TAMPEREEN AMMATTIKORKEAKOULU
Tietotekniikan koulutusohjelma
Ohjelmistotekniikka

Tutkintotyö

Marko Pellikka

VAHVUUSLUKUHALLINTA

Työn valvoja: Lehtori Erkki Hietalahti
Työn ohjaaja: Samuli Saarenpää, Suomen Renju ry
Tampere 2007

Tekijä: Marko Pellikka
Työn nimi: Vahvuuslukuhallinta

Päivämäärä: 4.3.2007
Sivumäärä: 36 sivua + 30 liitesivua
Hakusanat: vahvuuslukulaskenta, vahvuusluku, renju, ELO
Koulutusohjelma: Tietotekniikka
Suuntautumisvaihtoehto: Ohjelmistotekniikka

Työn valvoja: Lehtori Erkki Hietalahti

Työn ohjaaja: Samuli Saarenpää

Suomen Renju ry on suhteellisen nuori yhdistys. Se perustettiin vuonna 2003. Yhdistyksen jäsenluku on hiljalleen kasvanut, ja kasvun odotetaan lisääntyvän huomattavasti lähivuosien aikana. Yksi tärkeistä jäsenyystietoja koskevista osa-alueista on vahvuuslukuhallinta. Ei ole olemassa vahvuuslukuhallintajärjestelmää, joka olisi lajista tai pelistä riippumaton standardi. Kukin lajiyhdistys on tehnyt omat valintansa käyttämänsä järjestelmän suhteen. Usein on käynyt niin, että valmiit järjestelmät eivät ole täysin soveltuneet heidän tarpeisiinsa, mutta silti on päädytty käyttämään niitä, koska tällöin niihin ei ole tarvinnut perehtyä syvällisesti.

Tämä työ käsittelee erään vahvuuslukuhallintamallin valintaa, teoriataustaa ja sen soveltamista. Kyseinen malli, kuten monet muutkin olemassa olevat mallit, ei määrittele täydellisesti vahvuuslaskennan vaiheita, vaan antaa siihen kehykset, joihin lopullisen laskennan voi soveltaa tapauksittain. Työssä onkin keskitytty näihin ratkaisuihin ja työn ohessa syntyneen ohjelman perustoiminnallisuuteen ja tiedon varastoimiseen.

Author: Marko Pellikka
Title: Vahvuuslukuhallinta

Date: 4.3.2007
Pages: 36 pages + 30 appendix pages
Keywords: rating calculation, ratings, renju, ELO
Program: Computer Systems Engineering
Specialization: Software engineering

Supervisor: Lehtori Erkki Hietalahti

Instructor: Samuli Saarenpää

The Finnish Renju federation is a rather young, as it was founded in 2003. So far, the number of members has been growing slowly although it is supposed to grow rapidly in coming years. One of the main fields regarding to the membership information is the sustained rating calculation. There is no such existing implementation of rating calculation that has been chosen as a standard among all the different gaming or sport federations. Each federation has made their own decisions in applying their own rating system based on some rating calculation method. Some of them have ended up using badly implemented solutions without better insight of the operation of the system.

This thesis is concentrating on choosing suitable rating system model for Finnish Renju Federation including its theory and adaptation for association's needs. The chosen model, like many other existing models, doesn't define itself explicitly, but instead describes some basic fundamentals for rating calculation. This thesis is mostly about choosing the best rating system model, applying it by making a software based on it and keeping up a database of players' ratings.

ALKUSANAT

Tämä työ on tehty Tampereen ammattikorkeakoulun ohjelmistotekniikkalinjan tutkintotyönä.

Kiitokset mielenkiintoisesta aiheesta ja yhteistyöstä Suomen Renju ry:lle. Haluan myös kiittää työn valvojaa, lehtori Erkki Hietalahtea.

4.3.2007, Tampere

_____

Marko Pellikka

SISÄLLYSLUETTELO

KÄYTETYT TERMIT JA LYHENTEET

ASCII	(American Standard Code for Information Interchange) tietokoneiden merkistö, joka sisältää englannin kielen kirjaimet, numerot, joukon välimerkkejä ja joitakin ohjauskoodeja.
BYE	Merkintä, joka osoittaa, että tietty pelaaja on saanut vapaavuoron turnauksessa.
C++	Ohjelmointikieli
CR	(Carriage Return) Rivinvaihtoa osoittava ohjauskoodi
Crosstable	Turnaustuloslista
DAN	Itämainen vahvuusarvo, joka perustuu DAN/KYU -asteikkoon.
DRAW	Tasapeli. Pelitulos, jossa kumpikaan osapuoli ei voittanut eikä hävinnyt.
ELO	Vahvuuslukulaskentamalli
enum	(enumeration) C-kielen komento, jolla voi määritellä tietyn sanallisen termin käsittämään tiettyä arvoa, joka eroaa muista samassa yhteydessä ilmoitetuista arvoista. Arvojen tietäminen on ohjelmoijalle epäoleellista.
Gaussin käyrä	Normaalijakauma
Glicko System	Vahvuuslukulaskentamalli
Gomoku	Yksinkertaisempi versio renjusta
Heksaeditori	Ohjelma, jolla voi tarkastella ja muokata tietokoneella olevan muistin tai tiedoston sisältöä heksadesimaalimuodossa.
HONORARY	Jäsenyystyyppi, joka merkitsee Suomen Renju ry:n kunniajäsentä.
ID	(Identification) Yksilöllinen tunniste
IDE	(Integrated Development Environment) Integroitu ohjelmointiympäristö
K, K-arvo	Painoarvovakio, joka toimii tärkeänä kertoimena vahvuuslukulaskennassa.
KYU	Itämainen vahvuusarvo, joka perustuu DAN/KYU -asteikkoon

LF	(Line Feed) Rivinvaihtoa osoittava ohjauskoodi
MEMBER	Jäsenyystyyppi, joka merkitsee Suomen Renju ry:n jäsentä.
MM	Maailmanmestaruus
MySQL	SQL-tietokannan hallintajärjestelmä
N/A	(Not Available) Ei saatavilla
OTHER	Jäsenyystyyppi, joka merkitsee, että kyseinen pelaaja ei ole RIF:n eikä Suomen Renju ry:n jäsen.
RIF	(1) (Renju International Federation) Kansainvälinen Renjun kattojärjestö (2) Jäsenyystyyppi, joka merkitsee RIF:n jäsentä.
RR, Round robin	Turnaustyyppi, jossa kaikki pelaavat kaikkia vastaan.
ry	Rekisteröity yhdistys
SQL	(Structured Query Language) IBM:n kehittämä standardoitu relaatiotietokannan kyselykieli.
STL	(Standard Template Library) C++-ohjelmointikielessä käytetty käskykirjasto, joka sisältää mm komennot dynaamisten linkkilistojen käyttämiseksi.
Swiss	Turnaustyyppi, jossa pelaajaparien muodostus on suhteellisen monimutkainen operaatio.
USCF	(US Chess Federation) Yhdysvaltojen shakkiyhdistys
wxWidgets	Laitteistoriippumaton, avoimen lähdekoodin lisäkirjasto, joka on tarkoitettu graafisten käyttöliittymien ohjelmointiin.

1 JOHDANTO

Suomen Renju ry on ottamassa vahvuuslukujärjestelmää käyttöön. Tämä tutkintotyö koskee tietokoneohjelman luomista, joka mahdollistaa jäsenien vahvuuslukujen laskennan ja raportoinnin. Tarkoituksena on saada jokaiselle Suomen Renju ry:n jäsenelle oma henkilökohtainen vahvuusluku, jonka pääasiallinen tehtävä on kertoa kunkin pelaajan taitotaso. Vahvuuslukujärjestelmien käyttö on yleistä muissa lajeissa. Voidaankin perustellusti sanoa, että järjestelmän luominen vahvistaa toiminnan pohjaa ja lisää mielenkiintoa.

Vahvuuslukujärjestelmästä on hyötyä sekä pelaajille että yleisölle. Pelaajat voivat seurata omaa kehitystään ja etsiä tasoisiaan vastustajia. Jotkut jopa saavat ylimääräisen sysäyksen peliinsä tietäessään pelaavansa parempaansa vastaan. Vahvuuslukujärjestelmän myötä vertailu ulkomaalaisiin pelaajiin helpottuu. Tiettyjen marginaalien sisällä on mahdollista verrata omaa tasoaan esimerkiksi japanilaisten pelaajien tasoon.

Järjestelmän luomiseen on useita eri vaihtoehtoja. Jokin vaihtoehto voi olla liian yksinkertainen ollakseen toimiva, kun taas toinen vaihtoehto voi olla liian monimutkainen, jotta yksittäinen pelaaja voisi halutessaan ymmärtää laskentatavan toimintaperiaatteet. Laskentamallin valitseminen ja valitun mallin soveltaminen ovat osa tätä tutkintotyötä. Olisi hyvä saada aikaiseksi sellainen toimiva malli, joka olisi yhdenmukainen ja vertailukelpoinen muiden maiden vahvuuslukujärjestelmien kanssa. Valitettavasti joissakin maissa on asian osalta menty siitä, missä aita on matalin, eikä vertailukelpoisuus näin ollen ole kovin hyvä.

Yksi tärkeimmistä vertailukohteista on RIF (Renju International Federation), jolla on kansainvälinen vahvuuslista pelaajista, jotka ovat osallistuneet RIF:n hyväksymiin turnauksiin. Heidän vahvuuslukujärjestelmänsä on suhteellisen toimiva. Suomen Renju ry ei ole kuitenkaan halukas ottamaan täysin vastaavaa järjestelmää omaan käyttöönsä, sillä kyseistä järjestelmää sovellettaessa yksi-yhteen menisi vuosia ennen kuin pelaajien vahvuudet menisivät oikeille

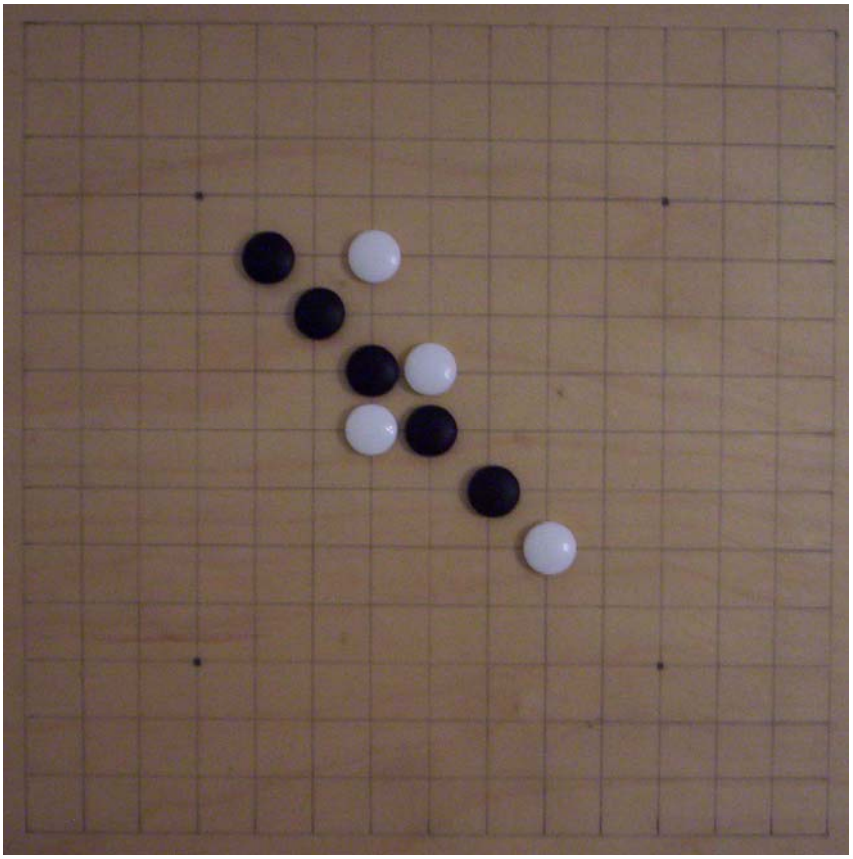
tasoilleen. Ottaessaan vahvuuslukujärjestelmän käyttöön 1996 RIF käyttikin hyväksi 88 eri turnauksen tuloksia vuosilta 1988 – 1996. /1/

Suomen Renju ry:llä ei ole omassa listassaan kuin runsaat kymmenen turnausta, joita voisi hyödyntää vahvuulukujärjestelmän pohjan luomisessa. Näin ollen on löydettävä keinoja, joilla vahvuuslukukehitys nopeutuu ja saavutetaan pelaajien kohdalla mahdollisimman pian sellaiset vahvuusluvut, jotka vastaavat heidän pelitaitoaan mahdollisimman hyvin. Vahvuuslukujärjestelmä on tarkoitus ottaa käyttöön lähitulevaisuudessa. Jäämme odottamaan, millaisen vastaanoton se saa renju-yhteisössä. Moni renjun harrastaja varmasti haluaa suomalaisen vahvuusluvun.

1.1 Renju

Renju on jätkänshakin kaltainen lautapeli, jossa on tarkoitus muodostaa viiden suora omista pelinappuloista. Toinen pelaaja pelaa mustilla ja toinen valkeilla nappuloilla. Renjun historia on tuhansia vuosia vanha. Sen kehittäjästä ja syntymisajankohdasta ei ole tietoa, mutta todennäköisesti peli on syntynyt Kiinassa. Tuolloin peli oli nimeltään gomoku, joka on säännöiltään yksinkertaisempi versio renjusta. Gomokussa aloittajalla on varma voitto, joten renjun sääntöjä on jouduttu kehittämään sellaisiksi, että ne olisivat mahdollisimman tasapuoliset molemmille pelaajille. Pelin perusidea on kuitenkin sama, viiden suoran muodostaminen. /2/

Pelin alkaessa on erityinen avausvaihe, jossa pelin aloittaja asettaa laudalle kaksi mustaa ja yhden valkean pelinappulan. Tämän jälkeen toinen pelaaja saa valita, kummalla värillä haluaa pelata. Valkoisilla pelinappuloilla pelaava pelaaja asettaa nyt laudalle pelinappulan. Mustilla pelaava pelaaja laittaa seuraavaksi kaksi pelinappulaa, joista valkeilla pelaava pelaaja saa ottaa toisen pois. Tämän jälkeen avausvaihe on jo ohi, ja peli jatkuu vuorosiirroin. Mustilla pelinappuloilla pelaavalle pelaajalle on lisäksi tiettyjä erityisrajoituksia, joilla rajoitetaan hänen peliään. Syynä tähän on se pieni etu, että mustia pelinappuloita on aina yksi enemmän pelilaudalla.



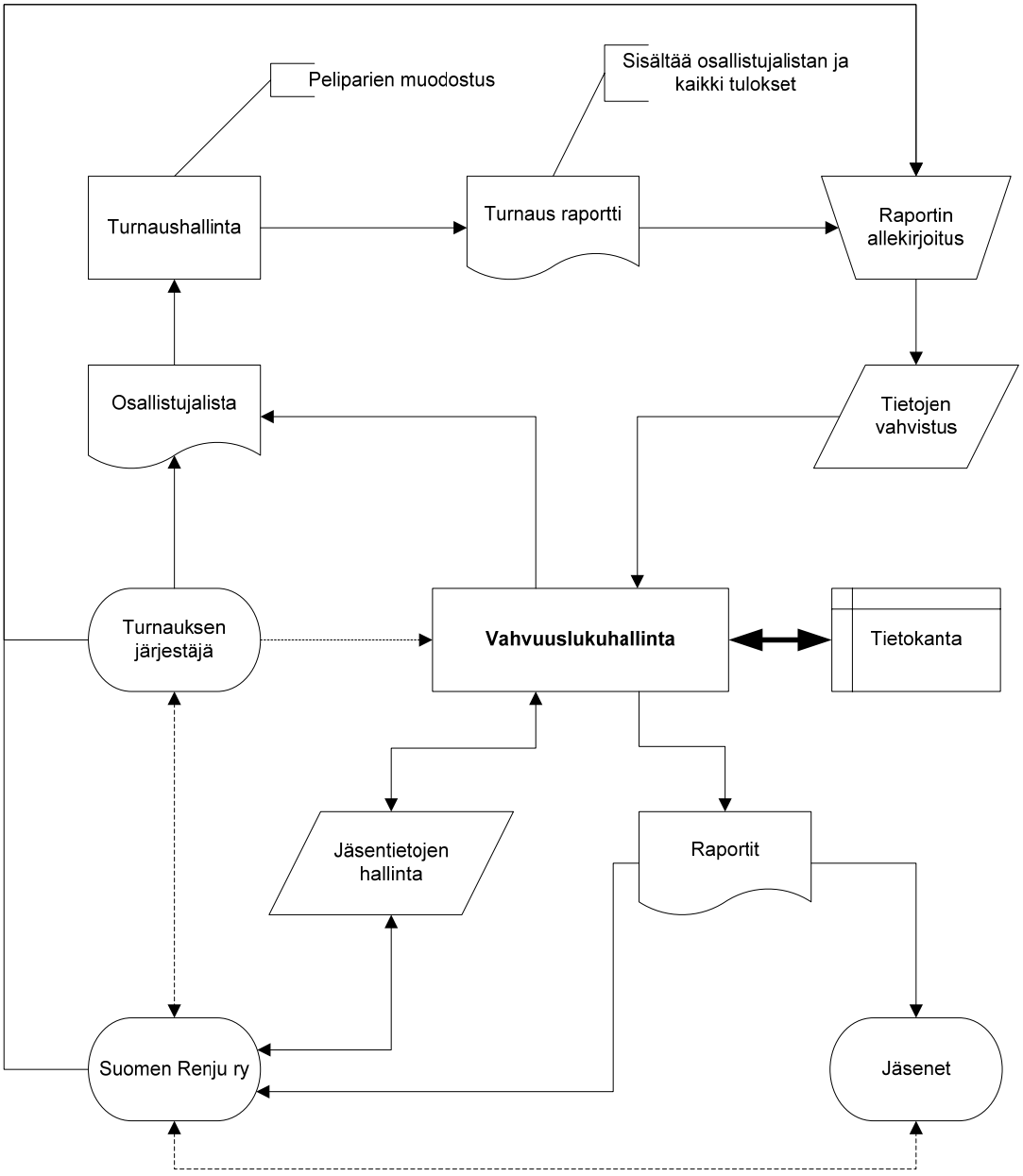
Kuva 1. Renjulauta

Peliä pelataan laudalla, jonka koko on 14 x 14 ruutua. Pelissä pelinappuloita sijoitetaan paikkoihin, joissa ruudukon viivat risteävät. Sen vuoksi pelattavan alueen todellinen koko on 15 x 15 risteystä. Kuvassa 1 on pelilauta, 5 mustaa ja 4 valkoista pelinappulaa. Mustilla pelinappuloilla pelannut on voittanut pelin, koska hän on muodostanut viiden suoran pelilaudalle. Suora kulkee kuvassa 1 viistoon, mutta voittaakseen voi muodostaa myös suoran vaaka- tai pystysuoraan.

Suomessa on aidon renjun pelaajia vain parikymmentä. Se on paljon vähemmän kuin naapurimaissamme Ruotsissa, Venäjällä tai Virossa. Suomen Renju ry aloitti toimintansa vuonna 2003, ja siitä lähtien renju-yhteisö on hitaasti kasvanut meidänkin maaperällämme. Useimmat uudet renju-pelaajat aloittavat pelaamisen ja harjoittelun Internetissä, ja vähitellen osallistuvat oikeisiin turnauksiin.

1.2 Pelaajatietojen käsittely

Vahvuuslukuhallinta käsittelee tietoja pelaajista ja laskee heille vahvuusluvut. Tiedot annetaan syötteinä ja vahvuuslukuhallinta raportoi joko ruudulle tai tiedostoon. Tämän lisäksi vahvuuslukuhallinta ylläpitää omaa tietokantaa sille syötetyistä turnauksista ja jäsentiedoista.



Kuva 2. Käyttötarkoituskaavio

Kuvassa 2 esitetään kaavio siitä, miten vahvuuslukuhallinta sijaitsee suhteessa eri tason käyttäjiin ja turnaushallintaan. Nuolet osoittavat mihin suuntaan tapahtumat, vaiheet tai toiminnot siirtyvät normaalitoiminnassa. Esimerkiksi vahvuuslukuhallinnan kautta voi antaa osallistujalistan turnaushallinnalle, jolta saadaan turnaustulokset sisältävä turnausraportti. Mikäli tulokset vahvistetaan pitäviksi, voidaan vahvuuslukuhallinnassa laskea turnauspelaajien vahvuus-lukumuutokset ja tämän jälkeen raportoida ne.

1.3 Aiheen rajaus

Tämä tutkintotyö keskittyy vahvuuslukuhallintaan ja sen käyttämään tietokantaan. Työssä ei juuri perehdytä ohjelman toimintaan rivi riviltä, vaan paneudutaan teoriaan ja valintoihin, jotka tähän työhön liittyvät.

2 VAHVUUSLUKULASKENTA-TEKNIIKAT

Tässä luvussa kerrotaan työssä sovellettavasta vahvuuslukualgoritmista ja sen taustoista. Laskenta perustuu yleensä pelituloksiin. Jokaisessa pelissä on yksi häviäjä ja yksi voittaja. Tasapelin voi rinnastaa siihen, että kumpikin puoliksi voitti ja puoliksi hävisi.

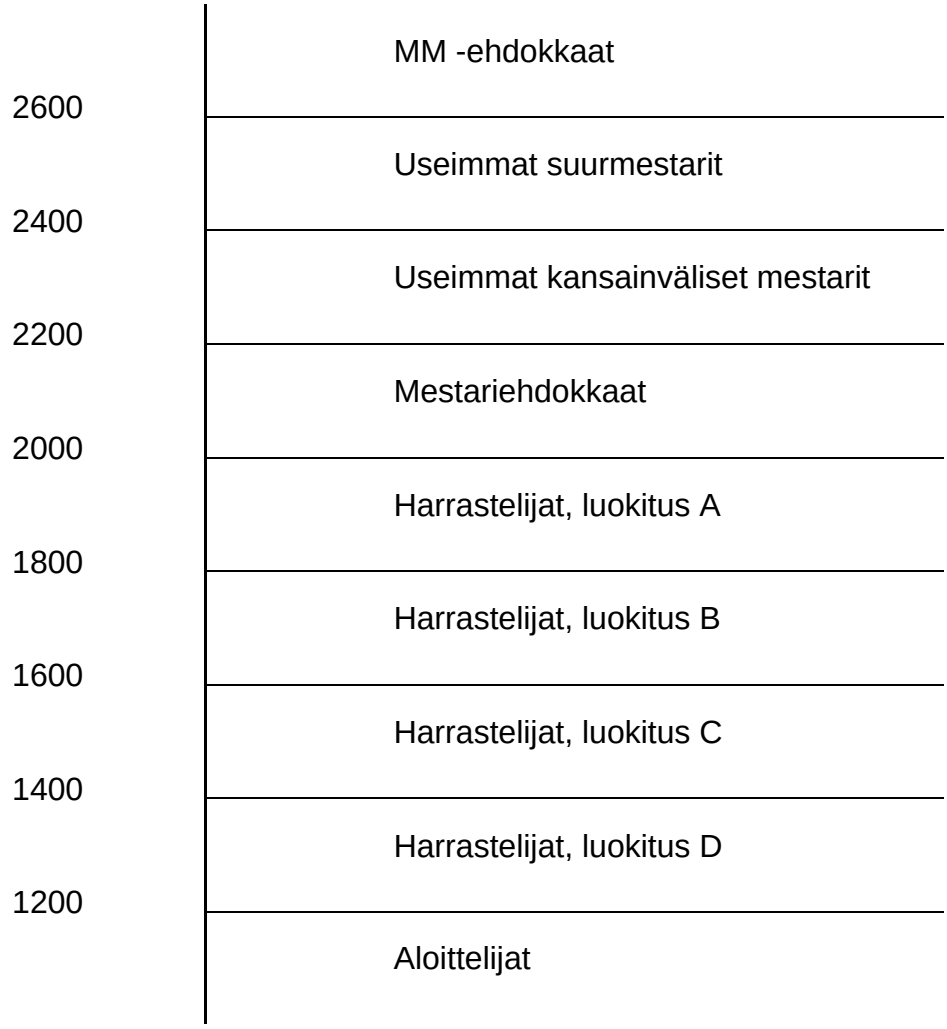
2.1 ELO-järjestelmä

ELO-järjestelmä on saanut nimensä luojansa matemaatikko Árpád Élõ:n mukaan. Vuonna 1960 USCF, joka on Yhdysvaltojen shakkiyhdistys, otti käyttöön ELO-järjestelmän. Samaa järjestelmää käytetään nykyään niin shakissa kuin monessa muussakin lajissa. Se on suhteellisen toimiva ja yksinkertainen järjestelmä. Parempiakin järjestelmiä on kehitetty, esimerkiksi Glicko System, jonka on luonut Mark Glickman. Tämän järjestelmän ongelma on lähinnä sen monimutkaisuus. Sitä on vaikea ymmärtää ja soveltaa. Tässä työssä sovelletaan mukailtua ELO-järjestelmää. Mukaillut kohdat koskevat *rating pool* (2.1.2.1) ja *provisional* -vaiheita (2.1.2.2). *Established*-vaihe (2.1.2.3), joka on myös

useimmin käytetty vaihe, noudattaa Árpád Élõn kehittämiä kaavoja ilman poikkeuksia. /3/, /4/

2.1.1 ELO-järjestelmän asteikko

ELO-järjestelmän asteikko perustuu lukuarvoihin. Nämä lukuarvot voidaan mieltää pisteiksi, joissa suuri pistelukema tarkoittaa hyvää pelitaitoa. Pistetaso voi niin nousta kuin laskeakin. Siihen vaikuttavat pelitulokset muita pelaajia vastaan. Tyypillisesti pelitaito lisääntyy kokemuksen myötä ja pelaaja nousee asteikossa ylöspäin.



Kuva 3. Vahvuusasteikko /4/

Kuvassa 3 esitetään luokittelu tavalla, jolla Árpád Élõ sen aikoinaan määritteli. Nykyään vahvuusasteikon skaala on hieman noussut. Aloittelijat saavat nopeasti vahvuudekseen 1300–1600 vahvuuspistettä ja MM-ehdokkaat ovat tyypillisesti 2700–2850 vahvuuspisteen välillä. Vahvuusarvojen jakauma noudattaa pitkälti Gaussin käyrää, jossa mediaanina on tyypillisesti vahvuuslukema väliltä 1600–2000 vahvuuspistettä. Yksi tyypillisimmistä poikkeuksista, joka hieman muotoilee tätä käyrää, koskee uusien pelaajien tulemista heikompana ja lähtevän vahvempana pelaajana, mikä on aivan luonnollista kehitystä pelaajan kohdalla. Poikkeuttaviin tekijöihin voidaan myös lukea se, että laskentaan mukaan otettavien pelaajien kokonaispelimäärä ei ole vakio. /5/

Lukuarvoihin perustuva asteikko on selkeästi suosituin länsimaissa. Idässä enemmän käytetty asteikko perustuu DAN/KYU -arvoihin. Tätä järjestelmää sovelletaan myös tulevaisuudessa Suomen Renju ry:ssä ELO-järjestelmän lisäksi.

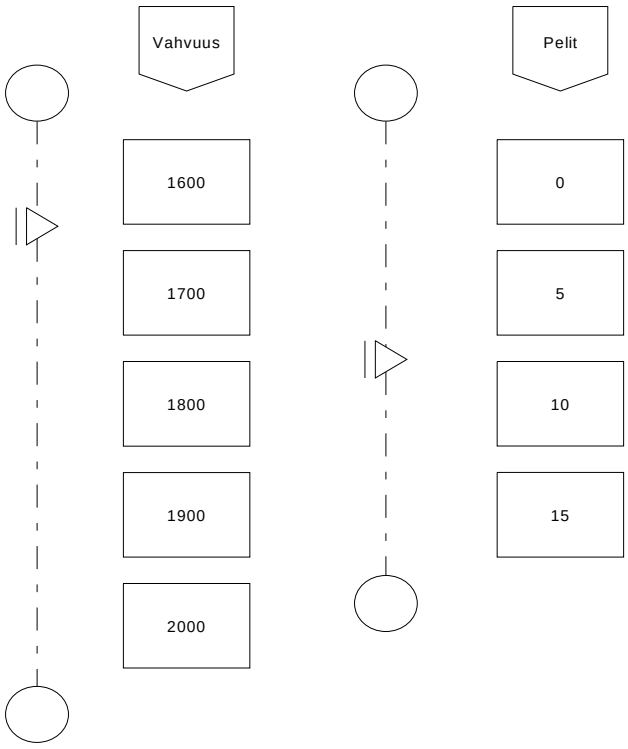
2.1.2 Vahvuuslukujen laskeminen

ELO-järjestelmän laskenta jaetaan kahteen eri vaiheeseen. Nämä kaksi perusvaihetta ovat nimityksiltään *provisional* ja *established*. Otettaessa järjestelmää käyttöön tulee myös esivaihe, *rating pool* (2.1.2.1), jonka muodostamiseen järjestelmä ei ota kantaa.

2.1.2.1 Rating pool -vaiheen laskenta

Rating pool -vaiheessa luodaan vahvuuskantaa pelaajien keskuuteen. Ei ole juurikaan mitään vertailukohtaa mitä käyttää, mutta kuitenkin pitäisi alkaa vahvuuslukujen kehittäminen. Tyypillistä tässä vaiheessa olevalle pelaajalle on se, että hän on todennäköisesti keskivertoa parempi pelaaja, joten hallinnollisesta näkökulmasta katsoen hänelle on oltava mahdollisuus asettaa tavallista korkeampi alkuvahvuus. Tämän ongelman ratkaisemiseksi on mahdollistettu seuraavanlaiset toimenpiteet.

Suomen Renju ry:llä on erityinen vahvuuslukutyöryhmä, jonka vastuualueisiin kuuluuvat pelaajien taitotasojen arviointi ja vahvuuslukujärjestelmän ylläpitäminen. Uudelle pelaajalle voidaan määritellä työryhmän aloitteesta alkuvahvuus, joka perustuu työryhmän arviointiin. Alkuvahvuus ei ole pelkästään vahvuuspistelukema, vaan se voi myös sisältää pelimäärän kasvatuksen, jolla autetaan pelaajaa pääsemään entistä nopeammin *established*-laskennan piiriin. (2.1.2.3)



Kuva 4. Alkuvahvuusarvot

Vahvuuspisteiksi voidaan asettaa 1600, 1700, 1800, 1900 tai 2000. Näistä vaihtoehdoista 1600 on oletusasetus. Pelaajalle voidaan myös antaa asetuksena pelattujen pelien määrä: 0, 5, 10 tai 15. Alkuvahvuusarvojen vaikutusta selvennetään seuraavin esimerkein:

- **Esimerkki 1:** Vahvuus 1600, Pelit 0

  Tämä on oletusasetus uuden pelaajan alkuvahvuusarvoiksi. Pelaajan on pelattava täydet 20 peliä päästäkseen *provisional*-vaiheesta *established*-

vaiheen laskentaan. Vahvuus on arvioitu keskitasolle eli 1600 vahvuuspisteeseen.

- **Esimerkki 2:** Vahvuus 1600, pelit 15

  Näillä alkuvahvuusasetuksilla pelaaja on arvioitu taitotasoltaan keskitasolle mahdollistamatta vahvuuden nopeaa muutosta. Tämä on turvallinen valinta, mikäli halutaan välttää tilastollisia poikkeamia *provisional*-vaiheessa. Tällainen asetus edellyttää, että rating-työryhmä on melko varma, että kyseisen henkilön taidot vastaavat vahvuuspistelukemaa 1600.

- **Esimerkki 3:** Vahvuus 2000, pelit 0

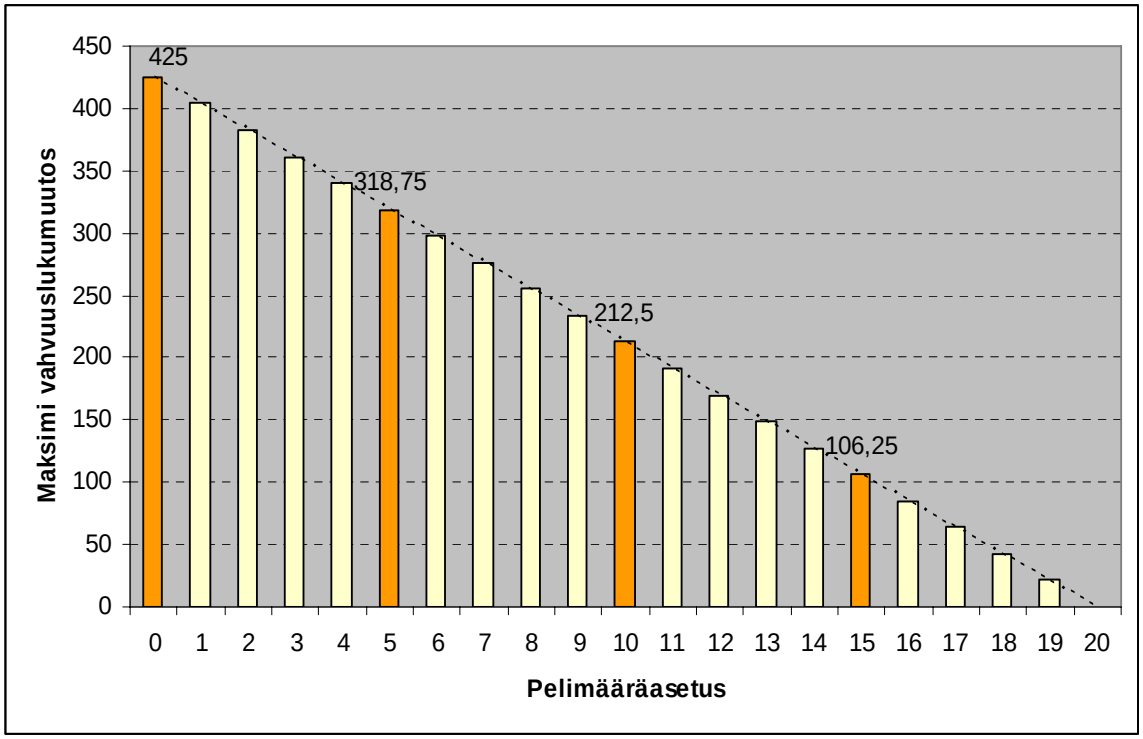
  Nämä ovat tyypilliset alkuvahvuusasetukset useimmille Suomen Renju ry:n nykyisistä pelaajista. Alkuvahvuudeksi annetaan suurin sallittu arvo ja pelimääräksi pienin mahdollinen lukumäärä, jotta vahvuuspistelukema voisi hakeutua joustavasti suuntaan tai toiseen *provisional*-vaiheen aikana. Suuri vahvuuspistelukema selittyy sillä, että pelaajat ovat jo hankkineet kokemusta ja taitoja ennen vahvuuslukujärjestelmän käyttöönottoa.

- **Esimerkki 4:** Vahvuus 2000, pelit 15

  Tässä esimerkissä pelaajan todellisen taitotason on arvioitu olevan aika tarkalleen 2000 vahvuuspistettä, joten pelaaja halutaan päästää pian *established*-laskennan piiriin.

Pelimäärän asetus vaikuttaa myös *provisional*-vaiheen aikaiseen maksimimuutokseen. Mitä suurempi esiasetettu pelimäärä, sitä pienempi muutos on mahdollista saavuttaa. Mikäli esiasetettu pelimäärä poikkeaa oletuksesta, tällöin esiasetettu vahvuusluku taivuttaa muutoksen lähtöpistettä kohti

esiasetetun vahvuusluvun lukuarvoa, joka muutoin olisi tarkalleen *provisional*-vaiheen aikana pelattujen vastusten keskivahvuuspistelukema.

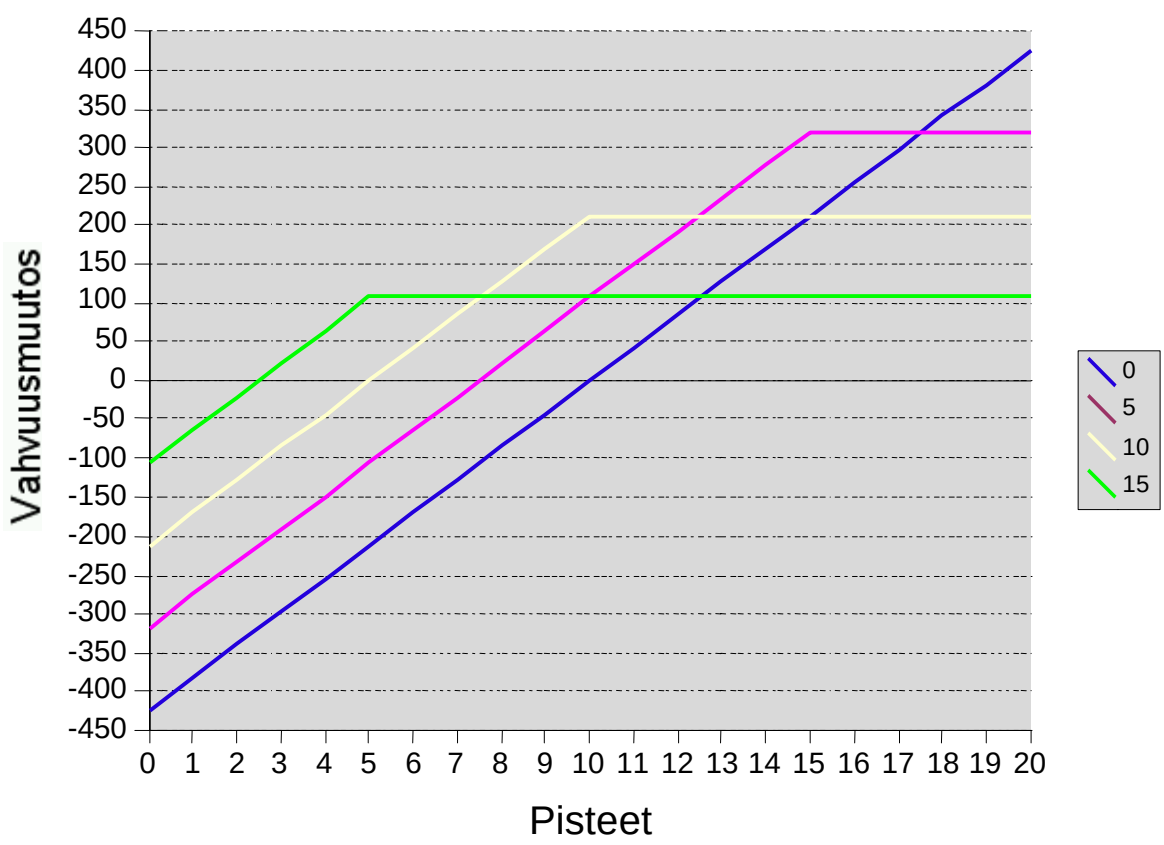


Kuva 5. Vahvuuslukumuutoksien maksimitapaukset

Kuva 5 esittää, miten suuri vahvuuslukumuutos enimmillään on mahdollinen asetetun pelimäärän mukaisesti. Mahdolliset pelimääräasetukset ovat oransseissa palkeissa. Tapaus nolla peliä (ensimmäinen ja pisin oranssi palkki) on edelleen oletus, joka koskee yhtä lailla normaalia *provisional*-vaiheen muutosta. Kyseisessä tapauksessa suurin mahdollinen vahvuuslukumuutos on 425 vahvuuspistettä suuntaan tai toiseen. Mikäli pelimääräasetus on 15, tällöin suurin mahdollinen vahvuuslukumuutos on 106 vahvuuspistettä.

Muutoksen lähtöpiste vastaa *provisional*-vaiheen aikana pelattujen vastusten keskivahvuuspistelukemaa. Mikäli pistesaldo peleistä on puolet, tällöin lopullinen muutos on nolla eli vastusten keskivahvuuspistelukema. Jos esiasetettu pelimäärä poikkeaa oletuksesta, eli nollasta, niin tällöin muutoksen lähtöpiste taipuu kohti esiasetettua vahvuuslukemaa. Maksimimuutoksen

tapahtuminen käytännössä on melko harvinaista. Se edellyttää, että kyseinen pelaaja joko voittaa tai häviää kaikki *provisional*-vaiheen pelinsä.



Kuva 6. Pistemäärän vaikutus vahvuusmuutokseen

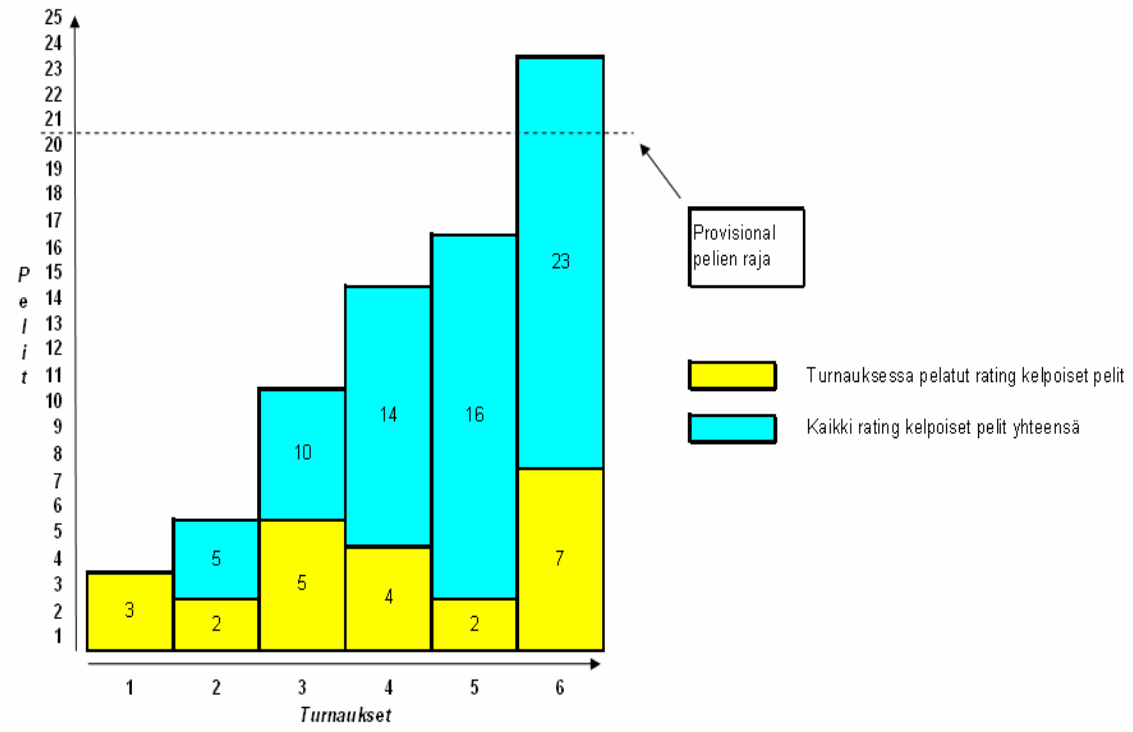
Kuvan 6 tummansinisessä käyrässä esitetään pistemäärän vaikutus vahvuusmuutokseen, kun pelimääräasetus on 0 peliä. Punaisessa käyrässä pelimääräasetus on 5, vaaleankeltaisessa 10 ja vihreässä 15 peliä. Jos tiedetään *rating pool* -vaiheen mukainen pelimääräasetus ja saavutettu pistemäärä *provisional*-vaiheen peleistä, voidaan pelaajan vahvuusmuutos katsoa kuvasta 6.

Suomen Renju ry on päättänyt hyväksyä *rating pool* -käyttöön kaikki renjuturnaukset, joissa on ollut suomalaisia mukana vuodesta 2003 alkaen. *Rating Pool* -vaiheen käyttö on tarkoitus lopettaa kokonaan vuosien 2007-2009 aikana. Tähän mennessä vaiheeseen hyväksytyt turnaukset on esitetty liitteessä 3.

2.1.2.2 Provisional-vaiheen laskenta

Provisional-vaiheen tarkoitus on toimia tietyn tyyppisenä esivaiheena vahvuuslukukehitykselle, jonka aikana järjestelmään lisätty uusi pelaaja voi saavuttaa tavallista nopeammin tietyn vahvuustason. Tämä voidaan käsittää iteratiivisena vaiheena ennen *established*-vaihetta. Voidaan ajatella, että pelaajalla on tietty vahvuus, joka koetetaan kartoittaa mahdollisimman tarkkaan hänen ensimmäisten peliensä aikana. Tämän vuoksi *provisional*-vaiheen laskenta on huomattavasti progressiivisempaa kuin *established*-vaiheen laskenta, eli saavutetut pelitulokset vaikuttavat enemmän vahvuusluvun muuttumiseen. Kuitenkaan vahvuutta ei julkaista vielä tämän vaiheen aikana, joten sisäinen toiminta on kyseiselle pelaajalle näkymätöntä.

Provisional-vaihe käsittää uuden pelaajan 20 ensimmäistä vahvuuslukukelpoista peliä. Tämän vaiheen laskenta eroaa normaalista *established*-vaiheen laskennasta siinä, että tämän vaiheen aikana ei vielä lasketa vahvuuslukua turnaus turnaukselta, vaan vahvuusluku julkaistaan juuri tämän vaiheen päättyessä.



Kuva 7. Vahvuuslukukelpoisten pelien kertyminen

Kuvassa 7 turnaukset, joihin oletettu pelaaja on osallistunut, esitetään vaaka-akselilla. Kyseisen pelaajan kokonaispelimäärä on puolestaan esitetty pystyakselilla. Peleiksi lasketaan vain vahvuuslukukelpoisiksi luettavat pelit, joiden valintaan vaikuttaa turnauksissa kohdattujen vastapelaajien vahvuuslukuluokitukset. Vastapelaajan tulee olla joko Suomen Renju ry:n tai RIF:n jäsen, jotta peli voidaan hyväksyä *provisional*-vaiheen laskentaan. Vaaka-asteikon alimmat palkit kertovat kussakin turnauksessa pelatuista vahvuuslukukelpoisista peleistä ja tämän päällä oleva palkki esittää näiden pelien kokonaiskertymää. Kuvan 7 esimerkissä oleva pelaaja on saavuttanut ensimmäisessä turnauksessaan kolme vahvuuslukukelpoista peliä. Toisessa turnauksessa lukema on kaksi. Kokonaiskertymä on viisi peliä, mikä saadaan kun lasketaan yhteen ensimmäisen ja toisen turnauksen vahvuuslukukelpoiset pelit.

Kokonaiskertymän ylittäessä 20 peliä (kuva 7, turnaus 6), on vaaditut 20 vahvuuslukukelpoista peliä suoritettuna, jolloin *provisional*-vaiheen vahvuusluku saadaan laskettua seuraavalla kaavalla:

$$R_p = Rc + [(W_C - 0.50) * 850]$$

Kaava 1. *provisional*-vaiheen vahvuuslukukaava /6/, /4/

Kaavassa 1 on käytetty seuraavia symboleita:

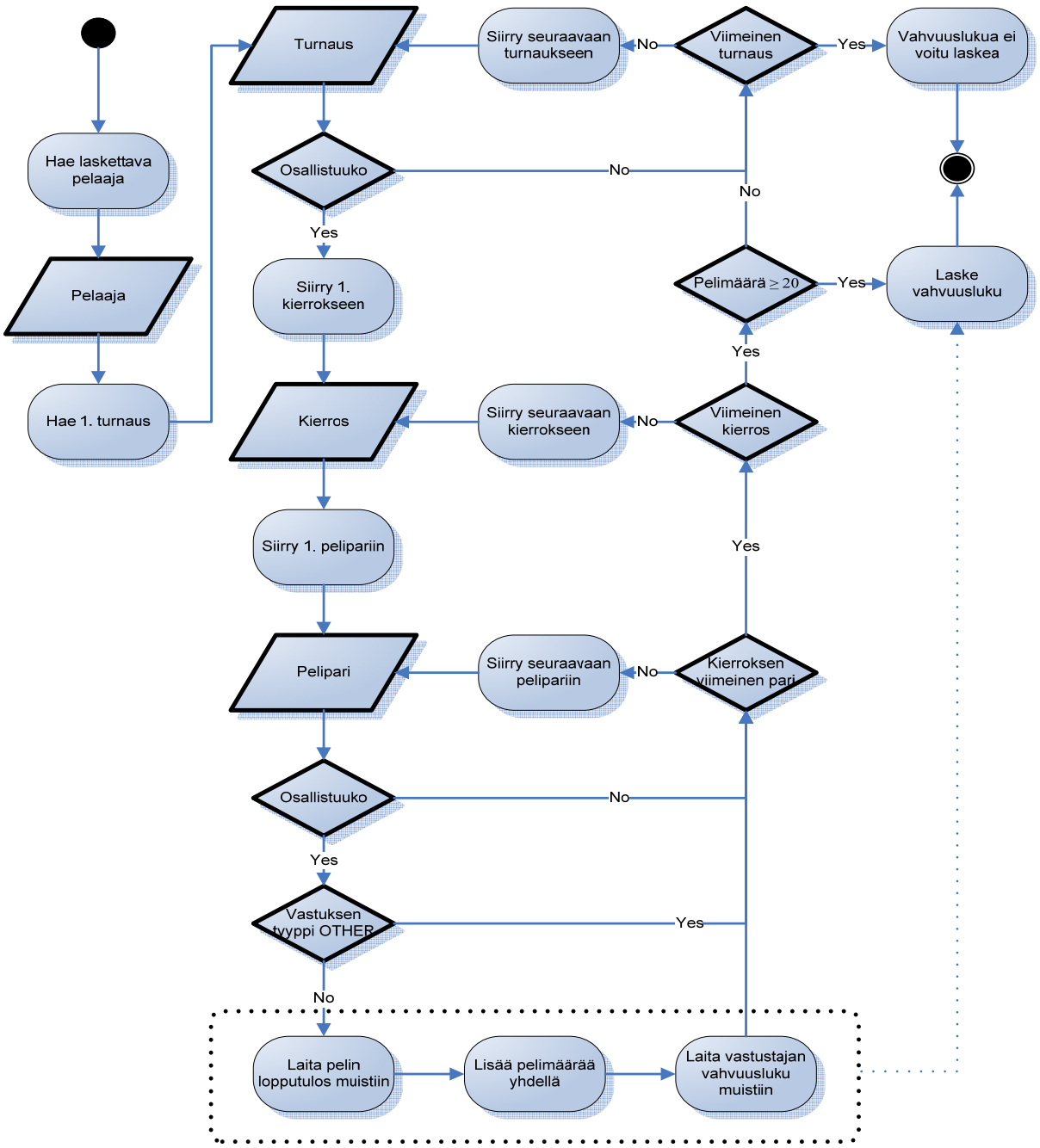
R_p (Performance Rating) = Vahvuus, joka toimii alkuna *established* laskennalle.

R_c (Average competition) = Vaiheen aikana pelattujen vastustajien keskimääräinen vahvuus.

W_C (PctScore) = Keskimääräinen suoritus *Provisional*-vaiheen aikana. Esimerkiksi, jos voitti kaikki pelit, W_C on silloin 1.

Provisional-vaiheen päättävän vahvuuslukulaskennan suorittamiseksi tulee pelata tarpeeksi suuri määrä turnauksia ja tuloksia eri pelivastuksia vastaan läpi,

jotta vaadittu 20 vahvuuslukukelpoisen pelin raja ylittyisi. Tällöin vahvuusluku voidaan myös laskea.



Kuva 8. Provisional -vaiheen vahvuuslukulaskenta

Vain pelit RIF-jäseniä ja Suomen Renju ry:n jäseniä vastaan huomioidaan. Myös pelit sellaisia vastustajia vastaan, joiden vahvuusluku on tyypiltään *provisional*, otetaan huomioon. Tällöin vahvuus on kappaleen 2.1.2.1 alkuasetusten mukainen. Muussa tapauksessa oletus eli 1600 vahvuuspistettä.

Tämä laskentavaihe mahdollistaa vahvuusmuutoksen 850/2 = 425 vahvuuspistettä (kuva 5) suuntaan tai toiseen. Mikäli *rating pool* -pelaajan alkuasetuksissa on pelimäärää nostettu, huomioidaan se keskivahvuuslukulaskennassa siten, että hän olisi pelannut esiasetetun pelimäärän mukaisesti pelejä vastustajia vastaan, joiden vahvuuspisteluku on sama kuin pelaajan esiasetettu vahvuus. Tämä luonnollisesti pienentää mahdollisuutta vaikuttaa vahvuuden muutokseen esiasetukseen nähden, mutta toisaalta tällainen pelaaja pääsee aikaisemmin *established*-vaiheen mukaisen laskennan piiriin.

2.1.2.3 Established-vaiheen laskenta

Established-vaiheessa vahvuuslukema muuttuu jokaisen turnauksen jälkeen. Pelaajan uusi vahvuuslukema on laskettavissa kaavalla 2, jos tiedetään edellinen vahvuuslukema, painoarvovakio, pelin lopputulos ja lopputuloksen odotusarvo.

$$R_n = R_O + K(W - W_e)$$

Kaava 2. Established-vaiheen vahvuuslukukaava

Kaavassa 2 on käytetty seuraavia symboleita:

R_n = Uusi vahvuuslukema
R_O = Edellinen vahvuuslukema
K = Painoarvovakio
W = Pelin lopputulos kyseisen pelaajan näkökulmasta, jossa voitosta saa yhden pisteen, tasapelistä ½ ja tappiosta 0 pistettä.
W_e = Lopputuloksen odotusarvo

Lopputuloksen odotusarvo W_e perustuu pelaajan ja hänen vastustajansa vahvuuslukemiin ennen peliä. Odotusarvo kertoo todennäköisyyden, jolla pelaajan A tulisi voittaa pelaaja B. Odotusarvot pelaajille A ja B voidaan laskea kaavoilla:

$$E_A = \frac{1}{1+10^{(R_B-R_A)/400}}$$

$$E_B = \frac{1}{1+10^{(R_A-R_B)/400}}$$

Kaava 3. Odotusarvot /4/

Kaavassa 3 on käytetty seuraavia symboleita:

E_A = Pelaajan A odotusarvo (=todennäköisyys, jolla hän voittaa pelaajan B)

E_B = Pelaajan B odotusarvo (=todennäköisyys, jolla hän voittaa pelaajan A)

R_A = Pelaajan A vahvuusluku

R_B = Pelaajan B vahvuusluku

Odotusarvo käsitellään laskennassa käyttäen kahden desimaalin tarkkuutta. Välttääksemme pyöristysvirheet ja yksinkertaistaaksemme ohjelmaa, kaavan 3 mukaisesta yhtälöstä on muodostettu taulukko ohjelmaan.

D	P		D	P		D	P	
Rtg. Dif.	H	L	Rtg. Dif.	H	L	Rtg. Dif.	H	L
0-3	0,50	0,50	122-129	0,67	0,33	279-290	0,84	0,16
4-10	0,51	0,49	130-137	0,68	0,32	291-302	0,85	0,15
11-17	0,52	0,48	138-145	0,69	0,31	303-315	0,86	0,14
18-25	0,53	0,47	146-153	0,70	0,30	316-328	0,87	0,13
26-32	0,54	0,46	154-162	0,71	0,29	329-344	0,88	0,12
33-39	0,55	0,45	163-170	0,72	0,28	345-357	0,89	0,11
40-46	0,56	0,44	171-179	0,73	0,27	358-374	0,90	0,10
47-53	0,57	0,43	180-188	0,74	0,26	375-391	0,91	0,09
54-61	0,58	0,42	189-197	0,75	0,25	392-411	0,92	0,08
62-68	0,59	0,41	198-206	0,76	0,24	412-432	0,93	0,07
69-76	0,60	0,40	207-215	0,77	0,23	433-456	0,94	0,06
77-83	0,61	0,39	216-225	0,78	0,22	457-484	0,95	0,05
84-91	0,62	0,38	226-235	0,79	0,21	485-517	0,96	0,04
92-98	0,63	0,37	236-245	0,80	0,20	518-559	0,97	0,03
99-106	0,64	0,36	246-256	0,81	0,19	560-619	0,98	0,02
107-113	0,65	0,35	257-267	0,82	0,18	620-735	0,99	0,01
114-121	0,66	0,34	268-278	0,83	0,17	>735	1,00	0,00

Kuva 9. Vahvuuserotustaulukko /4/

Kuvan 9 taulukossa käytetyt lyhenteet:

D	=	Vaihteluväli
Rtg. Dif.	=	Vahvuuserotuksen itseisarvo ($R_A - R_B$)
P	=	Todennäköisyys
H	=	Peliparista suuremman vahvuuslukeman omaavan pelaajan voittotodennäköisyys.
L	=	Peliparista pienemmän vahvuuslukeman omaavan pelaajan voittotodennäköisyys.

Kuvan 9 mukainen taulukko auttaa hahmottamaan nopeasti kahden eri pelaajan väliset voittotodennäköisyydet. Ensimmäisellä rivillä esitetään tapaus, jossa pelaajien vahvuuksien välinen erotus on 0-3 vahvuuspistettä. Tässä tapauksessa katsotaan, että kummallakin pelaajalla on yhtä suuri todennäköisyys voittaa peli eli P=0,50=50%. Taulukko jatkuu aina niin pitkälle, että ero on yli 735 vahvuuspistettä. Tämä erotus on jo niin suuri, että katsotaan toisella pelaajalla olevan 100%:n voittotodennäköisyys. Käytännössä ei voi koskaan sanoa, että pelaaja voittaa toisen pelaajan 100%:n todennäköisyydellä, vaan että todennäköisyys lähestyy 100%:a. Tämä voidaan kuitenkin pyöristää käytetyllä kahden desimaalin tarkkuudella 100%:iin. Vaikka kyseisen todennäköisyyden omaava pelaaja häviäisi pelin, ei vahvuuslukeman muutos ole kuitenkaan kovin suuri. Muutos vahvuuslukuun on enintään K × 1,00, missä K voi saada enimmillään arvon 32, mistä seuraa 32 × 1 = 32 vahvuuspisteen menettäminen hävitessä vastustajalle, jota vastaan olisi teoriassa ollut täysi 100%:n voittotodennäköisyys. Edellinen esimerkki havainnollistaa, että *established*-vaiheen vahvuuslukemamuutokset ovat suhteellisen pieniä.

2.1.2.3.1 Painoarvovakio K

Established-laskennan yhteydessä käytetään vakiota K. Se toimii kertoimena vahvuuslukumuutokselle. Tämä kerroin voi saada arvoja väliltä 0–32. Tämä kerroin on määritelty alkuperäisessä ELO-järjestelmässä, mutta sen soveltaminen on jätetty vahvuuslukujärjestelmiä käyttävien tahojen harteille. Tyypillisesti se vaihtelee kuitenkin välillä 0–40. Suomen Renju ry käyttää

perusarvona lukua 32. Tämä ei voi kasvaa, mutta voi mahdollisesti laskea tapauksen mukaan.

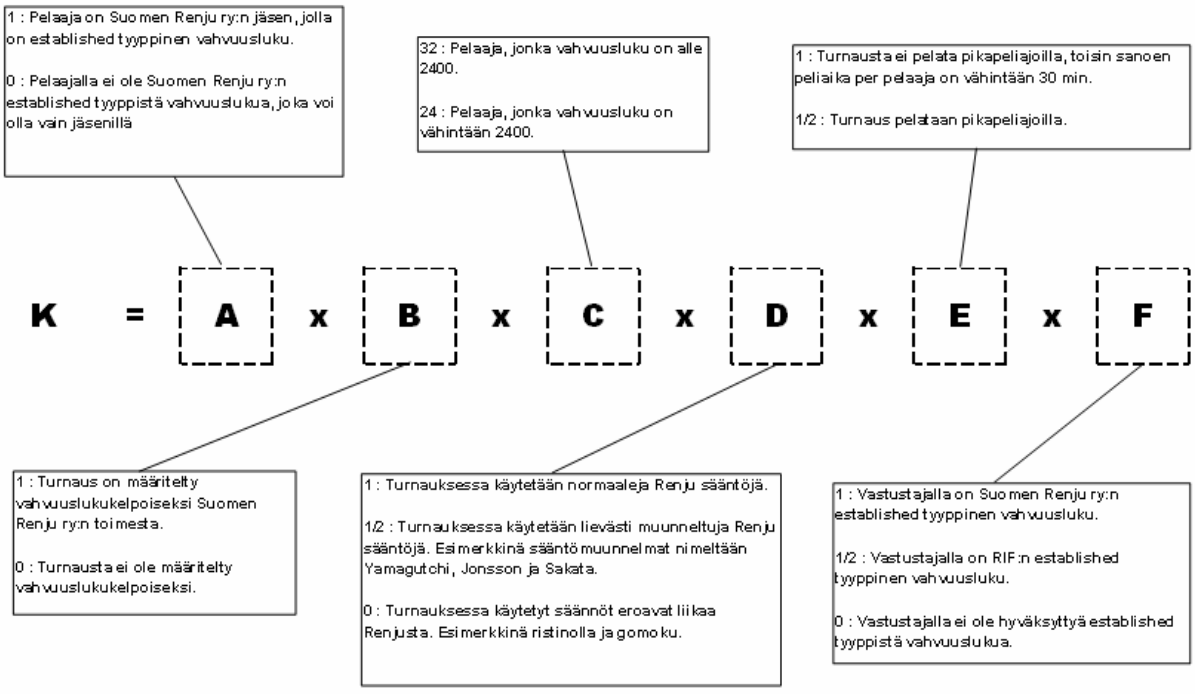
Established-laskennan aikana vahvuusmuutoksiin vaikuttaa myös osallistutun turnauksen tyyppi. Turnauksen tyyppi määritellään ja hyväksytään Suomen Renju ry:n toimesta. Pääsääntöisesti kukin seuraavista tapauksista laskee K-arvoa laskettavan pelaajan kohdalla aina puolella:

- Turnauksissa, joissa käytetään lievästi muunneltuja pelisääntöjä.

- Turnaukset, joissa peliaika on vähemmän kuin 30 minuuttia/pelaaja. Tällaiset turnaukset käsitetään yleensä peliterminologisesti pikapeleiksi.

- Vastustajat, joilla ei ole Suomen Renju ry:n *Established*-tyyppistä vahvuuslukua, mutta joilla on RIF-vahvuusluku.

Lisäksi pelaaja, jonka vahvuusluku on vähintään 2400, vahvuusmuunnosta hillitään pienentämällä K-arvo lukuun 24 ennen edellä mainittujen kohtien huomioimista.



Kuva 10. K-arvon laskenta

Kuvassa 10 esitetään kokonaisvaltainen kaava K-arvon laskemiseksi. Kertoimet A, B, C, D, E ja F vaikuttavat K-arvon muodostumiseen. Kertoimille esitetyistä arvovaihtoehdoista aina ylimmäinen on oletus, eli se mitä yleisimmin käytetään, mikäli ollaan päädytty tämän laskennan piiriin.

Laskennan jälkeen K voi saada arvokseen jonkun seuraavista: 0, 3, 4, 6, 8, 12, 16, 24, 32. Kertoimissa A, B, D ja F on vaihtoehtona myös arvo nolla. Kussakin näissä kohdista on eritelty tapaus, jonka ansiosta K-arvo voi päätyä nollaksi. Mikäli K-arvo on 0, ei vahvuuslukumuutosta tapahdu laskennan alaiselle pelaajalle. Kertoimet A ja B ovat mukana varmuuden vuoksi. Toteutuksellisesti laskentaa ei suoriteta, mikäli jompikumpi näistä voi saada arvoksi nollan, mutta joissain tapauksissa sillä voi havainnollistaa hypoteettisesti vahvuusmuutosta vastustajaa vastaan, jota vastaan pelattu peli ei ole Suomen Renju ry:n määritelmien mukaisesti vahvuuslukukelpoinen.

Esimerkki K-arvon laskennasta: Pelaaja A, jonka vahvuusluku on 2475, pelaa Suomen Renju ry:n jäsentä B vastaan, jonka vahvuusluku on 1959 ja tyyppiä *established*. Vastustajalla on myös RIF-vahvuusluku 1876. Turnaus on pikapeliturnaus, jossa käytetään lievästi muunneltuja pelisääntöjä.

- Pelaajan A vahvuusluku on yli 2400, joten K arvo ennen muita ehtoja on 24. Vastustajan vahvuusluku ei vaikuta millään tavalla K-arvon muodostumiseen.
- Koska käytössä on lievästi muunnellut pelisäännöt, joten K-arvo puolitetaan. Uusi K-arvo on 12.
- Pikapeliturnaus laskee K-arvon 6:een.
- Vastustaja B on Suomen Renju ry:n jäsen, jolla on *established*-tyyppinen vahvuusluku. K-arvo pysyy 6:ssa. Se, että B omaa myös RIF vahvuusluvun, vaikuttaisi vain siinä tapauksessa, että B:llä ei olisi Suomen Renju ry:n *established*-tyyppistä vahvuuslukua.

Esimerkin tapauksessa K sai arvokseen 6. Se on melko pieni arvo ja suhteellisen harvinainen.

2.2 Jäsenyystyypit

Vahvuuslukuhallintaan syötetyt pelaajat voivat olla niin jäseniä kuin muita pelaajia. Lisäämällä vahvuuslukulaskentaan kaikkien turnauksien pelaajat ja tulokset varmistetaan tuloslistojen eheys. Tiedoista on myös hyötyä, jos jossain vaiheessa päätetään nostaa pelaajan luokitusta. Vahvuuslukulaskennassa on pelaajien jäsenyystyypit jaettu seuraavasti.

- MEMBER : Suomen Renju ry:n jäsen, joka on oikeutettu saamaan lasketun vahvuuslukuarvon. Jos tämä pelaaja on myös RIF:n jäsen, hänen jäsenyystyyppinsä pysyy MEMBER-luokituksessa, eikä RIF:n jäsenyys käy ilmi vahvuuslukuhallinnassa.
- HONORARY : Suomen Renju ry:n erityisjäsen, jolla on laskennallisesti samat oikeudet kuin tavallisellakin jäsenellä.
- RIF : Kansainvälisen renjujärjestön jäsen. Tälle pelaajalle ei lasketa vahvuuslukua, mutta Suomen Renju ry:n jäsenien vahvuuslukuun vaikuttavat pelit RIF:n jäseniä vastaan.
- OTHER : Pelaaja, joka ei kuulu mihinkään edellämainituista tai pelaaja, jolla on vasta RIF:n *provisional*-tyyppinen vahvuusluku. Pelit OTHER-tyyppistä pelaajaa vastaan eivät vaikuta kummankaan vahvuuskehitykseen mitenkään.

3 TIETOKANTA

Työssä käytettävä tietokanta on itse suunniteltu binääritiedosto. Sen tiedostonimi on *data.bin*. Siihen tallennetaan kaikki pelaaja- ja turnaus-informaatio. MySQL-tyyppinen tietokanta olisi hyvä ja luotettava, mutta sitä ei voida käyttää, koska kaikilla tietokoneilla sitä ei ole asennettuna tai sen käyttöön on rajoituksia. Vahvuuslukuhallintaohjelman tulee toimia Windows 32-bit -järjestelmissä ilman erillistä asennusta, joten MySQL-kantaa ei voida luoda. Ohjelmaan on kuitenkin suunnitteilla ominaisuus, joka mahdollistaa MySQL-kannan luonnin käyttäen MySQL-luontikäskyjä. Tämä on hyödyllinen

toiminnallisuus esimerkiksi siinä tapauksessa, jos halutaan pitää Internetissä kantaa pelaajien tiedoista, joita voi tarkastella Web-käyttöliittymän kautta.

Binärityyppinen tiedosto on tiedostokooltaan pienempi kuin tekstitiedosto. Lisäksi se on helpompi käsitellä, kun tietoa tallennetaan suoraan tietueista ja luetaan tietueihin. Binääritiedostoihin ei tule myöskään kääntäjästä tai käyttöjärjestelmästä riippuvaisia rivinvaihtomerkkejä, joita ovat Windowsin CR- ja LF-merkit, Linuxin LF-merkki tai Mac OS X:n CR-merkki.

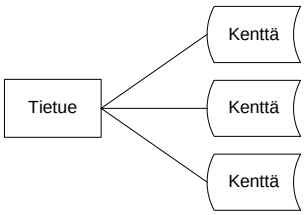
Testauksen kannalta binääritiedosto on huonompi kuin tekstitiedosto. Ohjelmoijan tulee tuntea tietokannan rakenne tarkoin, ja sisällön tarkastelu tapahtuu yleensä joko heksaeditorilla tai erillisillä testausfunktioilla.

4 TIETORAKENTEET

Tässä luvussa käsitellään ohjelman toiminnan kannalta oleellisia tietorakenteita. Ohjelma hyödyntää STL-kirjaston linkitettyjä listoja, joihin sijoitetaan sekä pelaajiin että turnauksiin liittyviä tietoja.

4.1 Tietueet

Tietuetyyppi on rakenteeltaan koosteinen ja sisältää tietokenttiä, jotka edustavat perustietotyyppiä tai johdettua tyyppiä. Näitä käytetään kokoamaan yhteen sellainen informaatio, jotka ovat yhteenkuuluvia käsiteltävästä kohteesta. /9/



Kuva 11. Tietue

Tietueessa on kenttiä (kuva 11), jotka sisältävät muuttujia, joilla on tyyppi ja tunnus.

Vahvuuslukuhallintaohjelmassa käytetään kahta eri tietuetta. Nämä tietueet ovat määritelty ohjelmakoodissa *struct*-määreen avulla. Käytettyjen tietueiden nimet ovat *henkilo* ja *turnaus*. Tietue *henkilo* sisältää pelaajaan liittyvät tiedot ja tietue *turnaus* peliturnaukseen liittyvät tiedot. Viittaukset pelaajaan tai henkilöön liittyvät aina *henkilo*-tietueeseen.

4.1.1 Henkilo-tietue

Kaikki turnauksiin osallistuvat henkilöt ovat pelaajia, joiden tiedot tallennetaan tietueeseen *henkilo*.

```
typedef struct
{
    unsigned int id;
    char nimi[32];
    unsigned int games;
    unsigned int rating;
    unsigned int rstat;
    unsigned int orig[2];
} henkilo;
```

Jokaisella järjestelmään lisätyllä henkilöllä on oltava oma ID, se tallennetaan tietueen kenttään id. Sen lisäys on automaattinen, joten vahvuuslukuhallinnan käyttäjän ei tarvitse huolehtia siitä, että järjestelmään tulisi päällekkäisyyksiä. ID-numerointi alkaa luvusta 101 ja jatkuu siitä yhden välein. Tähän numerointiin on vain yksi poikkeus, sillä järjestelmään on lisätty yhteensopivuuden parantamiseksi henkilö nimeltään BYE ja hänen ID:kseen on annettu luku 99. BYE on täysin kuvitteellinen ja sitä käytetään ainoastaan osoittamaan tilannetta, kun pelaaja on jäänyt jollakin turnauskierroksella ilman peliparia. Uuden pelaajan lisäyksen yhteydessä ohjelma myös tarkistaa, löytyykö toista samannimistä pelaajaa järjestelmästä. Pelaajan nimi on muotoa ”<Sukunimi> <Etunimi>” tallennettuna tietueen kenttään nimi[].

Henkilo-tietueen kenttä games kertoo turnauksissa pelattujen vahvuuslukukelpoisten pelien lukumäärän. Kenttä rating kertoo kyseisen pelaajan viimeisimmän vahvuusluvun, ja kenttä rstat kertoo pelaajan jäsenyystyypin. Tämä sisältää *enum*-määrittelyn mukaisesti vaihtoehdot

OTHER, RIF, MEMBER ja HONORARY (kappale 2.2). Nämä kuvaavat jäsenyystyyppiä. Huomionarvoista on, että vaikka pelaaja olisi samanaikaisesti RIF:n ja Suomen Renju ry:n jäsen, järjestelmälle oleellista on tietää pelaajan olevan tällöin vain Suomen Renju ry:n jäsen. Tietueen kenttä orig[] sisältää *rating pool* -vaiheen pelaajan alkuperäiset vahvuus ja pelimäärä tiedot.

4.1.2 Turnaus-tietue

Turnaus on tapahtuma, jossa joukko pelaajia kohtaavat toisensa ja pelaavat pelejä toisiaan vastaan. Oleellisia tietoja ovat ainakin turnauksen nimi ja osallistujamäärä. Pelitulokset, eli kuka voitti ja ketkä pelasivat vastakkain, ovat *turnaus*-tietueen viimeisessä kentässä nimeltä lista. Nimensä mukaisesti se on linkitetty lista ja merkityksensä vuoksi tunnetaan paremmin nimellä tuloslista, josta enemmän kappaleessa 4.2.3.

```
typedef struct
{
    unsigned int id;
    char nimi[32];
    unsigned int rating_status;
    unsigned int pvm;
    unsigned int participants;
    unsigned int rounds;
    unsigned int system;
    std::list<unsigned int> lista;
} turnaus;
```

Kuten jokaisella *henkilo*-tietueella, tulee myös jokaisella *turnaus*-tietueella olla oma ID:nsä, tietueen kenttä id. Tämänkin lisäys on automaattinen. Turnauksella on nimi tietueen kentässä nimi[]. Turnauksen päiväys on tallennettuna kenttään pvm. Osallistujamäärä on kentässä participants ja kierrosmäärä kentässä rounds. Turnaus-tietueen kenttä rating_status kertoo, mikä on vahvuuslukuluokittelu kyseiselle turnaukselle. Esimerkiksi pikapelit ja sovelletuin pelisäännöin pelatut turnaukset saavat eri vahvuuslukuluokituksen. Tämä vaikuttaa *established*-laskennan K-arvon laskemiseen (2.1.2.3.1).

Turnaus-tietueen kenttä system sisältää kuvauksen käytetystä peliparien muodostamismenetelmästä. Tämä tieto helpottaa manuaalista turnaustulosten syöttämistä ja syötettyjen tietojen eheyden tarkistusta.

4.2 Linkitetyt listat

Vahvuuslukuhallintaan tulee useiden pelaajien ja turnausten tiedot. Ohjelman ajon aikana ne tallennetaan linkitettyihin listoihin. Nämä listat ovat STL-kirjaston mukaisia, joten dynaaminen muistinhallinta hoituu automaattisesti ja operointi käyttäen asiaan kuuluvia listametodeja. Listojen käyttö helpottaa tiedon käsittelyä, etenkin järjestäminen onnistuu helposti valitun tietueen kentän mukaisesti. Tyypillisesti vahvuuslukuhallintaan liittyviä tietoja laitetaan järjestykseen tietueissa esiintyvän ID:n, päiväyksen tai nimen mukaan.

4.2.1 Henkilölista

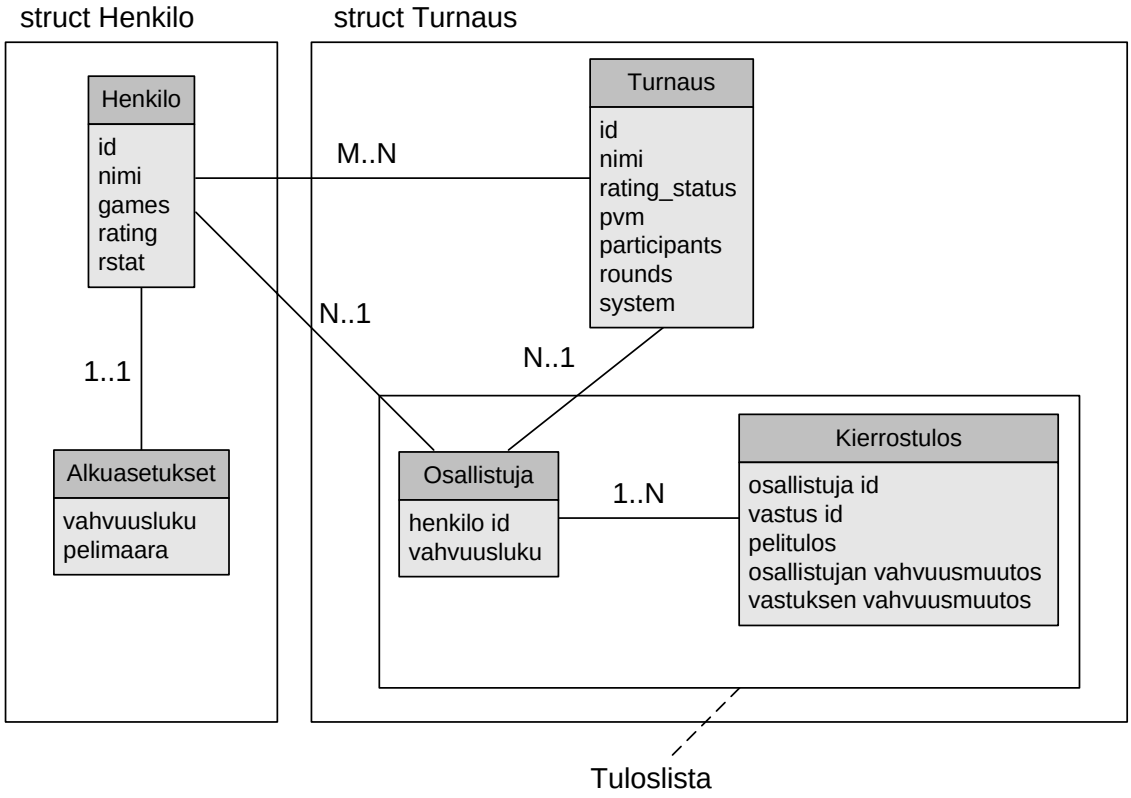
Listan esittely: std::list<henkilo> listh

Henkilölista pitää sisällään *henkilo*-tietueet (4.1.1). Lista on yleensä järjestettynä tietueiden ID:eiden mukaisesti.

4.2.2 Turnauslista

Listan esittely: std::list<turnaus> listt

Turnauslistaan on tallennettu kaikki *turnaus*-tietueet. Kukin *turnaus*-tietue pitää sisällään tuloslistan.



Kuva 12. Tietuekenttien sidokset linkitetyissä listoissa

Määriteltyjen tietueiden sidokset toisiinsa voi mieltää kuvan 12 ER-mallin mukaisesti. Tietueiden *turnaus* ja *henkilo* välillä on ”M..N” -yhteys. Se tarkoittaa, että henkilö voi olla osallistuneena useaan turnaukseen ja kukin turnaus voi sisältää useita henkilöitä. Turnaus sisältää monta osallistujaa, mutta sama henkilö ei voi kuitenkaan olla osallistuneena kahta kertaa samaan turnaukseen. Jokaiselle osallistujalle on useita kierrostuloksia, niiden lukumäärä on enintään *turnaus*-tietueen kentän rounds mukainen.

4.2.3 Tuloslista

Listan esittely: std::list<unsigned int> lista

Turnauksen osallistujat ovat pelaajia ja tyypilliseen turnaukseen osallistuu monta pelaajaa. Kukin pelaaja on henkilö, joka voi olla osallistujana useassa turnauksessa. Tällaisten sidosten muodostaminen on toteutettu tallentamalla

kunkin osallistujan *henkilo*-tietueen ID:t *turnaus*-tietueen sisältämän tuloslistan, nimeltään lista, alkioihin (4.1.2).

alkiot 1-8

Pelaajien ID:t				Pelaajien vahvuudet			
Pelaajan 1 ID	Pelaajan 2 ID	Pelaajan 3 ID	Pelaajan 4 ID	Pelaajan 1 vahvuus	Pelaajan 2 vahvuus	Pelaajan 3 vahvuus	Pelaajan 4 vahvuus

alkiot 9-13

Turnaus kierros 1 : Pelipari 1				
Pelaajan 1 indeksi	Pelaajan 2 indeksi	Pelin lopputulos	Pelaajan 1 vahvuusmuutos	Pelaajan 2 vahvuusmuutos

alkiot 14-18

Turnaus kierros 1 : Pelipari 2				
Pelaajan 3 indeksi	Pelaajan 4 indeksi	Pelin lopputulos	Pelaajan 3 vahvuusmuutos	Pelaajan 4 vahvuusmuutos

alkiot 19-23

Turnaus kierros 2 : Pelipari 1				
Pelaajan 1 indeksi	Pelaajan 3 indeksi	Pelin lopputulos	Pelaajan 1 vahvuusmuutos	Pelaajan 3 vahvuusmuutos

alkiot 24-28

Turnaus kierros 2 : Pelipari 2				
Pelaajan 2 indeksi	Pelaajan 4 indeksi	Pelin lopputulos	Pelaajan 2 vahvuusmuutos	Pelaajan 4 vahvuusmuutos

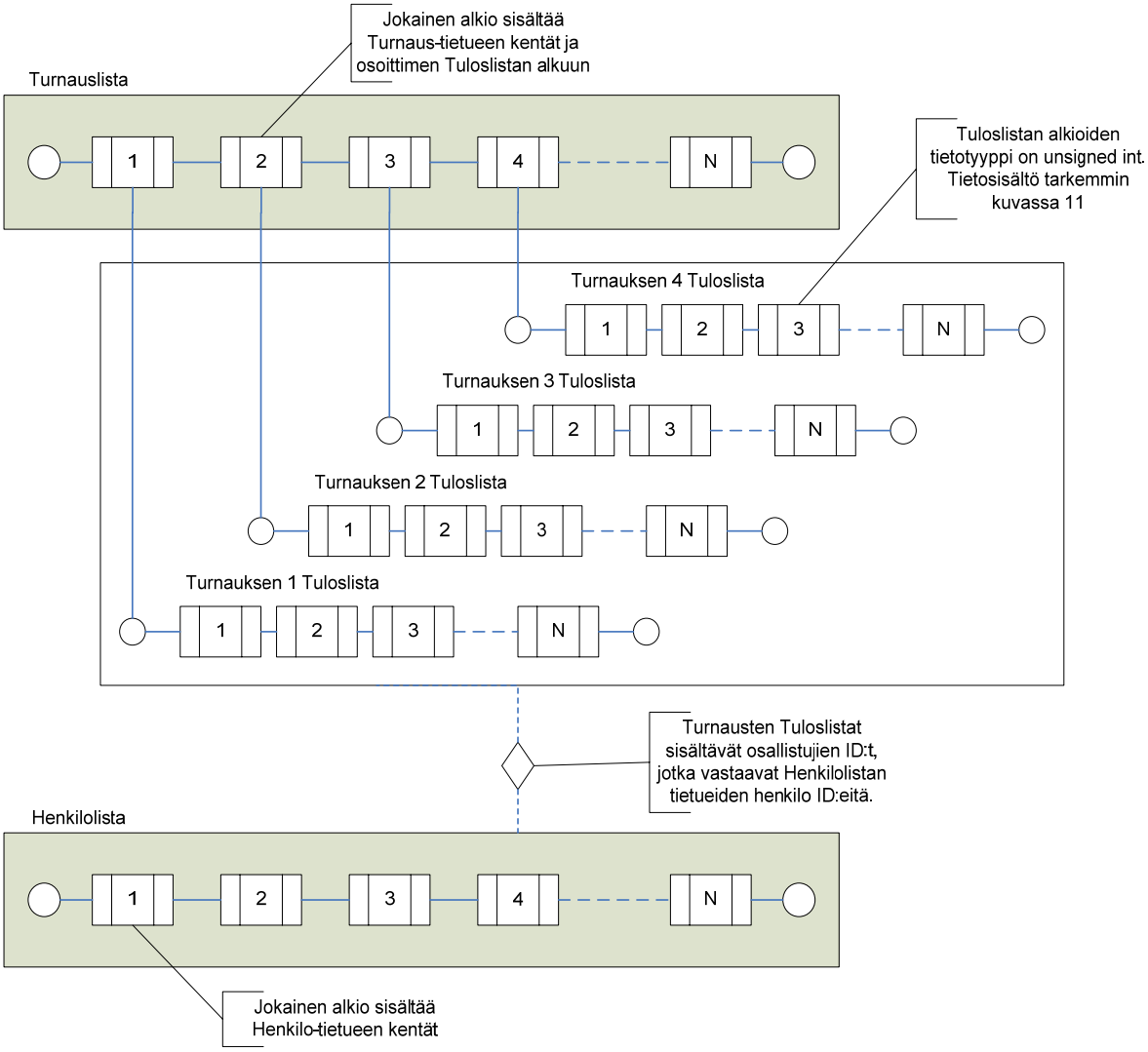
Kuva 13. Tuloslista

Tuloslista on nimensä mukaisesti lista, jonka sisällöstä löytyy turnaukseen osallistuvien pelaajien ID:t, vahvuusluvut, peliparit, tulokset ja vahvuuslukumuutokset. Tätä tietotyyppiä ei ole pilkottu osiin tai omaan tietueeseensa, sillä tästä listasta löytyvien tietojen järjestämistä voidaan tulla hallitsemaan erilaisin algoritmein. Tällaisen listan muodostus on saanut virikkeensä tuloslistasta (crosstable), joka sisältää oleelliset tiedot turnauksen pelituloksista. Esimerkki tällaisesta tuloslistasta liitteessä 1.

Kuvassa 14 on esimerkki tuloslistan sisällöstä, kun turnaukseen on osallistunut neljä pelaajaa ja he ovat pelanneet kaksi kierrosta. Kukin kenttä on tyyppiä *unsigned int* ja tuloslistan koko riippuu pelaajien ja pelikierrosten määrästä.

Vahvuuslukumuutokset lasketaan näihin alkioihin vain jäsenille, jotka ovat *established*-laskennan piirissä. Mikäli tietyllä pelaajalla ei ollut vastusta tietyllä pelikierroksella, tällöin hänen vastustajansa indeksiksi merkitään nolla ja myös lopputulos ja vahvuusmuutos alustetaan nolliksi. Merkinnällisesti tällainen

pelaaja pelasi vastusta BYE vastaan. Vaikka turnauksessa saakin voiton sellaisesta pelistä, sitä ei kuitenkaan hyväksytä vahvuuslukuhallinnassa pisteeksi. Vahvuuslukulaskennan kannalta turnauksen lopputulos ei ole niin oleellista kuin pelitulokset todellisten pelaajien välillä.



Kuva 14. Linkitettyjen listojen hierarkia

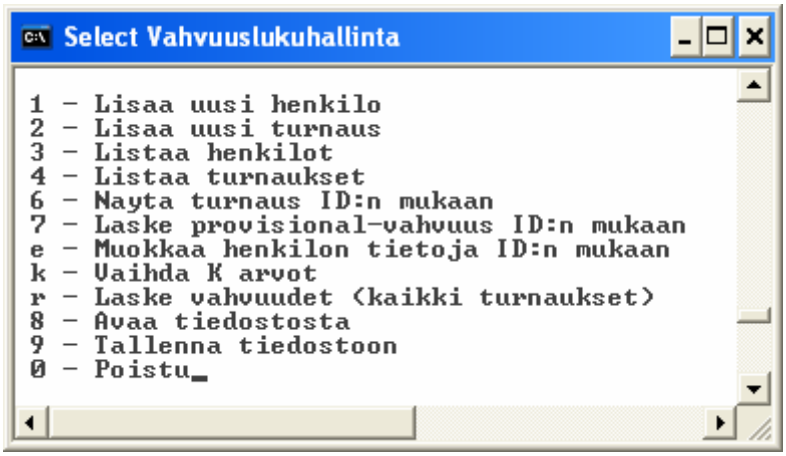
Turnauslistan kussakin tietueessa on kenttä nimeltään *lista*, joka määrittelee tuloslistan. Näin saadaan turnauslista ja tuloslista linkitettyä toisiinsa. Henkilölistan sidos turnauslistaan tapahtuu tuloslistan kautta siten, että tuloslistasta löytyvät pelaajien ID:t vastaavat henkilölistan pelaajien ID:eitä.

5 VAHVUUSLUKUHALLINNAN TOIMINTAA

Tässä luvussa tarkastellaan ohjelman toimintaa muutaman esimerkin avulla. Lisätään henkilöitä pelaajatietokantaan, muodostetaan kuvitteellinen turnaus ja tarkastellaan vahvuuslukujen muuttumista käyttäen todellisia pelituloksia *rating pool* -vaiheen turnauksista.

5.1 Tietojen syöttäminen

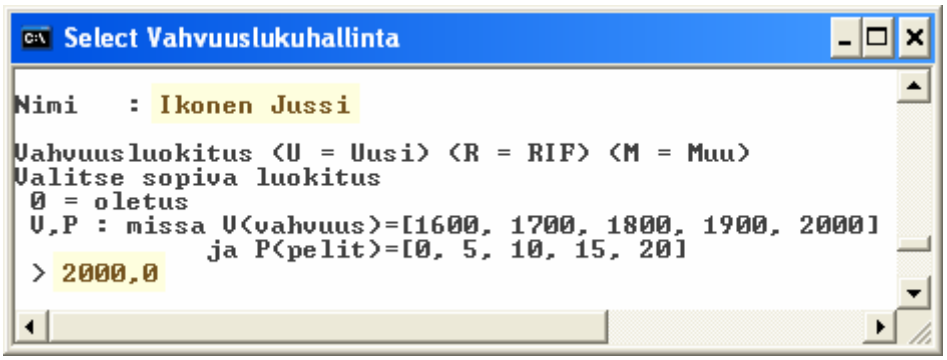
Vahvuuslukuhallinnan voi käynnistää Windowsin komentokehotteesta. Ensimmäisenä aukeaa päävalikko, jossa on esillä ohjelman keskeisimmät toiminnot.



Kuva 15. Päävalikko

Halutun toiminnon valinta tapahtuu painamalla näppäimistöltä asianmukaista numeroa tai kirjainta. Esimerkiksi painamalla numeroa nolla voi ohjelmasta poistua.

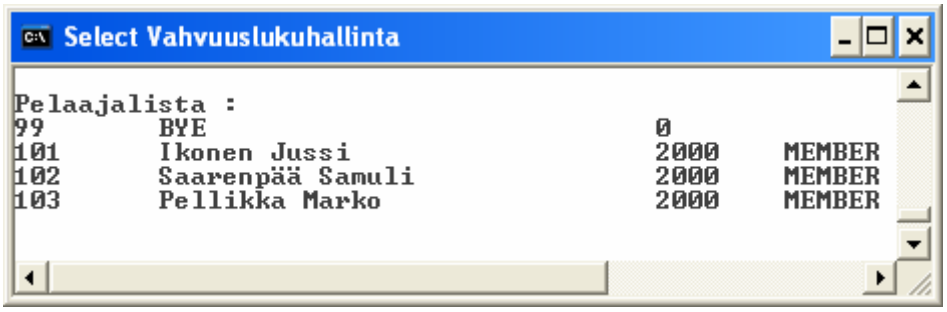
Seuraavaksi tarkastellaan, miten vahvuuslukuhallinnan henkilöiden lisääminen toimii käytännössä. Lisätään järjestelmään henkilöt Ikonen Jussi, Saarenpää Samuli ja Pellikka Marko. Esimerkissä käyttäjän syötteet ovat korostettuina vaaleankeltaisella pohjalla.



Kuva 16. Uuden henkilön lisääminen

Kuvassa 16 on lisättynä henkilö nimeltään Ikonen Jussi ja hänelle on annettu uusi vahvuusluokitus, jossa *rating pool* -vaiheen mukainen vahvuuslukema on 2000 ja pelimäärä 0. Mikäli *rating pool* -vaiheen lukemat halutaan sivuuttaa, niin nolla riittää antamaan oletusasetukset. Vaihtoehtoisesti voi antaa asetuksiksi ”1600,0”, mikä vastaa samaa.

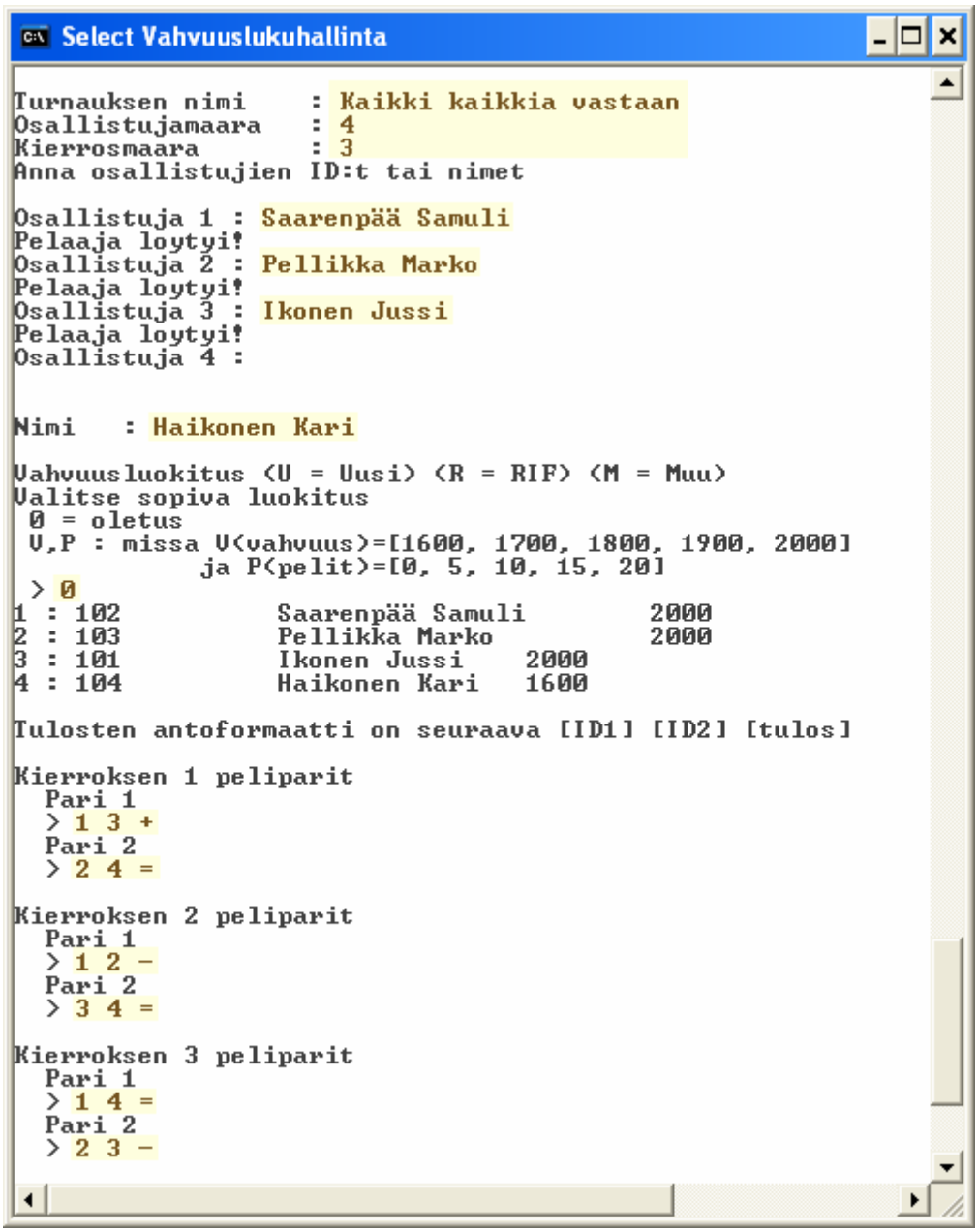
Lisätään vastaavasti vielä kaksi henkilöä, Saarenpää Samuli ja Pellikka Marko. Nyt kun tarkastellaan päävalikosta valittua toimintoa, joka listaa henkilöt, saadaan seuraava näkymä.



Kuva 17. Lisätyt henkilöt

Hetki sitten lisätyt henkilöt näkyvät pelaajalistassa. Heidän vahvuuslukuina näkyy tietojen syöttövaiheessa annetut vahvuusluvut, jokaisella 2000. Pelaajalistassa näkyy myös heidän ID:t ja jäsenyysluokituksensa MEMBER, mikä vastaa Suomen Renju ry:n jäsentä.

Seuraavaksi tehdään neljän henkilön turnaus, jossa kaikkien on tarkoitus pelata kaikkia vastaan. Tällainen turnaus tunnetaan myös nimikkeellä *round robin*.

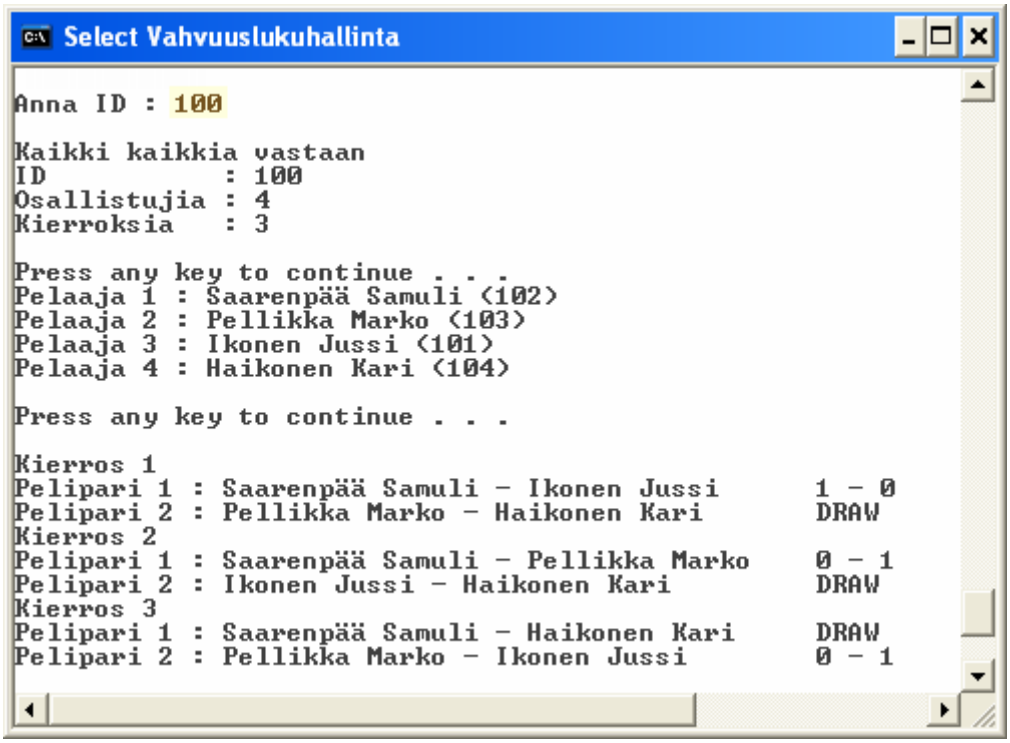


Kuva 18. Turnauksen lisäys

Turnaustiedoissa kysytään turnauksen nimi, osallistujamäärä ja kierrosmäärä. Tämän jälkeen käydään läpi pelaajat, jotka osallistuvat kyseiseen turnaukseen. Neljättä pelaajaa ei ollut vielä lisättynä. Se ei kuitenkaan haittaa, sillä uuden henkilön lisäys onnistuu myös turnaustietoja syötettäessä. Neljäs pelaaja on Haikonen Kari ja hänelle annetaan tässä esimerkissä oletusasetusten mukainen vahvuusluku, joka on 1600.

Tuloksien syöttäminen tapahtuu antamalla kunkin peliparin pelaajien ID:eitä osoittavien indeksien numerot ja lisäämällä merkki, joka osoittaa oliko kyseinen

peli voitto (+ / W), tasapeli (= / D) vai häviö (- / L). Mikäli syöttää vahingossa väärin, niin antamalla minkä tahansa laittomaa syntaksia noudattavan tulosrivin, kysyy ohjelma kyseisen kierroksen tulokset uudestaan.



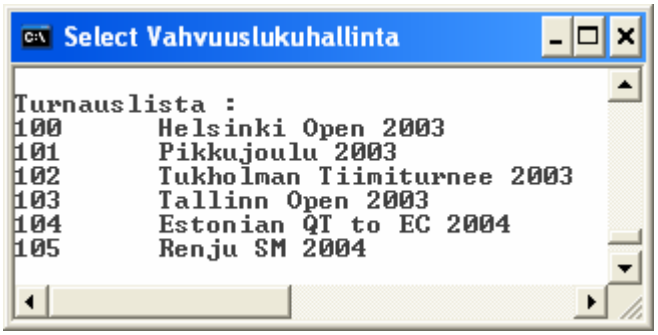
Kuva 19. Turnaustulosten raportointi

Kun turnaus on syötettynä järjestelmään, voidaan jälkikäteen tarkistaa minkä tahansa aiemmin syötetyn turnauksen perustiedot ja kierrostulokset.

5.2 Vahvuusluvun muuttuminen

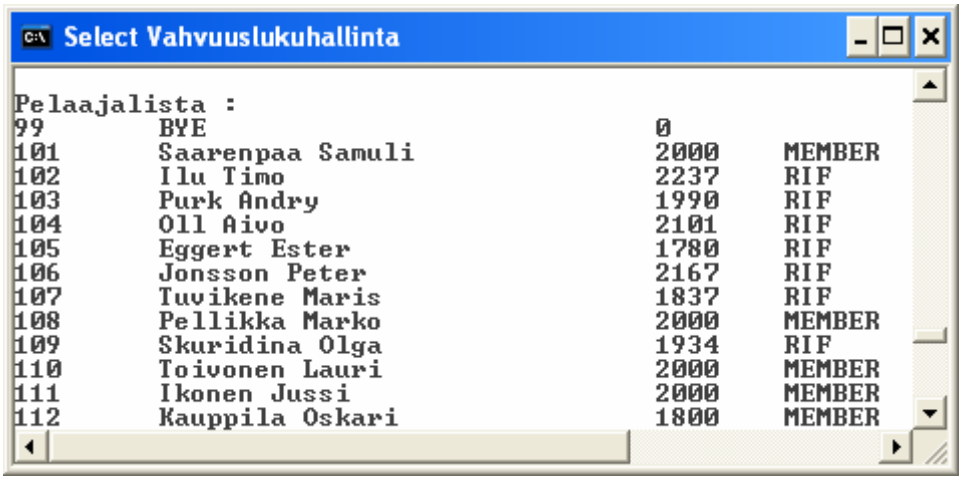
Vahvuuslukuhallinta laskee Suomen Renju ry:n jäsenille vahvuusluvut. Yksi tai kaksi turnausta ei yleensä vielä riitä siihen, että saataisiin tarpeeksi vahvuuslukukelpoisia pelejä, jotta ensimmäinen alkuasetuksista poikkeava vahvuusluku voitaisiin laskea. Rajoittavana tekijänä toimii *provisional*-vaiheen (2.1.2.2) laskenta, missä edellytyksenä vahvuusluvun julkaisemiselle ja samalla siirtymiselle *established*-laskennan puolelle on vähintään 20 vahvuuslukukelpoista turnauspeliä.

Otetaan tarkasteluun 6 ensimmäistä turnausta, jotka ovat myös hyväksytty *rating pool* -vaiheen käyttöön. Todellinen lista kokonaisuudessaan on nähtävillä liitteessä 4.



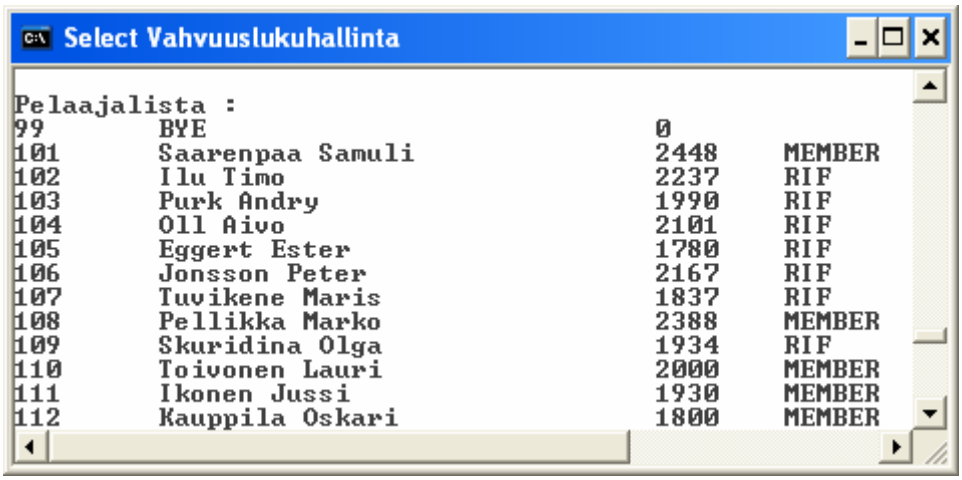
Kuva 20. Turnauslista

Kaikki kuvan 20 mukaiset turnaukset kierrostuloksineen ovat syötettyinä, joten voidaan lähteä laskemaan vahvuuslukumuutoksia Suomen Renju ry:n jäsenille.



Kuva 21. Vahvuusluvut ennen laskentaa

Kuvassa 21 esiintyy 5 jäsentä (Saarenpää Samuli, Pellikka Marko, Toivonen Lauri, Ikonen Jussi ja Kauppila Oskari), joiden vahvuusluvut ovat toistaiseksi alkuasetusten mukaisia. Listattuna on myös seitsemän RIF:n jäsentä (Ilu Timo, Purk Andry, Oll Aivo, Eggert Ester, Jonsson Peter, Tuvikene Maris ja Skuridina Olga), mutta heihin ei kiinnitetä tässä vaiheessa huomiota, koska vahvuuslukuhallinta ei laske muiden kuin Suomen Renju ry:n jäsenten vahvuuslukuja.



Kuva 22. Vahvuusluvut laskennan jälkeen

Nyt on nähtävissä, että seuraavilla pelaajilla on vahvuusluku noussut:

Pellikka Marko 2000 → 2388

Saarenpää Samuli 2000 → 2448

Yhdellä vahvuusluku on hieman laskenut:

Ikonen Jussi 2000 → 1930

Lauri Toivosella ja Oskari Kauppilalla vahvuusluku ei ole muuttunut. Tämä johtuu siitä, että heidän kohdallaan ei tarvittava 20 vahvuuslukukelpoisen turnauspelin lukumäärä ole vielä ylittynyt.

6 TYÖSSÄ KÄYTETYT TYÖKALUT

Vahvuuslukuhallinnan ohjelmakoodi on kirjoitettu käyttäen avoimen lähdekoodin Dev-c++ v4.9.9.2 ohjelmaa. Se on integroitu kehitysympäristö, mikä tarkoittaa sitä, että se sisältää toimivan ohjelman tuottamiseen erilaisia työkaluja. Tärkein työkalu on editori, joka mahdollistaa ohjelmakoodin kirjoittamisen. Ohjelmakoodin kääntämiseen voi käyttää jotain mukana tulleista kääntäjistä tai asentaa jonkun muun monista vaihtoehdoista. Työhön valittu C/C++ kääntäjä on Dev-c++ ohjelman mukana tullut MinGW 5.0.2. Käännetyn ohjelman toimintaa voi testata käyttämällä Dev-c++:n debuggeria, jonka avulla voi seurata ohjelman toimintaa ja muuttujien sisältöä rivi riviltä, ja virheellisen toiminnan aiheuttanut virhe pyritään paikallistamaan.

Ohjelman ylläpitämän pelaajatietokannan, joka tallentuu binääritiedostoon, testaamiseen käytettiin ohjelmaa HHD Software Free Hex Editor 3.12. Se on heksaeditori, jonka avulla voi tutkia tiedostoon tallentuvia kenttiä ja niiden sijainteja heksadesimaalimuodossa.

7 LOPPUSANAT

Työn tuloksena syntyi ohjelma, johon voi syöttää vahvuuslukulaskennan kannalta oleellisia jäsen- ja turnaustietoja. Ohjelma laskee pelaajien vahvuusluvut ja raportoi ne. Tulevissa renjuturnauksissa tulemme kirjoittamaan pelaajien nimilappuihin heidän kulloisenkin vahvuuslukunsa, mikä on yleistä esimerkiksi shakissa. Vahvuuslukujen esille tuonti on jo nyt tuonut paljon myönteistä palautetta. Tyytyväisiä ollaan myös oltu siihen, että vahvuusluvut ovat suuruusluokaltaan oikeantuntuisia.

Tulevaisuuden näkymissä on työn toteutus käyttäen avoimen lähdekoodin wxWidgets-grafiikkakirjastoa, joka mahdollistaa graafisen käyttöliittymän toteuttamisen ohjelmalle. Se parantaa ohjelman käytettävyyttä ja lisää yhteensopivuutta Windows:n ja Linux:n välillä. /8/

Vahvuuslukuhallintaan on suunnitteilla jo lisäominaisuuksia. Mahdollistetaan erillisen turnaushallintaohjelmiston käyttäminen, jolta saadaan turnaustulokset suoraan, eikä tarvitse käsin syöttää tietoja. Lisäksi yksi tärkeä ominaisuus, jolla ohjelman käyttämä binääritietokanta saadaan muunnettua MySQL-tiedostoksi. Se tapahtuu luomalla ohjelmasta tekstitiedosto, joka sisältää asianmukaiset SQL-luontikäskyt. Tämä on erityisen hyödyllistä, sillä se mahdollistaa kattavamman online-jäsentietokannan. Tietokantaa suunnittelee Suomen Renju ry:n tämän hetkinen puheenjohtaja Jussi Ikonen.

LÄHTEET

1. RIF General Assembly 1996 pöytäkirja, 05/1996
2. FIVE-IN-A-ROW RENJU For Beginners to Advanced Players
   By Goro Sakata & Wataru Ikawa,
   THE ISHI PRESS, INC. Tokio 1981
3. Elo vahvuuslukulaskennasta [www-sivu][viitattu 2.3.2007]
   Saatavissa: http://en.wikipedia.org/wiki/ELO_rating_system
4. THE RATING OF CHESSPLAYERS, PAST AND PRESENT
   By ÁRPÁD E. ELO 0-7134-1860-5, 1978
5. The Legacy of Árpád Elo
   The Development of a Chess-Rating System
   Universiteit van Amsterdam
   VRT-2 Sandra de Blécourt 1.12.1998
6. Jeff Sonasin vahvuuslukulaskentaa käsittelevä sivusto. [www-sivu][viitattu 13.1.2007] Saatavissa: http://db.chessmetrics.com
7. Virallinen RIF vahvuuslukusivusto. [www-sivu][viitattu 4.2.2007]
   Saatavissa: http://stepanov.lk.net/rifrat/rifrat.html
8. wxWidgets [www-sivu][viitattu 2.3.2007]
   Saatavissa: http://www.wxwidgets.org
9. C++ ja olio-ohjelmointi, Päivi Hietanen 1999, neljäs painos

LIITELUETTELO

LIITE 1: Crosstable

Esimerkki todellisesta turnaustuloslistasta. /7/

Helsinki Open 2003 (20-21.09.2003)

	N Player	1	2	3	4	5	6	7	Po	R ±
	1. Saarenpaa Samuli	2 1	8 1	4 1	3 1	5 1	11 1	6 1	7	2442
	2. Ilu Timo	1 0	9 1	13 1	14 1	3 1	4 1	5 1	6	+23
	3. Purk Andri	11 1	6 1	7 1	1 0	2 0	12 1	4 1	5	+48
	4. Oll Aivo	10 1	5 1	1 0	11 1	6 1	2 0	3 0	4	+3
	5. Eggert Ester	8 1	4 0	12 1	13 1	1 0	14 1	2 0	4	+18
	6. Jonsson Peter	15 1	3 0	16 1	7 1	4 0	10 1	1 0	4	-31
	7. Tuvikene Maris	+	12 1	3 0	6 0	9 0	13 1	11 1	4	-25
	8. Pellikka Marko	5 0	1 0	9 1	15 1	11 0	+	16 1	4	-10
	9. Skuridina Olga	12 0	2 0	8 0	+	7 1	16 1	15 1	4	-7
	10. Toivonen Lauri	4 0	+	11 0	12 1	16 1	6 0	17 1	4	-19
	11. Ikonen Jussi	3 0	15 1	10 1	4 0	8 1	1 0	7 0	3	1898
	12. Kauppila Oskari	9 1	7 0	5 0	10 0	17 1	3 0	14 1	3	1673
	13. Kokkonen Eevert	17 1	16 1	2 0	5 0	14 0	7 0	+	3	1551
	14. Pinola Tuomo	16 0	17 1	15 1	2 0	13 1	5 0	12 0	3	1605
	15. Siiria Simo	6 0	11 0	14 0	8 0	+	17 1	9 0	2	1626
	16. Horelli Lauri	14 1	13 0	6 0	17 1	10 0	9 0	8 0	2	1616
	17. Sneitz Nina	13 0	14 0	+	16 0	12 0	15 0	10 0	1	1589

LIITE 2: Crosstablen lukuohjeet

+	11. Ikonen Jussi	$_{3}0$	$_{15}1$	$_{10}1$	$_{4}0$	$_{8}1$	$_{1}0$	$_{7}0$	3	**1898**

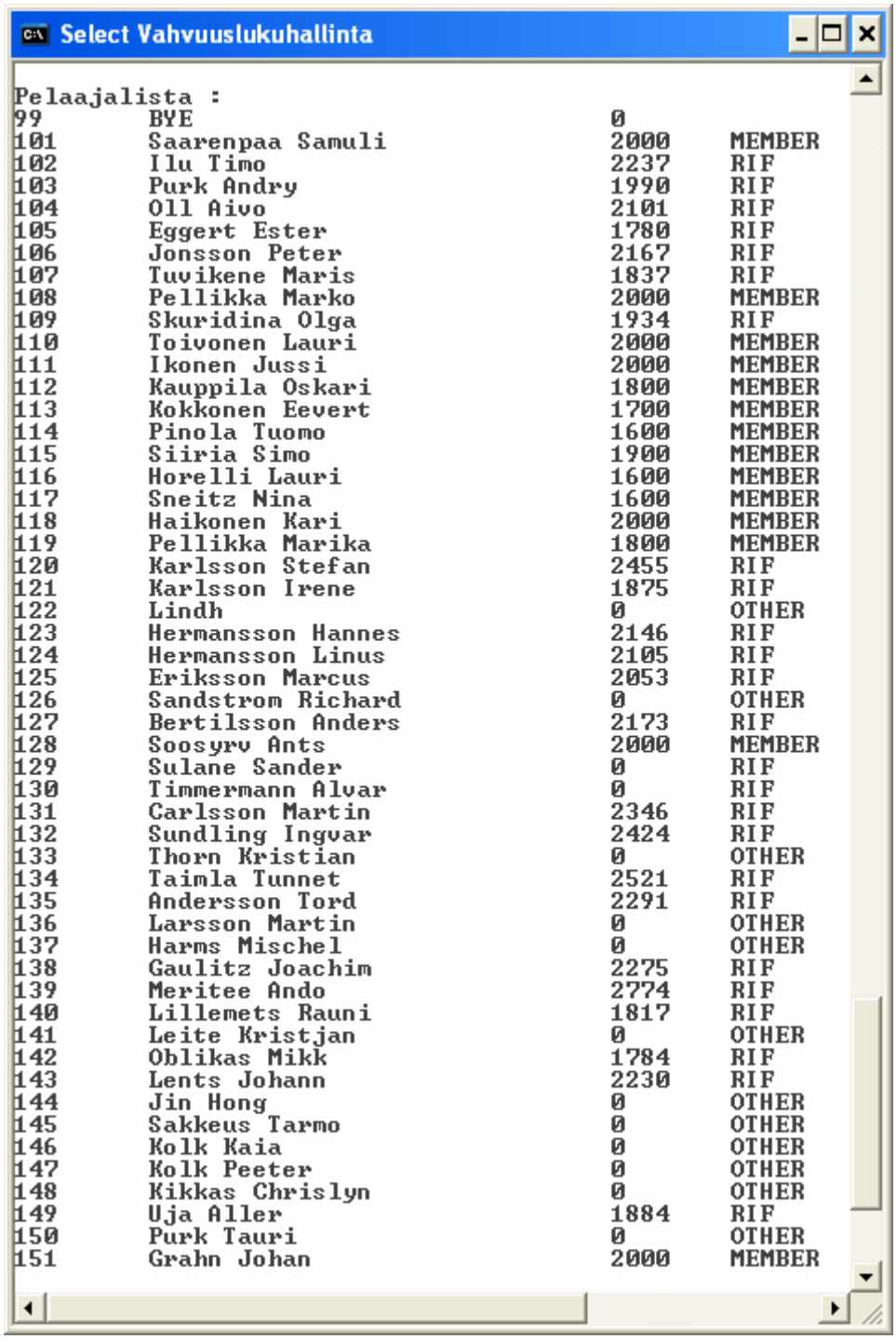
Crosstable sisältää kunkin pelaajan kaikki kierrostulokset ja se on järjestetty pelaajien lopputuloksien mukaan. Tulos-lähtökohtainen järjestäminen ei ole välttämätöntä, mutta helpottaa monesti luettavuutta. Turnauksessa 11. oli pelaaja nimeltään Jussi Ikonen. Ensimmäisellä kierroksella hän pelasi pelaajaa numero kolme vastaan (huomaa alaindeksi). Hän hävisi pelin, joten hänelle merkittiin kierroksesta 0 pistettä. Toisella kierroksella hän pelasi pelaajaa numero 15 vastaan. Hän voitti sen pelin. Turnauksessa pelattiin yhteensä 7 kierrosta. Kierroksia seuraava numero 3 tarkoittaa hänen kokonaispistesaldoaan. Koska tämä taulukko on otettu viralliselta RIF-vahvuuslukusivustolta, niin siitä löytyy myös merkintä turnauksen jälkeisestä vahvuusluvusta. Jussin RIF-vahvuusluku oli turnauksen jälkeen 1898. /7/

LIITE 3: Rating pool -käyttöön hyväksytyt turnaukset

Lista aikajärjestyksessä:

HO2003	Helsinki Open 2003
PCP2003	Pre-Christmas party 2003
OSTC2003	Open Swedish Team Championships 2003
TO2003	Tallinn Open 2003
EECQT2004	Estonian European Championship Qualification Tournament
FC2004	Finnish Championship 2004
WO2004	Warsaw Open 2004
KO2004	Karepa Open 2004
HO2004	Helsinki Open 2004
OSTC2004	Open Swedish Team Championships 2004
PCP2004	Pre-Christmas party 2004
TO2004	Tallinn Open 2004
BL2005	Baltic League 2005
PO2005	Poland Open 2005
FC2005	Finnish Championship 2005
HO2005	Helsinki Open 2005
OSTC2005	Open Swedish Team Championships 2005
TO2005	Tallinn Open 2005
TM2006	Tampere Masters 2006
TWC2006	Team WC 2006
BL2006	Baltic League 2006

LIITE 4: Pelaajalista



LIITE 5: Makefile

```
# Project: rank
# Makefile created by Dev-C++ 4.9.9.2

CPP  = g++.exe
CC   = gcc.exe
WINDRES = windres.exe
RES  =
OBJ  = main.o prompt.o tournament.o player.o file.o rating.o $(RES)
LINKOBJ  = main.o prompt.o tournament.o player.o file.o rating.o $(RES)
LIBS =  -L"C:/Dev-Cpp/lib"
INCS =  -I"C:/Dev-Cpp/include"
CXXINCS =  -I"C:/Dev-Cpp/lib/gcc/mingw32/3.4.2/include"  -I"C:/Dev-
Cpp/include/c++/3.4.2/backward"  -I"C:/Dev-Cpp/include/c++/3.4.2/mingw32"  -I"C:/Dev-
Cpp/include/c++/3.4.2"  -I"C:/Dev-Cpp/include"
BIN  = rank.exe
CXXFLAGS = $(CXXINCS)
CFLAGS = $(INCS)
RM = rm -f

.PHONY: all all-before all-after clean clean-custom

all: all-before rank.exe all-after


clean: clean-custom
      ${RM} $(OBJ) $(BIN)

$(BIN): $(OBJ)
      $(CPP) $(LINKOBJ) -o "rank.exe" $(LIBS)

main.o: main.cpp
      $(CPP) -c main.cpp -o main.o $(CXXFLAGS)

prompt.o: prompt.cpp
      $(CPP) -c prompt.cpp -o prompt.o $(CXXFLAGS)

tournament.o: tournament.cpp
      $(CPP) -c tournament.cpp -o tournament.o $(CXXFLAGS)

player.o: player.cpp
      $(CPP) -c player.cpp -o player.o $(CXXFLAGS)

file.o: file.cpp
      $(CPP) -c file.cpp -o file.o $(CXXFLAGS)

rating.o: rating.cpp
      $(CPP) -c rating.cpp -o rating.o $(CXXFLAGS)
```

LIITE 6: struct.h

```
#ifndef STRUCT_H
#define STRUCT_H

/*
    Henkilo-tietue sisältää id:n, pelaajan nimen, pelilukumäärän, vahvuusluvun,
    vahvuusluokituksen ja alkuperäiset rating pool -arvot.
*/

typedef struct
{
    unsigned int id;
    char nimi[32];
    unsigned int games;
    unsigned int rating;
    unsigned int rstat;   // 0 = Ei tiedossa, 1 = RIF, 2 = SuRe
    unsigned int orig[2]; // Rating pool -arvot
} henkilo;

/*
    Turnaus-tietue sisältää id:n, turnauksen nimen, vahvuusluokituksen
    mukaisen K-arvon, päiväyksen, osallistujamäärän, kierroslukumäärän ja
    käytetyn peliparien muodostus menetelmän.
*/

typedef struct
{
    unsigned int id;
    char nimi[32];
    unsigned int rating_status; // Jokin seuraavista 32/24/16/12/8/6/4/3/2/0

    unsigned int pvm;
    unsigned int participants;
    unsigned int rounds;
    unsigned int system;   // 0 = Ei tiedossa, 1 = swiss, 2 = round robin
    std::list<int> lista;  // Tuloslista
} turnaus;

#endif
```

LIITE 7: main.cpp

```
#include <cstdlib>
#include <conio.h>
#include <list>

#include "struct.h"
#include "file.h"
#include "player.h"
#include "rating.h"
#include "tournament.h"

using namespace std;

int menu(list<henkilo>& listh, list<turnaus>& listt);

int main(int argc, char *argv[])
{
    std::list<henkilo> listh;
    std::list<turnaus> listt;

    system("title Vahvuuslukuhallinta");
    system("mode con: cp select=850");
    CreateBYE(listh);

    menu(listh, listt);

    system("PAUSE");
    return EXIT_SUCCESS;
}

int menu(list<henkilo>& listh, list<turnaus>& listt)
{
    int me, n;

      // main loop
    for(;;)
    {
        printf("\n");
        printf("\n 1 - Lisaa uusi henkilo");
        printf("\n 2 - Lisaa uusi turnaus");
        printf("\n 3 - Listaa henkilot");
        printf("\n 4 - Listaa turnaukset");
        printf("\n 6 - Nayta turnaus ID:n mukaan");
        printf("\n 7 - Laske provisional-vahvuus ID:n mukaan");
        printf("\n e - Muokkaa henkilon tietoja ID:n mukaan");
        printf("\n k - Vaihda K arvot");
        printf("\n r - Laske vahvuudet (kaikki turnaukset)");
        printf("\n 8 - Avaa tiedostosta");
        printf("\n 9 - Tallenna tiedostoon");
        printf("\n 0 - Poistu");

        me = 0;
        while(!me) me = getch();

        switch(me)
        {
            case '1' : AddPlayer(listh);
                break;

            case '2' : AddTournament(listt, listh);
                break;

            case '3' : ShowPlayers(listh);
                break;

            case '4' : ShowTournaments(listt);
                break;

            case '6' :
                printf("\nAnna ID : "); scanf("%d", &n);
                ShowTournamentByID(listt, listh, n);
                break;
```

```
            case '7' :
                printf("\nAnna ID : "); scanf("%d", &n);
                CalculateProvisional(listh, listt, n, 105);
                break;

            case 'e' :
                printf("\nAnna ID : "); scanf("%d", &n);
                EditPlayerByID(listh, n);
                break;

            case 'k' : UpdateKValues(listt);
                break;

            case 'r' :
                CalculateRatingsByTournament(listh, listt);
                break;

            case '8' : OpenAll(listh, listt);
                break;

            case '9' : SaveAll(listh, listt);
                break;

            case '0' : return 0;
                break;

            default : break;
        }
    }
    return 0;
}
```

LIITE 8: player.h

```
#ifndef PLAYER_H
#define PLAYER_H

using namespace std;

unsigned int AddPlayer(list<henkilo>& listh);
void ChangeRatingStatusByID(list<henkilo>& listh, unsigned int id,
                            unsigned int status);

void CreateBYE(list<henkilo>& listh);
void EditPlayerByID(list<henkilo>& listh, unsigned int id);
void RemovePlayerByID(list<henkilo>& listh, unsigned int id);
char * SearchNameByID(list<henkilo>& listh, unsigned int id);
unsigned int SearchRatingByID(list<henkilo>& listh, unsigned int id);
unsigned int SearchRatingStatusByID(list<henkilo>& listh, unsigned int id);
int ShowPlayers(list<henkilo>& listh);

#endif
```

LIITE 9: player.cpp

```
#include <cstdlib>
#include <list>
#include <conio.h>

#include "struct.h"
#include "tournament.h"

using namespace std;

enum members {HONORARY = 1, MEMBER = 2, RIF = 3, OTHER = 4};

/* Lisää uusi pelaaja/henkilo  */

unsigned int AddPlayer(list<henkilo>& listh)
{
    henkilo temp;
    int odota;
    char nappain;
    char resu[12];
    char *pch;

    memset(temp.nimi, 0, sizeof(temp.nimi));

    // pelaajien ID:t numeroidaan juoksevasti 100:sta eteenpäin
    temp.id = 100 + listh.size();
    printf("\n\n");
    fflush(stdin);
    printf("Nimi   : "); gets(temp.nimi);

    for(odota = 0; !odota;)
    {
        printf("\nVahvuusluokitus (U = Uusi) (R = RIF) (M = Muu)");
        nappain = 0;
        while(!nappain) nappain = getch();
        odota = 1;
        switch(nappain)
        {
        // tapaus Uusi jäsen
        case 'u' :
            temp.rstat = MEMBER;
            printf("\nValitse sopiva luokitus");
            printf("\n 0 = oletus");
            printf("\n V,P : missa V(vahvuus)=[1600, 1700, 1800, 1900, 2000] ");
            printf("\n            ja P(pelit)=[0, 5, 10, 15, 20]");
            printf("\n > ");
            fflush(stdin);
            gets(resu);
            pch = strtok(resu, " ,"); // erottimena pilkku ja välilyönti

            if(pch)
            {
                if(atoi(pch) == 0)  // asetetaanko oletukset?
                {
                    temp.orig[0] = temp.rating = 1600;
                    temp.orig[1] = temp.games = 0;
                }
                else // muutoin rating pool vaiheen mukaiset asetukset
                {
```

```
                    // Tarkistetaan onko annettu vahvuuslukema OK
                    if(atoi(pch) == 1600 || atoi(pch) == 1700 ||
                        atoi(pch) == 1800 || atoi(pch) == 1900 ||
                        atoi(pch) == 2000)
                    {
                        temp.orig[0] = temp.rating = atoi(pch);
                    }
                    else
                    {
                        printf("Virhe");
                        temp.orig[0] = temp.rating = 1600;
                    }
                    pch = strtok(NULL, " ,");

                    if(pch)
                    {
                        // Tarkistetaan onko annettu pelilkm OK
                        if(atoi(pch) == 0 || atoi(pch) == 5 || atoi(pch) == 10
                            || atoi(pch) == 15 || atoi(pch) == 20)
                        {
                            temp.orig[1] = temp.games = atoi(pch);
                        }
                        else
                        {
                            printf("Virhe");
                            temp.orig[1] = temp.games = 0;
                        }
                    }
                    else
                    {
                        printf("Virhe");
                        temp.orig[1] = temp.games = 0;
                    }
                }
            }
            else // Annetaan oletusasetukset
            {
                printf("Virhe");
                temp.orig[0] = temp.rating = 1600;
                temp.orig[1] = temp.games = 0;
            }
            break;

        // tapaus RIF:n jäsen
        case 'r' :
            temp.rstat = RIF;
            printf("Vahvuusluku : ");
            gets(resu);
            temp.rating = atoi(resu);
            temp.orig[0] = temp.orig[1] = temp.games = 0;
            break;

        // tapaus muu, undefined
        case 'm' :
            temp.rstat = OTHER;
            temp.orig[0] = temp.orig[1] = temp.rating = temp.games = 0;
            break;

        default : odota = 0; break;

        }
    }

    // talletetaan pelaaja henkilo-listaan
    listh.push_back(temp);

    return temp.id;
}
```

```
/*  Palauta henkilo:n, jonka ID on tiedossa, vahvuusluokitus */

void ChangeRatingStatusByID(list<henkilo>& listh, unsigned int id,
                            unsigned int status)
{
    list<henkilo>::iterator hi;
    for(hi = listh.begin(); hi != listh.end(); hi++)
        if(id == hi->id) hi->rstat = status;
}

/*
    Luo BYE pelaaja. Tarvitaan turnauspeliparien muodostuksessa,
    jos pelaajien lukumäärä on pariton
*/

void CreateBYE(list<henkilo>& listh)
{
    henkilo temp;
    strcpy(temp.nimi, "BYE");
    temp.rstat = 0;
    temp.rating = 0;
    temp.id = 99; // special reserved ID for BYE
    listh.push_back(temp);
}

/*  Muutetaan tietyn pelaajan/henkilo:n tietoja */

void EditPlayerByID(list<henkilo>& listh, unsigned int id)
{
    char name[30];
    unsigned int i;
    list<henkilo>::iterator hi;

    // Etsitään ID:n mukainen henkilo listasta
    for(hi = listh.begin(); hi != listh.end(); hi++)
    {
        if(id == hi->id)
        {
            // Muutetaan nimi
            printf("\nNimi : %s", hi->nimi);
            printf("\nNimi : ");

            fflush(stdin);
            gets(name);
            strcpy(hi->nimi, name);

            // Muutetaan vahvuusluku
            printf("\nVahvuusluku : %d", hi->rating);
            printf("\nVahvuusluku : ");

            fflush(stdin);
            scanf("%d", &i);
            hi->rating = hi->orig[0] = i;

            // Muutetaan pelilukumäärä
            printf("\nPelit : %d", hi->games);
            printf("\nPelit : ");

            fflush(stdin);
            scanf("%d", &i);
            hi->games = hi->orig[1] = i;

            printf("\nTietojen muuttaminen valmis\n");
        }
    }
}
```

```
/* Henkilo:n poistaminen */

void RemovePlayerByID(list<henkilo>& listh, unsigned int id)
{
    list<henkilo>::iterator hi;

    // Etsitään poistettava henkilö henkilo-listasta
    for(hi = listh.begin(); hi != listh.end(); hi++)
        if(id == hi->id) break;

    // voidaan poistaa vain jos pelaajalla ei ole rekisteröityjä pelejä
    if(hi->games == 0) listh.erase(hi);
    else printf("\nEi voitu poistaa!");
}

/*  Palauttaa henkilo:n, jolla on tietty ID, nimen */

char * SearchNameByID(list<henkilo>& listh, unsigned int id)
{
    list<henkilo>::iterator hi;
    static char NA[4] = {'N', '/', 'A', '\n'}; // not available string
    for(hi = listh.begin(); hi != listh.end(); hi++)
        if(id == hi->id) return hi->nimi;

    return NA;
}

/* Palauttaa henkilo:n, jolla tietty ID, vahvuusluvun */

unsigned int SearchRatingByID(list<henkilo>& listh, unsigned int id)
{
    list<henkilo>::iterator hi;
    for(hi = listh.begin(); hi != listh.end(); hi++)
        if(id == hi->id) return hi->rating;

    return 0;
}

/* Palauttaa henkilo:n, jolla tietty ID, vahvuusluokituksen */

unsigned int SearchRatingStatusByID(list<henkilo>& listh, unsigned int id)
{
    list<henkilo>::iterator hi;
    for(hi = listh.begin(); hi != listh.end(); hi++)
        if(id == hi->id) return hi->rstat;

    return 0;
}

/* Tulostaa tallennettujen henkilo:iden perustiedot */

int ShowPlayers(list<henkilo>& listh)
{
    int temp;
    list<henkilo>::iterator ti;

    printf("\n\n");
    printf("Pelaajalista : \n");

    for(ti = listh.begin(); ti != listh.end(); ti++)
    {
        printf("%d \t %s", ti->id, ti->nimi, ti->rating);

        temp = 4 - (strlen(ti->nimi) + 1) / 8;
        for(;temp--;) printf("\t");
        printf("%d\t", ti->rating);

        switch(ti->rstat)
```

```
        {
            case RIF : printf("RIF"); // RIF
                break;

            case OTHER : printf("OTHER"); // Määrittelemätön tai ei tiedossa
                break;

            case MEMBER : printf("MEMBER"); // Jäsen
                break;

            case HONORARY : printf("HONORARY MEMBER"); // Kunniajäsen
                break;

            default: break;
        }
        printf("\n");
    }

    return 0;
}
```

LIITE 10: tournament.h

```
#ifndef TOURNAMENT_H
#define TOURNAMENT_H

using namespace std;

int AddTournament(list<turnaus>& listt, list<henkilo>& listh);
int CleanMemberRatings(list<henkilo>& listh);
int ShowTournamentByID(list<turnaus>& listt, list<henkilo>& listh,
                       unsigned int id);

int ShowTournaments(list<turnaus>& listt);
int UpdateKValues(list<turnaus>& listt);
int UpdateRatingFields(list<henkilo>& listh, list<turnaus>& listt,
                       unsigned int id);

#endif
```

LIITE 11: tournament.cpp

```
#include <cstdlib>
#include <list>
#include "struct.h"
#include "player.h"
#include "file.h"

using namespace std;

enum results {WIN = 1, LOSE = 2, DRAW = 3, OPEN = 4, EMPTY = 5};
enum members {HONORARY = 1, MEMBER = 2, RIF = 3, OTHER = 4};

/* Turnauksen lisäys */

int AddTournament(list<turnaus>& listt, list<henkilo>& listh)
{
    int i, j, rdy, virhe;
    char osa[30];
    char resu[12];
    unsigned int ratings[100];
    unsigned int uusi;
    char *pch;
    turnaus temp;
    list<henkilo>::iterator hi;
    list<int>::iterator ni;
    list<int>::iterator nj;

    memset(temp.nimi, 0, sizeof(temp.nimi));

    temp.id = 100 + listt.size();

    fflush(stdin);
    printf("\n\n");
    printf("Turnauksen nimi    : "); gets(temp.nimi);
    printf("Osallistujamaara   : "); scanf("%d", &(temp.participants));
    printf("Kierrosmaara       : "); scanf("%d", &(temp.rounds));

    printf("Anna osallistujien ID:t tai nimet\n");

    temp.lista.clear();
    fflush(stdin);

    // Käydään läpi kaikki osallistujat
    for(i = 0; i < temp.participants; i++)
    {
        fflush(stdin);
        printf("\nOsallistuja %d : ", i + 1); gets(osa);
        if(atoi(osa))
        {
            for(hi = listh.begin(); hi != listh.end(); hi++)
            {
                if(hi->id == atoi(osa)) // pelaaja löytyi
                {
                    temp.lista.push_back(atoi(osa));
                    ratings[i] = SearchRatingByID(listh, hi->id);
                }
                else hi--;
            }
        }
        else
        {
            rdy = 0;
            for(hi = listh.begin(); hi != listh.end() && !rdy; hi++)
            {
```

```
                // Tarkistetaan ettei ole toista saman nimistä pelaajaa
                if(!strcmp(osa, hi->nimi))
                {
                    temp.lista.push_back(hi->id);
                    ratings[i] = hi->rating;

                    if(hi->rstat == OTHER)
                    {
                        printf("\nPelaajalla ei ole RIF statusta, annetaanko");
                        printf(" se? (Y/N) : ");
                        gets(resu);

                        if(!strcmp(resu, "y") || !strcmp(resu, "Y"))
                            hi->rstat = RIF;
                    }

                    if(hi->rstat == RIF)
                    {
                        printf("\nAlkuperäinen RIF vahvuusluku on %d, anna");
                        printf(" uusi ");
                        printf("(0 = pidä nykyinen)\n > :", ratings[i]);
                        gets(resu);

                        if(atoi(resu)) ratings[i] = atoi(resu);
                    }

                    rdy = 1;
                    printf("Pelaaja loytyi!");
                }

            }

            // Jos valmis, lisätään henkilö turnauksen osallistujaksi
            if(!rdy)
            {
                uusi = AddPlayer(listh);
                ratings[i] = SearchRatingByID(listh, uusi);
                temp.lista.push_back(uusi);
            }
        }
    }

    // tulostetaan osallistujat muodossa
    // 1 : ID   NIMI
    // 2 : ID   NIMI
    // 3 : ID   NIMI
    //     ...

    ni = temp.lista.begin();
    for(i = 0; i < temp.participants; i++)
    {
        printf("%d : %d \t %s \t %d\n", i + 1, *ni, SearchNameByID(listh, *ni),
            ratings[i]);
        temp.lista.push_back(SearchRatingByID(listh, *ni));
        ni++;
    }
    if(i%2) printf("0 : BYE\n");

    // Tulostetaan osallistujalista myös tiedostoon
    WriteParticipants(temp.lista, listt, listh, temp.participants);

    printf("\nTulosten antoformaatti on seuraava [ID1] [ID2] [tulos]\n");


    // Käydään läpi kaikki pelikierrokset
    for(i = 0; i < temp.rounds; i++)
    {
        printf("\nKierroksen %d peliparit\n", i + 1);
```

```
        virhe = 0;
        for(j = 0; j < (temp.participants + 1) / 2; j++)
        {
            printf("  Pari %d\n  > ", j + 1);
            fflush(stdin);
            gets(resu);  // formaatti [ID1 ID2 +/-/=/w/l/d/W/L/D]

            pch = strtok(resu, " ");
            if(pch)
            {
                if(atoi(pch) == 0) temp.lista.push_back(0); // BYE
                else if(atoi(pch) - 1 >= 0 && atoi(pch) - 1 < temp.participants)
                    temp.lista.push_back(atoi(pch));
                else virhe = 1;
            } else temp.lista.push_back(90); // VIRHE

            pch = strtok(NULL, " ");
            if(pch)
            {
                if(atoi(pch) == 0) temp.lista.push_back(0); // BYE
                else if(atoi(pch) - 1 >= 0 && atoi(pch) - 1 < temp.participants)
                    temp.lista.push_back(atoi(pch));
                else virhe = 1;
            } else temp.lista.push_back(90); // VIRHE

            pch = strtok(NULL, " ");
            if(pch)
            {
                // Merkit "+", "W", "w" tarkoittavat voittoa
                if(!strcmp(pch, "+") || !strcmp(pch, "W") || !strcmp(pch, "w"))
                {
                    temp.lista.push_back(WIN);
                }

                // Merkit "-", "L", "l" tarkoittavat häviötä
                else if(!strcmp(pch, "-") || !strcmp(pch, "L") ||
                        !strcmp(pch, "l"))
                {
                    temp.lista.push_back(LOSE);
                }

                // Merkit "=", "D", "d" tarkoittavat tasapeliä
                else if(!strcmp(pch, "=") || !strcmp(pch, "D") ||
                        !strcmp(pch, "d"))
                {
                    temp.lista.push_back(DRAW);
                }

                // Muussa tapauksessa pelin lopputulos on määrittelemätön
                else
                {
                    temp.lista.push_back(EMPTY);
                    virhe = 1;
                }
            } else temp.lista.push_back(EMPTY);

            // Lisätään alustavasti tyhjät kentät vahvuuslukumuutoksille
            temp.lista.push_back(0);
            temp.lista.push_back(0);
        }

        // Virheen sattuessa poistetaan ylimääräiset alkiot
        if(virhe)
        {
            i--;
            for(j = 0; j < 5 * ((temp.participants + 1) / 2); j++)
                temp.lista.pop_back();
        }
```

```
    }

    // Viimeisin turnaus lisätään turnaus-listaan
    listt.push_back(temp);

    return 0;
}

// Tyhjennetään vahvuusluvut ja asetetaan alkuperäiset rating pool -asetukset
int CleanMemberRatings(list<henkilo>& listh)
{
    list<henkilo>::iterator hi;

    for(hi = listh.begin(); hi != listh.end(); hi++)
    {
        if(hi->rstat == MEMBER || hi->rstat == HONORARY)
        {
            hi->rating = hi->orig[0];
            hi->games = hi->orig[1];
        }
    }

    return 0;
}

// Tulostetaan ID:tä vastaavan turnauksen tiedot
int ShowTournamentByID(list<turnaus>& listt, list<henkilo>& listh,
                       unsigned int id)
{
    int i, j;
    list<turnaus>::iterator ti;
    list<henkilo>::iterator hi;
    list<int>::iterator ni;

    unsigned int plys[30];

    // Käydään läpi listassa olevia turnauksia järjestäin kunnes oikea löytyy
    for(ti = listt.begin(); ti != listt.end() && id != ti->id ;) ti++;

    if(ti == listt.end())
        printf("Turnausta ei ole\n");
    else
    {
        printf("\n");
        printf("%s\n", ti->nimi);
        printf("ID           : %d\n", ti->id);
        printf("Osallistujia : %d\n", ti->participants);
        printf("Kierroksia   : %d\n", ti->rounds);
        printf("\n");
        system("PAUSE");

        ni = ti->lista.begin();

        plys[0] = 99; // BYE

        // Turnauksen kaikki osallistujat tulostetaan
        for(i = 0; i < ti->participants; i++, ni++)
        {
            printf("Pelaaja %d : %s (%d)\n", i + 1, \
                SearchNameByID(listh, *ni), *ni);

            plys[i + 1] = *ni;
        }

        /*
            Kelataan indeksi i oikeaan kohtaan, jotta voidaan tarkastella
            Kierrostuloksia
        */
```

```
        for(i = 0; i < ti->participants; i++, ni++);

        printf("\n");
        system("PAUSE");

        // Aloitetaan kierrostuloksien raportointi
        for(i = 0; i < ti->rounds; i++)
        {
            printf("\nKierros %d", i + 1);
            for(j = 0; j < (ti->participants + 1) / 2; j++)
            {
                printf("\nPelipari %d : %s - ", j + 1, \
                    SearchNameByID(listh, plys[*ni++]));

                printf("%s \t ", SearchNameByID(listh, plys[*ni++]));

                switch(*ni++)
                {
                    case WIN : printf(" 1 - 0 "); break;
                    case LOSE : printf(" 0 - 1 "); break;
                    case DRAW : printf(" DRAW  "); break;
                    default : printf(" N/A "); break;
                }
                *ni++;
                *ni++;
            }
        }
    }

    return 0;
}

// Tulostetaan kaikki vahvuuslukuhallintaan syötetyt turnaukset
int ShowTournaments(list<turnaus>& listt)
{
    list<turnaus>::iterator ti;

    printf("\n\n");
    printf("Turnauslista : \n");

    for(ti = listt.begin(); ti != listt.end(); ti++)
        printf("%d \t %s \n", ti->id, ti->nimi);

    return 0;
}

/*
    Tätä funktiota käytetään päivittämään turnaus tietueen tuloslistan
    vahvuuslukukentät. Tämän toimenpiteen avulla voidaan välttää pelaajien
    vahvuuksien päivittyminen turnauksen aikana, mikä puolestaan voisi sotkea
    laskentaa.
*/

int UpdateRatingFields(list<henkilo>& listh, list<turnaus>& listt,
                       unsigned int id)
{
    unsigned int i;
    list<henkilo>::iterator hi;
    list<turnaus>::iterator ti;
    list<int>::iterator ni;
    list<int>::iterator nj;

    // Hae ID:n mukainen turnaus
    for(ti = listt.begin(); ti != listt.end(); ti++)
        if(ti->id == id) break;

    // Asetetaan iteraattori ni osoittamaan ensimmäisen pelaajan ID:tä
```

```
    nj = ni = ti->lista.begin();

    /*
        Asetetaan iteraattori nj osoittamaan paikkaa, johon ensimmäisen pelaajan
        vahvuusarvo tallennetaan
    */

    for(i = ti->participants; i--; nj++);

    // Käydään läpi kaikki turnauksen osallistujat
    for(i = 0; i < ti->participants; i++, ni++, nj++)
    {
        // Haetaan pelaaja ID:tä vastaava henkilö
        for(hi = listh.begin(); hi != listh.end(); hi++)
            if(hi->id == *ni) break;

        // Tarkistetaan jäsenyys
        if(hi->rstat == MEMBER || hi->rstat == HONORARY)
        {
            /*
                Tarkistetaan onko pelaajalla provisional- vai
                established-tyyppinen vahvuusluku.

                Mikäli pelaaja kuuluu provisional -laskennan piiriin, tällöin
                turnauksen vahvuuskenttään merkitään arvo 0. Mikäli tälläinen
                pelaaja kuitenkin pelaa toista provisional -laskennan piiriin
                kuuluvaa pelaajaa vastaan, saadaan laskennassa käytettävä
                vahvuus haettua tarkistamalla pelaajan orig kenttä, johon
                on aina tallennettuna alkuvahvuus.
            */

            if(hi->games < 20) *nj = 0;
            else *nj = hi->rating;
        }
    }

    return 0;
}

/*
    Funktio, joka käy läpi kaikki syötetyt turnaukset ja kysyy
    uudet K arvot turnauksille.
*/

int UpdateKValues(list<turnaus>& listt)
{
    int i = 1, K;
    list<turnaus>::iterator ti;

    printf("\n");
    for(ti = listt.begin(); ti != listt.end(); ti++, i++)
    {
        printf("\n\nTurnaus %d : %s", i, ti->nimi);
        printf("\nK arvo      : %d", ti->rating_status);
        printf("\nAnna uusi K : ");
        scanf("%d", &K);
        ti->rating_status = K;
    }

    return 0;
}
```

LIITE 12: rating.h

```
#ifndef RATING_H
#define RATING_H

using namespace std;

unsigned int CalculateProvisional(list<henkilo>& listh, list<turnaus>& listt,
                                  unsigned int id, unsigned int max);

float WinProbability(unsigned int ratingA, unsigned int ratingB);
unsigned int CalculateRatingsByTournament(list<henkilo>& listh,
                                          list<turnaus>& listt);

#endif
```

LIITE 13: rating.cpp

```
#include <cstdlib>
#include <list>
#include "struct.h"
#include "player.h"
#include "tournament.h"

using namespace std;

/*
    Taulukko, jonka avulla saadaan laskettua voittotodennäköisyys ilman
    pyöristysvirheitä.
*/

const unsigned int REXPECT[51] = {4, 11, 18, 26, 33, 40, 47, 54, 62, 69, 77, \
        84, 92, 99, 107, 114, 122, 130, 138, 146, 154, 163, 171, 180, 189, \
        198, 207, 216, 226, 236, 246, 257, 268, 279, 291, 303, 316, 329, 345, \
        358, 375, 392, 412, 433, 457, 485, 518, 560, 620, 736};

enum results {WIN = 1, LOSE = 2, DRAW = 3, OPEN = 4, EMPTY = 5};
enum members {HONORARY = 1, MEMBER = 2, RIF = 3, OTHER = 4};

/*  Laskee voittotodennäköisyyden ja palauttaa sen desimaalilukuna */

float WinProbability(unsigned int ratingA, unsigned int ratingB)
{
    int i;
    int sign = 0;
    int D;           // itseisarvo vahvuuseroista
    float H = 0.50;

    // Tarkistetaan, kumpi vahvuuspistelukema on suurempi: A vai B
    if(ratingA >= ratingB) D = ratingA - ratingB;
    else
    {
        D = ratingB - ratingA;
        sign = 1;
    }


    // Haetaan oikea voittotodennäköisyys taulukosta REXPECT
    for(i = 0; i < 51; i++)
    {
```

```
    // Haetaan oikea voittotodennäköisyys taulukosta REXPECT
    for(i = 0; i < 51; i++)
    {
        if(D < REXPECT[i])
        {
            if(sign) return 1-H;
            else return H;
        }

        H = H + 0.01;
    }

    if(sign) return 0;
    else return 1;

}

/*  Provisional-vahvuuden laskenta */

unsigned int CalculateProvisional(list<henkilo>& listh, list<turnaus>& listt,
unsigned int id, unsigned int max)
{
    unsigned int pelim;
    float score = 0;
    unsigned int rpino = 0;  // Vahvuuskeskiarvolukemaa varten
    int pos, i;
    unsigned int nro;
    unsigned int p1, p2, p3;
    unsigned int temp[50];

    list<turnaus>::iterator ti;
    list<henkilo>::iterator hi;
    list<int>::iterator ni;
    list<int>::iterator nj;

    // Asetetaan iteraattori osoittamaan oikeaa henkiloa
    for(hi = listh.begin(); hi != listh.end(); hi++)
        if(hi->id == id) break;

    if(hi == listh.end())
    {
        printf("Virhe, ID ei ole oikein");
        return 0;
    }

    // Rating pool vaikutus
    pelim = hi->orig[1];
    if(pelim) for(i = 0; i < pelim; i++)
    {
        rpino += hi->orig[0];
        score += 0.5;
    }

    // Mennään turnaukset läpi yksi kerrallaan
    for(ti = listt.begin(); ti != listt.end(); ti++)
    {
        pos = ti->participants;
        nro = 0; // Pelaajan numero tuloslistassa

        // Etsitään pelaaja tuloslistasta
        for(ni = ti->lista.begin(); pos > 0; pos--, ni++, nro++)
        {
            // Jos pelaajan havaitaan osallistuvan
            if(*ni == id)
            {
                pos = 2 * ti->participants;
                i = 0;
                for(nj = ti->lista.begin(); pos; pos--, nj++, i++)
```

```
                    temp[i] = *nj;  // Asetetaan iteraattori kierroksen alkuun

                // Jatketaan etsien tuloksia
                for(;nj != ti->lista.end();)
                {
                    p1 = *nj++;
                    p2 = *nj++;
                    p3 = *nj++;
                    *nj++;
                    *nj++;

                    // Tarkistetaan vastustajan vahvuusluokitus
                    if(p1 == nro + 1 &&
                        SearchRatingStatusByID(listh, temp[p2 - 1]) !=
                        OTHER && p2)
                    {
                        // Lisätään vastustajan vahvuuspistelukema rpino:on
                        rpino += temp[p2 - 1 + ti->participants];
                        pelim++;

                        if(p3 == WIN) score++;
                        else if(p3 == DRAW) score += 0.5;
                    }

                    /*
                        Sama vastustajan vahvuusluokitus tarkistus kuin
                        aiemmin, mutta tuloksen vaikutus käänteinen.
                    */

                    if(p2 == nro + 1 &&
                        SearchRatingStatusByID(listh, temp[p1 - 1]) !=
                        OTHER && p1)
                    {
                        rpino += temp[p1 - 1 + ti->participants];
                        pelim++;

                        if(p3 == LOSE) score++;
                        else if(p3 == DRAW) score += 0.5;
                    }
                }
            }
        }

        /*
            Provisional-vahvuuslaskenta saadaan päätökseen vain,
            jos kokonaispelimäärä on vähintään 20
        */

        if(pelim >= 20)
        {
            hi->games = pelim;
            hi->rating = (int) ((float) hi->orig[0] + \
                        ((float) rpino / (float) pelim) * \
                        (1 - (float) score / (float) pelim));

            return hi->rating;
        }

        /*
            Jos parametrina annettu max turnaus id tulee vastaan,
            ei anneta jatkaa laskentaa pisemmälle.
        */

        if(ti->id == max) return 0;
    }

    return 0;
}
```

```
/* Funktio, joka laskee vahvuusluvut ensimmäisestä turnauksesta järjestäin */

unsigned int CalculateRatingsByTournament(list<henkilo>& listh,
                                          list<turnaus>& listt)
{
    unsigned int i, j, K, rstat, temp;
    list<turnaus>::iterator ti;
    list<henkilo>::iterator hi;
    list<int>::iterator ni; // pelaaja alkion iteraattori tuloslistassa
    list<int>::iterator nj; // pelaaja alkion iteraattori tuloslistassa
    list<int>::iterator nk; // kierrostulos iteraattori tuloslistassa
    list<int>::iterator nt; // vastustajan alkio tuloslistassa
    list<int>::iterator nu;

    // Siivotaan vanhat vahvuusmerkinnät pois, sillä nyt lasketaan uudestaan
    CleanMemberRatings(listh);

    // Tarkistetaan onko turnauksia listassa
    if(listt.empty())
    {
        printf("\nEi turnauksia listassa!\n");
        return 0;
    }

    ti = listt.begin();

    // Ensimmäisestä turnauksesta viimeiseen, yksi kerrallaan
    for(; ti != listt.end(); ti++)
    {
        // Päivitetään kyseisen turnauksen vahvuuskentät Crosstable listassa
        UpdateRatingFields(listh, listt, ti->id);

        // Haetaan turnauksen ensimmäisen pelaajan ID
        nj = ni = ti->lista.begin();

        /*
            Asetetaan iteraattori nj osoittamaan paikkaa, johon ensimmäisen
            pelaajan vahvuusarvo on tallennettuna.
        */

        for(temp = ti->participants; temp--; nj++);

        // Käydään kaikki turnauksen pelaajat läpi yksitellen
        for(i = 0; i < ti->participants; i++)
        {
            // Haetaan pelaajan ID:tä vastaava henkilö
            for(hi = listh.begin(); hi != listh.end(); hi++)
                if(hi->id == *ni) break;

            // Tarkistetaan jäsenyys (vahvuusluku lasketaan vain jäsenille)
            if(hi->rstat == MEMBER || hi->rstat == HONORARY)
            {
                // Tarkistetaan, kuuluuko pelaaja provisional laskennan piiriin
                if(*nj == 0)
                {
                    // Yritetään laskea provisional vahvuusluku
                    CalculateProvisional(listh, listt, *ni, ti->id);
                }
                else

                /*
                    Jos pelaaja ei kuulu provisional laskennan piiriin, hän
                    kuuluu established laskennan piiriin
                */
                {
                    // Asetetaan iteraattori nk ensimmäisen kierroksen alkuun
                    temp = 2 * ti->participants;
```

```
                    for(nk = ti->lista.begin(); temp--; nk++);

                    for(;nk != ti->lista.end();)
                    {
                        K = 1;

                        /*
                            Tarkistetaan, onko peliparin ensimmäinen pelaaja se,
                            jolle vahvuuslukua ollaan päivittämässä
                        */

                        if(*nk == i + 1)
                        {
                            // Jos epätosi niin vastus on BYE
                            nk++;

                            if(*nk)
                            {
                                j = *nk - 1;

                                // Asetetaan iteraattori vastustajan ID:hen
                                for(nt = ti->lista.begin(); j--; nt++);

                                 // Haetaan vastustajan jäsenyysluokitus
                                rstat = SearchRatingStatusByID(listh, *nt);

                                /*
                                    Mikäli jäsenyysluokitus on mikä tahansa muu
                                    kuin OTHER - tyyppinen, voidaan laskentaa
                                    jatkaa.
                                */

                                if(rstat != OTHER)
                                {
                                    if(rstat == RIF)
                                    {
                                        K >>= 1;
                                    }

                                    /*
                                        Asetetaan iteraattori nu osoittamaan
                                        vastustajan vahvuuslukua.
                                    */

                                    temp = ti->participants;

                                    for(nu = nt; temp--; nu++);

                                    /*
                                        Vahvuus voidaan laskea vain jos
                                        vastustajan vahvuusluku on tyyppiä
                                        ESTABLISHED.
                                    */

                                    if(*nu)
                                    {
                                        temp = hi->rating + (ti->rating_status \
                                               / K) * (*(++nk) - \
                                               WinProbability(hi->rating, *nu));

                                        *(++nk) = temp - hi->rating;
                                        hi->rating = temp;
                                        *nk--;
                                        *nk--;
                                    }
                                }
                            }
                            *nk--;
```

```
                        }

                        /*
                            Siirrytään seuraavaan pelaajaan ja tarkistetaan,
                            onko peliparin toinen pelaaja se, jolle
                            vahvuuslukua ollaan päivittämässä.
                        */

                        else
                        {
                            *nk++;
                            if(*nk == i + 1)
                            {
                                nk--;

                                // Jos epätosi, vastus on BYE
                                if(*nk)
                                {
                                    j = *nk - 1;

                                    // Asetetaan iteraattori vastustajan ID:hen
                                    for(nt = ti->lista.begin(); j--; nt++);

                                     // Haetaan vastustajan jäsenyysluokitus
                                    rstat = SearchRatingStatusByID(listh, *nt);

                                    /*
                                        Mikäli jäsenyysluokitus on mikä tahansa
                                        muu kuin OTHER - tyyppinen, voidaan
                                        laskentaa jatkaa.
                                    */

                                    if(rstat != OTHER)
                                    {
                                        if(rstat == RIF)
                                        {
                                            K >>= 1;
                                        }

                                        /*
                                            Asetetaan iteraattori nu osoittamaan
                                            vastustajan vahvuuslukua.
                                        */

                                        temp = ti->participants;

                                        for(nu = nt; temp--; nu++);

                                        /*
                                            Vahvuus voidaan laskea vain jos
                                            vastustajan vahvuusluku on tyyppiä
                                            ESTABLISHED.
                                        */

                                        if(*nu)
                                        {
                                            nk++;
                                            temp = hi->rating + \
                                               (ti->rating_status / K) * \
                                               (*(++nk) - \
                                               WinProbability(hi->rating, *nu));

                                            nk++;
                                            *(++nk) = temp - hi->rating;
                                            hi->rating = temp;
                                            *nk--;
                                            *nk--;
                                            *nk--;
```

```
                                        *nk--;
                                    }
                                }
                            }
                            nk++;
                        }
                        nk--;
                    }

                    // Siirrytään seuraavaan peliparii
                    *nk++;
                    *nk++;
                    *nk++;
                    *nk++;
                    *nk++;
                }
            }
        }
        // Seuraava pelaaja
        *ni++;
        *nj++;
    }
}

    return 0;
}
```

LIITE 14: file.h

```
#ifndef FILE_H
#define FILE_H

using namespace std;

int OpenAll(list<henkilo>& listh, list<turnaus>& listt);
int SaveAll(list<henkilo>& listh, list<turnaus>& listt);
void WriteParticipants(list<int>& lista, list<turnaus>& listt,
                       list<henkilo>& listh, unsigned int n);

#endif
```

LIITE 15: file.cpp

```
#include <cstdlib>
#include <list>
#include "struct.h"
#include "player.h"

using namespace std;

/*
    MAXSIZE arvoa käytetään estämään liian suurien tai virhetilanteista
    johtuvien tiedostoon kirjoitusten tapahtuminen.
*/

#define MAXSIZE 5000

/* Tiedostosta avaamisen hoitava funktio */

int OpenAll(list<henkilo>& listh, list<turnaus>& listt)
{
    FILE * pF;
    int i, j, dummy;
    unsigned int buf[1];

    henkilo htemp;
    turnaus ttemp;
    int ltemp;

    int headeri[MAXSIZE + 6 + 2];
    memset(headeri, 0, sizeof(headeri));

    list<henkilo>::iterator hi;
    list<turnaus>::iterator ti;
    list<int>::iterator ni;

    // Tyhjennetään listat
    listh.clear();
    listt.clear();

    // Tietokantana käytetään tiedostoa data.bin
    pF = fopen("data.bin", "rb");

    if(pF != NULL)
    {
        for(i = 0; !feof(pF) && i < MAXSIZE + 6 + 2; i++)
        {
            fscanf(pF, "%c", &dummy);
            headeri[i] |= dummy;

            if(!feof(pF)) fscanf(pF, "%c", &dummy);

            headeri[i] |= dummy << 8;

            if(!feof(pF)) fscanf(pF, "%c", &dummy);

            headeri[i] |= dummy << 16;

            if(!feof(pF)) fscanf(pF, "%c", &dummy);

            headeri[i] |= dummy << 24;

            if(headeri[i] == 0xefefefef) break;
        }
```

```
        // Luodaan henkilo -lista
        for(i = 0; i < headeri[4]; i++)
        {
            fread(&(htemp.id), sizeof(henkilo), 1, pF);

            listh.push_back(htemp);
        }

        // Luodaan turnaus -lista
        for(i = 0; i < headeri[5]; i++)
        {
            fread(&(ttemp.id), sizeof(turnaus) - sizeof(unsigned int) \
                                - sizeof(list<unsigned int>), 1, pF);

            ttemp.lista.clear();

            for(j = 0; j < headeri[6 + i]; j++)
            {
                dummy = 0;
                ltemp = 0;

                fread(&buf[0], sizeof(unsigned int), 1, pF);

                ttemp.lista.push_back(buf[0]);
            }

            listt.push_back(ttemp);
        }

        fclose(pF);
    }
    else exit(0); // virhe

    return 0;
}

/*
    SaveAll funktio tallettaa kaiken olleelisen informaation
    tiedostoon data.bin. Alussa on headeri ja se sisältää seuraavat
    tiedot. Tiedoston alusta:
        xx xx : tietueen henkilo koko
        xx xx : tietueen turnaus koko
        xx xx : henkilo listan listh koko
        xx xx : turnaus listan listt koko
        xx xx : elementtien lukumäärä listassa listh
        xx xx : elementtien lukumäärä listassa listt (=N)
        N x (xx xx) : Elementtien lukumäärä crosstable listassa, joka on osana
                      turnaus listaa listt.

        ef ef ef ef : päättää headerin
*/

int SaveAll(list<henkilo>& listh, list<turnaus>& listt)
{

    FILE * pF;
    int i;
    unsigned int buf[1];

    int headeri[MAXSIZE + 6 + 2];

    list<henkilo>::iterator hi;
    list<turnaus>::iterator ti;
    list<int>::iterator ni;

    // Täytetään headerin 6 ensimmäistä kenttää
    headeri[0] = sizeof(henkilo);
    headeri[1] = sizeof(turnaus);
```

```
    headeri[2] = sizeof(listh);
    headeri[3] = sizeof(listt);
    headeri[4] = listh.size();
    headeri[5] = listt.size();

    // Täytetään headerin variable length -osio
    for(i = 6, ti = listt.begin(); ti != listt.end(); i++, ti++)
        headeri[i] = ti->lista.size();

    // Headeri päättyy
    headeri[i++] = 0xefefefef;

    pF = fopen("data.bin", "w+b");
    if(pF != NULL)
    {
        fwrite(headeri, sizeof(int), i, pF);

        // Ensin tallennetaan henkilo-listan sisältö
        for(hi = listh.begin(); hi != listh.end(); hi++)
            fwrite(&(hi->id), sizeof(henkilo), 1, pF);

        // Sitten tallennetaan turnaus-listan sisältö
        for(ti = listt.begin(); ti != listt.end(); ti++)
        {
            fwrite(&(ti->id), sizeof(turnaus) - sizeof(unsigned int) \
                            - sizeof(list<unsigned int>), 1, pF);

            // turnaus-lista pitää sisällään tuloslistan, joka tallennetaan nyt
            for(ni = ti->lista.begin(); ni != ti->lista.end(); ni++)
            {
                buf[0] = *ni;
                fwrite(&buf[0], sizeof(unsigned int), 1, pF);
            }
        }

        fclose(pF);
    }
    else exit(0); // virhe

    printf("\nHeaderi : ");
    for(i = 0; headeri[i] != 0xefefefef && i < MAXSIZE + 6 + 2; i++)
        printf(" %d ", headeri[i]);

    return 0;
}

/* Turnauksen osallistujalistan tulostus tiedostoon tekstimuotoisena */

void WriteParticipants(list<int>& lista, list<turnaus>& listt,
                       list<henkilo>& listh, unsigned int n)
{
    FILE * pF;
    unsigned int lkm;
    list<int>::iterator ni;
    unsigned int temp[100];
    unsigned int tp = 0;

    lkm = n;

    // Tallennetaan tiedostoon players.txt
    pF = fopen("players.txt", "w+");
    if(pF != NULL)
    {
        // Käydään tuloslistasta läpi osallistujien ID:t
        for(ni = lista.begin(); ni != lista.end() && lkm; lkm--, tp++)
            temp[tp] = *ni++;

        lkm = n;
```

```
        for(ni = lista.begin(); ni != lista.end() && lkm; lkm--, tp++)
            temp[tp] = *ni++;

        lkm = n;

        // Etsitään ID:eitä vastaavat henkilot ja tallennetaan tiedostoon
        for(tp = 0; lkm; lkm--)
        {
            fprintf(pF, "%d - %s (%d) \t %d \n", (tp + 1) / 2 + 1, \
                SearchNameByID(listh, temp[tp]), temp[tp + 1]);

            tp += 2;
        }

        fclose(pF);
    }
    else exit(0);
}
```