paketti uudelleen ja käynnistä peli uudelleen (kts. luku 6.2).
Pelikortin grafiikka ei näy.	Pelikortin kuvatiedosto Kortti.jpg puuttuu. Kortti.jpg ei sijaitse pelihakemistossa.	Pura Muistipeli.jar-paketti uudelleen ja käynnistä peli uudelleen, (kts. luku 6.2).
Peli näyttää vain pelkän peli-ikkunan eikä pelikortteja.	Java runtime enviroment on liian vanha tai versio väärä .	Asenna Sunin sivulta uusin Javan versio 1.6.
Peli ei näytä kortin numeroita.	Pelin resoluution tulee olla vähintään 1280 x 800.	Suurenna pelin resoluutiota näyttöasetuksista.

6.7 Ohjelman rajoitukset

Pelin pelaaminen pienikokoisella näytöllä saattaa muodostaa ongelmia kortin numeroiden näkymisen suhteen. Tämän vuoksi resoluutiosuositus on 1280 x 800 pikseliä. Peli toimii parhaiten 15 tuuman tai sitä suuremmalla näytöllä. Helpoimmalla tasolla on kuitenkin mahdollista pelata peliä jopa pienemmällä näytöllä. Peliä voidaan pelata vain yksinpelinä, ja se vaatii hiiren. Muita rajoituksia pelissä ei ole.

7 Yhteenveto

Opinnäytetyössä tehtävänä oli toteuttaa graafinen muistipeli Javalla. Työ oli omasta mielenkiinnosta valittu käytännönläheinen ohjelmistotuotantoprojekti, joka jakautui selkeisiin kokonaisuuksiin eli määrittely-, suunnittelu-, toteutus- ja testausvaiheisiin. Työssä käytettiin aikaa puolet ohjelman koodaukseen ja puolet raportin kirjoittamiseen. Työ alkoi tutustumalla NetBeansin ohjelmiston verkkosivulta löytyviin malliesimerkkikoodeihin [2]. Myös Java-kirjoissa ja netissä olevista koodiesimerkeistä oli tässä vaiheessa jonkin verran hyötyä. Muutamia pieniä koodinpätkiä kokeilemalla oppi ymmärtämään, miten graafinen ohjelmisto voidaan NetBeans-sovelluskehittimen avulla toteuttaa.

Aluksi oli vaikeaa ymmärtää, mitä komponentteja graafisen ohjelman tekemisessä tarvitaan. Myös luokissa tarvittavien komponenttien periyttäminen tuntui aluksi vaikealta. Määrittelydokumentin kirjoittaminen kuitenkin edesauttoi hahmottamaan, mitä ominaisuuksia muistipeli toimiakseen vaatii.

Pelin suunnittelussa lähdettiin liikkeelle miettimällä kaikki mahdolliset käyttötapaukset, joilla oli yhteisvaikutus käyttäjän ja pelin välillä. Myös tietokoneen omat käyttötapaukset mietittiin. Muistipelissä käyttötapauksia tässä ovat mm. pelin aloittaminen, pelaaminen, uusi peli, vaikeustason valinta, pistemäärien laskeminen, pelin parhaat tulokset ja pelin lopetus.

Käyttötapauksista piirrettiin kaavio ja tämän jälkeen toteutettiin ensimmäinen versio käyttöliittymän rakenteesta generoimalla se NetBeans-ohjelman avulla. Näin saatiin esiversio käyttöliittymästä täysin ilman koodausta. Valitettavasti generoitua koodia ei kuitenkaan osattu hyödyntää työssä, koska se vaikutti liian hankalalta ymmärtää. Tämän jälkeen suunniteltiin luokat ja piirrettiin luokkakaavio. Näiden yllämainittujen pohjalta sitten alkoi varsinainen koodausvaihe, joka alkoi luokkien pelikortti ja pelilauta metodien kirjoittamisella. Näitä molempia tarvittiin, jotta saatiin ensimmäinen graafinen ikkuna ohjelmasta näkymään. Ikkunassa näkyi aluksi pelkästään pelilauta, jossa kortit sijaitsivat. Myöhemmin luotiin kehysluokka, jossa määriteltiin menut, paneelit ja menujen

tapahtumien kuuntelijat. Lopuksi tehtiin parhaiden pelaajien lista pistemäärän tallennusta varten, mikä vaati kahden lisäluokan tekemisen. Toinen luokka on nimeltään PelitulosTietue, joka muodostaa pelaajista tietueet. Näitä tietueita hyödyntää Tulostaulu, joka tarkistaa, onko pelitulos riittävän hyvä päästäkseen parhaiden pelaajien listalle. Mikäli tulos oli riittävän hyvä, tallentaa tulostaulu pelaajan tiedot muuttamalla olemassa olevan tiedoston sisältöä yhden tietueen verran. Jokaisen vaikeustason tulostiedot sijaitsevat eri tiedostoissa, jotta pelitulokset olisivat vertailukelpoisia keskenään.

Työ opetti graafisen ohjelmoinnin perusperiaatteet ja siinä pääsi tutustumaan NetBeans-ohjelman työkaluihin ja myös kokeilemaan niitä. Työ toteutui suunnitelmien mukaisesti. Määrittelydokumentissa esitettiin työn prioriteetit, jonka mukaisesti ohjelmistoa tehtiin. Ensimmäisessä versiossa ei parhaita tuloksia vielä huomioitu. Ohjelman kehittelyä jatkettiin ja niin syntyi 2.0 -versio, joka täytti kaikki ohjelmalle asetetut vaatimukset. Yksi ainoa ominaisuus, joka lopullisessa versiossa jätettiin tekemättä, oli pistemäärien tallennus parhaiden tulosten listalle. Sen tilalla käytettiin aikalaskuria, koska huomattiin, että tuloksien tarkastelussa riittää vain aikojen vertailu. Peliaika tallennetaan taulukkoon vain, jos se on riittävän hyvä ja mahtuu 10 parhaan pelaajan listalle. Tulostaulu järjestellään pelissä käytetyn ajan mukaan, eli vähiten peliaikaa läpäisemiseen käyttänyt pääsee listan kärkeen.

Tämä opinnäytetyö ei luonut mitään mullistavaa, tosin siitä saattaisi olla hyötyä jollekin opinnäytetyötä tekevälle. Työ kuvaa ohjelmistotyön prosessien vaiheet kattavasti. Muita muistipeliä netistä käsitteleviä töitä käytettiin vertailupohjana, kun mietittiin käyttöliittymässä tarvittavia toimintoja ja ominaisuuksia. Koodi kuitenkin luotiin itse. Työ auttoi ymmärtämään, mitä graafisten ohjelmien tekeminen käytännössä vaatii ja minkälainen prosessi se todellisuudessa on. Työn tekeminen oli erittäin opettavaista ja vaati pitkää pinnaa. Välillä ohjelmointityössä saattoi tulla sellaisia ajanjaksoja, jolloin ei saatu aikaiseksi yhtään mitään. Jotta ohjelmointityö onnistuisi, tulisi antaa riittävästi aikaa ajatusten kypsymiselle.

Ohjelmaa voidaan tulevaisuudessa kehittää lisäämällä siihen vielä uusia ominaisuuksia. Seuraavaan versioon voitaisiin kehittää kaksinpeliominaisuus tietokone vs. ihminen. Se

tulisi toteuttaa tekoälyn avulla siten, että kone etsisi automaattisesti pareja ja tallentaisi niiden sijaintitietoja taulukkoon. Vuorottelua varten pitäisi keksiä jokin sääntö, jonka perusteella pelivuoro vaihtuisi. Toinen vaihtoehto pelin kehittämiselle olisi toteuttaa kaksinpeli, jossa pelaajat vuorotellen kääntelevät kortteja pelilaudalla. Tästä voitaisiin myöhemmin toteuttaa myös verkon tai Internetin välityksellä pelattava peliversio. Myös grafiikkaa voisi tulevissa versioissa vielä parantaa, jotta pelistä saataisiin entistä houkuttelevampi ja miellyttävämpi pelikokemus. Myös kortin kääntämisefektin luominen voisi tuoda peliin uuden piristävän vaikutelman.

On huomattavaa myös, että graafisten elementtien valinnassa on usein ohjelmointiprojekteissa monta eri mahdollisuutta. Sopivimman komponentin valinta monista vaihtoehdoista ei ole helppoa. Näitä vaihtoehtoja ei tässä työssä vielä osattu oikein punnita, koska kokemuksia vastaavista projekteista ei ollut. Esimerkiksi kumpi komponenteista *Jbutton* vai *Jlabel* kortin toteutukselle olisi ollut parempi vaihtoehto? Valittiin *Jbutton*, mutta myöhemmin ohjelmointityön loppuvaiheessa osoittautui, että saman olisi voinut toteuttaa huomattavasti paremmin *Jlabelilla*, koska se olisi mahdollistanut graafiset kuvat myös kortin toiselle puolelle. Samalla numeroiden näkymisen ongelma pienemmällä resoluutiolla olisi saattanut poistua.

Hyvänä puolena pelin koodissa oli joustavien taulukkotietorakenteiden käyttäminen korttien määrille ja pelitulostietueille. Tietorakenteiden osalta tähän päädyttiin, koska haluttiin, että taulukon koko voisi kasvaa ohjelman tarpeiden mukaisesti. Tämä lyhensi myös tarvittavaa koodin määrää ja antaa entistä enemmän mahdollisuuksia kehittää ohjelmaa tulevaisuudessa eteenpäin.

Lähteet

1 NetBeans-sovelluskehittimen kotisivu. (WWW-dokumentti.) Sun Microsystems, Inc. <www.netbeans.org/index_fi.html>. Luettu 25.10.2008.

2 Creating a GUI with JFC/Swing. The Swing tutorial. (WWW-dokumentti.) Sun Microsystems, Inc. <www.java.sun.com/docs/books/tutorial/uiswing/>. Luettu 25.10.2008.

3 Haikala Ilkka, Märijärvi Jukka, Ohjelmistotuotanto. 11. painos. Helsinki: Talentum Media Oy. 2006.

4 Nenonen, Jaakko, Ohjelmoinnin harjoitustyö. (WWW-dokumentti.) HY / TKTL 2006. <www.cs.helsinki.fi/u/jnenonen/alabra/maardoc.html>. Luettu 5.11.2008.

5 Nenonen, Jaakko, Ohjelmoinnin harjoitustyö. (WWW-dokumentti.) HY / TKTL 2005. <www.cs.helsinki.fi/u/jnenonen/alabra/suunnittelu.html>. Luettu 5.11.2008.

6 Nenonen, Jaakko, Ohjelmoinnin harjoitustyö. (WWW-dokumentti.) HY / TKTL 2005. < www.cs.helsinki.fi/u/jnenonen/alabra/toteutusdoc.html>. Luettu 11.11.2008

7 Java Look And Feel Design Guidelines. (WWW-dokumentti.) Second Edition. 2001. Sun Microsystems, Inc. < www.java.sun.com/products/jlf/ed2/book/>. Luettu 6.2.2009.

8 Kosonen, Peltomäki & Silander, Java 2 ohjelmoinnin peruskirja. 1. painos. Jyväskylä: Docendo Finland Oy. 2005.

9 Ryynänen, Pekka, Säikeiden toiminta ja käyttö. (WWW-dokumentti.) <www.niksula.cs.hut.fi/~pjryynan/stu1portfolio/esseet/essee5/essee5threads.html>. Luettu 8.11.2008.

10 Wikla, Arto, Ohjelmoinnin perusteet Java-kielellä. 4.painos. Hämeenlinna: Karisto Oy. 2003.

Liite1: Luokkakaavio