Tampereen teknillinen yliopisto
Ohjelmistotekniikan laitos

# OHJ-1151
# Ohjelmointi IIe

Harjoitustyö
Hervannan Hakkerit
Loppudokumentti

21.5.2008

Petteri Aimonen (205441)
aimonen
petteri.aimonen@tut.fi

# 1 Ohjelman rakenne

## 1.1 Ohjelman tietorakenteet

Pelitilanne sisältyy Pelitilanne-structiin, jossa on tallennettuna kaikki peliin liittyvä tilatieto.

```
struct Asetukset {
    Asetukset();

    // Yleisasetustiedoston tiedot.
    int koodauskulutus;
    int reitityskulutus;
    int puskurointikulutus;
    int tankkauskulutus;
    int ryhmanvarastot;
    int kolauusiutuminen;
};

struct Hakkeriryhma {
    Hakkeriryhma(const std::string &ryhmannimi);

    std::string nimi; // Hakkeriryhmän nimi.
    std::vector<Hakkeri *> hakkerit; // Osoittimet ryhmän hakkereihin.
    int vuoro; // Vuorossa olevan hakkerin indeksi.
};

struct Pelitilanne {
    Pelitilanne();

    Asetukset asetukset; // Pelin yleisasetukset.
    std::vector<Hakkeriryhma> ryhmat; // Peliin kuuluvat hakkeriryhmät.
    int vuoro; // Vuorossa olevan ryhmän indeksi.

    bool lopetettu; // True, jos pelissä ei enää suoriteta vuoroja.
};
```

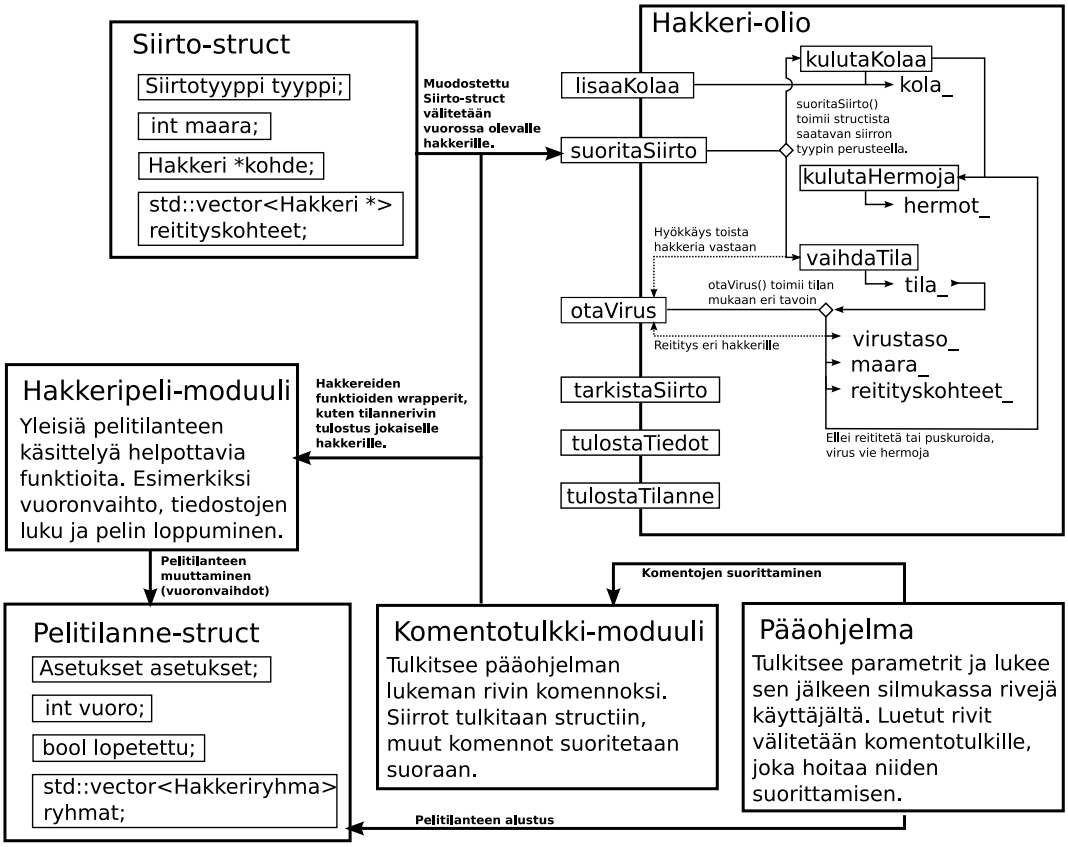
Pelin siirrot tallennetaan structiin, jossa on siirron tyyppi enumina ja siirtoon liittyvien hakkerien tunnisteet. Structin kenttien käyttö riippuu siirtotyypistä. Tämä struct voidaan välittää parametrina mm. siirron oikeellisuuden tarkistusta varten.

```
enum Siirtotyyppi {EISIIRTOA, KOODAA, PUSKUROI, REITITA, RESETOI, HYOKKAA,
                   TANKKAA, ODOTA};

struct Siirto {
    Siirto(Siirtotyyppi ntyyppi);

    Siirtotyyppi tyyppi; // Siirron tyyppi - määrää muiden kenttien käytön
    int maara; // KOODAA: maara on koodattavan viruksen taso.
        // TANKKAA: tankattavan kolan määrä.
        // PUSKUROI, REITITA: maara on puskurointien/reititysten määrä.
        // RESETOI, HYOKKAA, ODOTA: kenttä ei ole käytössä.
```

```
    Hakkeri *kohde; // Vain HYOKKAA-komennossa, jossa osoitin hyökkäyksen
                    // kohteena olevaan hakkeriin

    std::vector<Hakkeri *> reitityskohteet; // Vain REITITA-komennossa,
                // jossa vektori osoittimista annettuihin reitityskohteisiin.
};
```

## 1.2 Ohjelman suoritus

Pääohjelma alustaa ensiksi pelitilanteen ja lukee sitten käyttäjältä rivejä, jotka välitetään komentotulkille. Komentotulkki suorittaa komennot ja välittää siirrot Hakkeri-olioiden suoritettavaksi. Pääohjelman silmukka loppuu, kun siirron jälkeen havaitaan pelin loppuneen. Lopuksi tulostetaan mahdollinen voittaja, ja lopputilanne.




Kuva 1: Ohjelman suoritus

# 2 Moduulit

## 2.1 Pääohjelma

### 2.1.1 Apufunktio: tulkitseParametrit

Tulkitsee komentoriviparametrit ja toteuttaa ne. Voi tuottaa poikkeukset Parametrivirhe, Avausvirhe ja Lukuvirhe.

*Esittely:*
`bool tulkitseParametrit(int argc, char *argv[], Pelitilanne &pelitilanne);`

*Parametrit:*
Parametri argc ja argv ovat pääohjelman saamat parametrit.
Parametri pelitilanne on struct, johon luetut asetukset ja ryhmät tehdään.

*Paluuarvo:*
Palauttaa true jos peli alkaa normaalisti. Palauttaa false, jos parametrina oli −−help, eli pelin ei pidä alkaa.

*Toiminta:*
Käydään parametrit järjestyksessä läpi. Jos parametri on -a, -1 tai -2, seuraava on sille annettu tiedostonimi. Tiedostonimet tallennetaan väliaikaismuuttujiin.

Jos parametri on -h tai −−help, tulostetaan ohjeteksti, asetetaan lopetusarvo ja palautetaan false.

Virheitä havaittaessa tuotetaan poikkeus ja vapautetaan jo luodut hakkerit.

*Käyttö:*
Pääohjelma kutsuu tätä funktiota.

### 2.1.2 Funktio: main

*Toiminta:*
Pelitilanne on mainissa paikallisena muuttujana, joka alustetaan apufunktiolla tulkitseParametrit. Jos se tuottaa poikkeuksen, tulostetaan vastaava virheilmoitus ja lopetetaan suoritus paluuarvolla **EXIT_FAILURE**. Jos poikkeusta ei tule, mutta paluuarvo on false, on kyseessä ohjetekstin tulostus ja lopetetaan paluuarvolla **EXIT_SUCCESS**.

Muutoin siirrytään pääsilmukkaan. Tulostetaan komentotulkin kautta haettu kehote ja luetaan `std::getline`:lla yksi rivi. Mikäli rivi alkaa risuaidalla, luetaan uusi rivi. Välitetään luettu rivi komentotulkin toteutaRivi-funktiolle. Silmukka päättyy kun vuoron suorittamisen seurauksena *pelitilanne.lopetettu* on true tai rivin lukeminen epäonnistuu.

Silmukan jälkeen tulostetaan lopputilanne ja lopetetaan paluuarvolla EXIT_SUCCESS.

## 2.2 Moduuli: Komentotulkki

Komentotulkki hoitaa komentojen tulkitsemisen ja osittain suorittamisen.

Tässä moduulissa on määritelty tietotyypit ja olemassaolevat komennot:

```
enum Toiminto {EITOIMINTOA, SIIRTO, TILANNE, TIEDOT, OHJE, LOPETUS};

struct Komento {
    std::string komento;
    Toiminto toiminto;
    SiirtoTyyppi siirtotyyppi; // Vain jos toiminto == SIIRTO
```

```
    int parametreja_enintaan; // Jos == -1, tarkistus ei käytössä
    int parametreja_vahintaan; // Jos == -1, tarkistus ei käytössä
};

const Komento KOMENNOT[] = {{"LOPETA", LOPETUS, 0, 0, 0},
                            {"HYOKKAA", SIIRTO, HYOKKAA, 1, 1},
                            ...
                            };
```

### 2.2.1 Funktio: toteutaRivi

Toteuttaa rivin tai tulostaa virheen.

*Esittely:*
`bool toteutaRivi(const std::string &rivi, Pelitilanne &pelitilanne, std::ostream &out, std::ostream &err);`

*Parametrit:*
Parametri **rivi** on käyttäjältä luettu kokonainen rivi.
Parametri **pelitilanne** on struct, johon komennot suoritetaan.
Parametri **out** on virta, johon pelin normaalit tulosteet tulostetaan.
Parametri **err** on virta, johon virheilmoitukset tulostetaan.

*Paluuarvo:*
False, jos komennon suorituksessa oli virhe.

*Toiminta:*
Rivin tyhjällä tilalla erotellut osat luetaan *merkkijonokasittely*-moduulin lueOsat-funktiolla vektoriin. Jos vektori on tyhjä (tarkoittaa että rivillä on vain tyhjää tilaa) palautetaan **false** tulostamatta mitään. Muutoin ensimmäinen osa eli komentosana välitetään etsiKomento-funktiolle, joka etsii vakiotaulukosta vastaavan Komento-structin.

Jos tämä palauttaa **false**, tulostetaan "Virhe: Tuntematon komento." virhevirran kautta ja palautetaan **false**.

Loppujen osien määrää verrataan komennon `parametreja_enintaan`- ja `parametreja_vahintaan`-kenttiin. Virheistä tulostetaan joko "Virhe: Liikaa parametreja." tai "Virhe: Liian vähän parametreja." ja palautetaan **false**.

Muutoin komento ja parametrit välitetään toteutaKomento-funktiolle.

*Käyttö:*
Pääohjelma kutsuu tätä luettuaan käyttäjältä rivin.

### 2.2.2 Funktio: etsiKomento

Tulkitsee komennon etsimällä vastaavan komennon vakiotaulukosta. Palauttaa false jos vastaavuuksia ei ole tai on liikaa.

*Esittely:*
`bool etsiKomento(const std::string &annettu, Komento &tulos);`

*Parametrit:*
Parametri **annettu** on käyttäjän antama komentosana.
Parametri **tulos** on viite, jonka kautta löydetty structi tallennetaan.

*Paluuarvo:*
True, jos löytyi täsmälleen yksi vastaavuus.

*Toiminta:*
Käydään KOMENNOT-vakiotaulukko järjestyksessä läpi. Komentoja verrataan merkkijonomoduulin funktiolla vertaaLyhennos ja lasketaan vastaavuudet. Jos vastaavuuksia on tasan yksi, tallennetaan se ja palautetaan **true**. Muutoin palautetaan **false**.

*Käyttö:*
Funktio toteutaRivi tulkitsee tämän funktion avulla käyttäjän antaman komennon.

### 2.2.3 Funktio: toteutaKomento

Tulkitsee parametrit ja toteuttaa komennon. Virheen sattuessa palauttaa false ja tulostaa virheen.

*Esittely:*
```
bool toteutaKomento(const Komento &komento,
const std::vector<std::string> &parametrit, Pelitilanne &pelitilanne,
std::ostream &out, std::ostream &err);
```

*Parametrit:*
Parametri **komento** on tulkittu Komento-struct.
Parametri **parametrit** on vektori annetuista parametreista.
Parametri **pelitilanne** on struct, johon komento toteutetaan.
Parametri **out** on tulostevirta, ja parametri **err** on virhevirta.

*Paluuarvo:*
True, jos komennossa ei ollut virheitä.

*Toiminta:*
Jos komennon tyyppi on SIIRTO, tulkitaan parametrit tulkitseSiirto-funktiolla. Se voi tuottaa Hakkerivirhe- ja Lukuvirhe-poikkeukset komennon virheistä. Ne kaapataan myöhemmin tässä funktiossa. Tulkittu siirto tarkistetaan vuorossa olevan hakkerin tarkistaSiirto-jäsenfunktiolla, joka hoitaa myös virheilmoitusten tulostamisen. Jos virheitä ei ollut, kutsutaan *hakkeripeli*-moduulin suoritaSiirto-funktiota, joka hoitaa myös vuoron vaihtamisen.

Muut komentotyypit suoritetaan vastaavasti, lähinnä *hakkeripeli*-moduulin funktioita hyödyntäen tai suoraan tietoja tulostaen ja muuttaen.

Mahdolliset Lukuvirheet ja Hakkerivirheet kaapataan, ja tulostetaan niistä virheilmoitukset. Hakkerivirhe ilmaisee olematonta hakkeria, ja Lukuvirhe parametria, joka ei ollut positiivinen kokonaisluku.

*Käyttö:*
Funktio toteutaRivi käyttää tätä funktiota.

### 2.2.4 Funktio: tulkitseSiirto

Tulkitsee siirroksi tiedetyn komennon parametrit Siirto-rakenteeseen. Tuottaa poikkeuksen Lukuvirhe tai Hakkerivirhe jos parametreissa on virheitä.

*Esittely:*
```
Siirto tulkitseSiirto(const Komento &komento,
const std::vector<std::string> &parametrit, const Pelitilanne &pelitilanne);
```

*Parametrit:*
Parametri **komento** on tulkittu Komento-struct.
Parametri **parametrit** on vektori annetuista parametreista
Parametri **pelitilanne** on struct, josta haetaan osoittimet siirron kohteita varten.

*Paluuarvo:*
Paluuarvo on Siirto-struct, johon tulkitut tiedot on tallennettu. Mahdolliset virheet hoidetaan poikkeuksin.

*Toiminta:*
Komento-structista saadaan tieto siirron tyypistä. Parametrit tulkitaan hyödyntäen kokonaisluvuksi- ja etsiHakkeri_P-funktioita. Jälkimmäinen on apufunktio, joka tuottaa poikkeuksen hakkerin puuttuessa. Normaali etsiHakkeri ei tätä tee, jotta sillä voi tehokkaasti myös tarkistaa että nimi on vapaana.

*Käyttö:*
Funktio toteutaKomento kutsuu tätä funktiota parametrien tulkitsemiseksi.

## 2.3 Moduuli: Merkkijonokäsittely

Tämä moduuli sisältää pieniä merkkijonojen käsittelyyn käytettyjä apufunktioita:

```
// Lukee kaikki välilyönnillä erotellut osat virrasta vektoriin
void lueOsat(std::istream &virta, std::vector<std::string> &tulos);

// Muuntaa merkkijonon positiiviseksi kokonaisluvuksi.
// Tuottaa poikkeuksen Lukuvirhe jos merkkijonossa on virheitä.
// Poikkeuksen parametrina on annettu merkkijono.
int kokonaisluvuksi(std::string merkkijono);

// Vertaa merkkijonoja kirjainkokoriippumattomasti.
bool vertaaKKR(const std::string &eka, const std::string &toka);

// Vertaa merkkijonoja kirjainkokoriippumattomasti siten,
// että riittää kun annettu on kokonaisen alku.
bool vertaaLyhennos(const std::string &annettu,
                    const std::string &kokonainen);
```

## 2.4 Moduuli: Poikkeukset

Tämä moduuli sisältää pelin käyttämät poikkeusluokat:

- **Parametrivirhe:** Ohjelmaa käynnistettäessä on annettu virheellinen komentoriviparametri. Käytössä pääohjelmassa.
- **Avausvirhe:** Asetustiedostoa ei voitu avata. Käytössä hakkeripeli-moduulissa, kaapataan pääohjelmassa. Parametrina tiedoston nimi.
- **Lukuvirhe:** Lukemisessa tapahtunut virhe, asetustiedostoissa tai komentotulkin komentojen parametreissa. Parametri on virheeseen liittyvä merkkijono, kuten tiedoston nimi tai virheellinen komennon parametri.
- **Hakkerivirhe:** Komentotulkille annettua hakkerin nimeä ei löydy. Parametrina annettu nimi.

## 2.5 Moduuli: Hakkeripeli

Sisältää pelitilanteen hallintaan liittyviä funktioita.

### 2.5.1 Funktio: lueYleisasetukset

Yleisasetustiedoston lukeminen. Virheen sattuessa tuottaa poikkeuksen Lukuvirhe. Funktiosta on myös ylikuormitettu versio, joka ottaa hoitaa tiedoston avauksen sekä lisää tiedostonimen tuotettuihin poikkeuksiin.

*Esittely:*
`void lueYleisasetukset(std::istream &virta, Asetukset &tulos);`

*Parametrit:*
Parametri **virta** on virta, josta asetustiedosto luetaan.
Parametri **tulos** on struct, johon yleisasetukset luetaan.

*Paluuarvo:*
Ei paluuarvoa.

*Toiminta:*
Syötevirrasta luetaan kaikki osat lueOsat-funktiolla. Jos osia ei ole tasan 6, tuotetaan Lukuvirhe-poikkeus. Osat muunnetaan kokonaisluvuiksi kokonaisluvuksi-funktiolla, joka myös tuottaa tarvittaessa virheen.

Osat tallennetaan järjestyksessä tulos-structin kenttiin *koodauskulutus*, *reitityskulutus*, *puskurointikulutus*, *tankkauskulutus*, *ryhmanvarastot* ja *kolauusiutuminen*.

*Käyttö:*
Pääohjelman tulkitseParametrit-funktio kutsuu tätä funktiota asetustiedoston lukemiseksi.

### 2.5.2 Funktio: lueHakkeriryhma

Hakkeriryhmän tiedoston lukeminen ja Hakkeri-olioiden luominen. Virheen sattuessa tuottaa poikkeuksen Lukuvirhe. Funktiosta on myös ylikuormitettu versio, joka ottaa hoitaa tiedoston avauksen sekä lisää tiedostonimen tuotettuihin poikkeuksiin.

*Esittely:*
`void lueHakkeriryhma(std::istream &virta, Pelitilanne &pelitilanne);`

*Parametrit:*
Parametri **virta** on virta, josta hakkeriryhmätiedosto luetaan.
Parametri **pelitilanne** on struct, johon luetut hakkeriryhmät tallennetaan.

*Paluuarvo:*
Ei paluuarvoa.

*Toiminta:*
Aluksi luetaan *std::getline*:llä ryhmän nimi. Loput rivit käydään järjestyksessä läpi ja jaetaan osiin. Osista tulkitaan hakkerin nimi sekä hermo- ja kolavarannot, jotka välitetään Hakkeriluokan rakentajalle.

Osoittimet dynaamisesti luotuihin hakkereihin tallennetaan Hakkeriryhma-structin sisällä olevaan vektoriin.

Lisäksi suoritetaan muita tarkistuksia, kuten nimien yksiselitteisyys sekä osuuksien summa. Näistäkin tuotetaan poikkeus Lukuvirhe. Poikkeuksen sattuessa luodut hakkerit vapautetaan eikä ryhmää lisätä tilanteeseen.

*Käyttö:*
Pääohjelman tulkitseParametrit-funktio käyttää tätä hakkeriryhmien lukemiseen.

### 2.5.3 Funktio: etsiHakkeri

Etsii hakkerin nimeä annetuista ryhmistä. Palauttaa osoittimen tai NULL.

*Esittely:*

```
Hakkeri *etsiHakkeri(const std::string &nimi,
const std::vector<Hakkeriryhma> &ryhmat);
```

*Parametrit:*
Parametri **nimi** on etsitty hakkerin nimi, joka on kirjainkokoriippumaton.
Parametri **ryhmat** on Hakkeriryhmat-structeja sisältävä vektori, josta hakkeria etsitään.

*Paluuarvo:*
Paluuarvo on osoitin etsittyyn Hakkeri-olioon, tai NULL jos hakkeria ei löytynyt.

*Toiminta:*
Ryhmien hakkerit käydään läpi ja nimiä verrataan vertaaKKR-funktiolla. Löydetty hakkeri palautetaan heti. Silmukan loputtua palautetaan **NULL**.

*Käyttö:*
Hakkeriryhmien lukeminen tarkistaa tällä ettei nimi ole vielä käytössä. Komentotulkki käyttää tätä komentojen tulkitsemisessa.

### 2.5.4 Funktio: haeVuoro

Palauttaa osoittimen vuorossa olevaan hakkeriin.

*Esittely:*

```
Hakkeri *haeVuoro(const Pelitilanne &pelitilanne);
```

*Parametrit:*
Parametri **pelitilanne** on struct, josta vuoro haetaan.

*Paluuarvo:*
Paluuarvo on osoitin vuorossa olevaan Hakkeri-olioon.

*Toiminta:*
Palautetaan suoraan pelitilaneen ja hakkeriryhmän vuoro-indeksien perusteella haettu osoitin.

*Käyttö:*
Tätä funktiota käytetään eri osista ohjelmaa, eniten komentotulkista ja hakkeripelimoduulista.

### 2.5.5 Funktio: laskeKulutus

Laskee siirron kolakulutuksen.

*Esittely:*

```
int laskeKulutus(const Siirto &siirto, const Asetukset &asetukset) ;
```

*Parametrit:*
Parametri **siirto** on Siirto-struct, jonka sisältämän siirron kulutus lasketaan.
Parametri **asetukset** on Yleisasetukset-struct, josta haetaan kulutustiedot.

*Paluuarvo:*
Paluuarvo on kokonaisluku, joka ilmaisee kulutetun kolan määrän.

*Toiminta:*
Siirron tyypin ja määrän perusteella palautetaan kulutus. Jos siirto ei kuluta kolaa, palautetaan **0**.

*Käyttö:*
Hakkeri-oliot käyttävät tätä suorittaessaan siirtoa.

### 2.5.6 Funktio: suoritaSiirto

Suorittaa siirron vuorossa olevalla hakkerilla, tulostaa ilmoitukset ja vaihtaa tarvittaessa vuoron.

*Esittely:*
```
void suoritaSiirto(const Siirto &siirto, Pelitilanne &pelitilanne,
std::ostream &out);
```

*Parametrit:*
Parametri **siirto** on suoritettava siirto.
Parametri **pelitilanne** on struct, johon siirto suoritetaan.
Parametri **out** on tulostevirta.

*Paluuarvo:*
Ei paluuarvoa.

*Toiminta:*
Vuorossa olevan hakkerin suoritaSiirto-jäsenfunktiota kutsutaan. Se palauttaa tiedon vuoron vaihtumisesta.

Jos vuoro vaihtuu, etsitään seuraavaksi vuoroon tuleva hakkeri, lisätään sille kolaa ja tulostetaan ilmoitus vuorossa olevasta hakkerista.

Pelin loppuminen tarkistetaan peliLoppu-funktiolla. Pelin loppuessa asetetaan pelitilanne.lopetettu trueksi, jolloin pääohjelman silmukka päättyy.

*Käyttö:*
Komentotulkki kutsuu tätä siirron suorittamiseksi.

### 2.5.7 Funktio: ryhmaMukana

Tarkistaa, että ryhmä on yhä mukana pelissä.

*Esittely:*
```
bool ryhmaMukana(const Hakkeriryhma &ryhma);
```

*Parametrit:*
Parametri **ryhma** on hakkeriryhmä, jonka aktiivisuus tarkistetaan.

*Paluuarvo:*
True, jos ryhmässä on ainakin yksi aktiivinen hakkeri.

*Toiminta:*
Ryhmän hakkerit käydään läpi, ja tarkistetaan niiden aktiivisuus online-jäsenfunktiolla. Aktiivisen hakkerin löydyttyä palautetaan **true**, muutoin silmukan loputtua **false**.

*Käyttö:*
Funktio peliLoppu käyttää tätä ryhmien tarkistamiseen ja pääohjelma voittajan toteamiseen.

### 2.5.8 Funktio: peliLoppu

Tarkistaa, onko peli loppu.

*Esittely:*
```
bool peliLoppu(const std::vector<Hakkeriryhma> &ryhmat);
```

*Parametrit:*
Parametri **ryhmat** on vektori pelissä olevista hakkeriryhmistä.

*Paluuarvo:*
True, jos pelissä on enää yksi aktiivinen ryhmä.

*Toiminta:*
Käydään ryhmät läpi ja tarkistetaan aktiivisuus ryhmaMukana-funktiolla. Jos aktiivisia ryhmiä oli alle kaksi, palautetaan **true**.

*Käyttö:*
Siirron suoritettuaan suoritaSiirto tarkistaa tällä funktiolla, onko peli päättynyt.

#### 2.5.9 Funktio: tulostaVuoro

Tulostaa vuorossa olevan ryhmän ja hakkerin nimen.

*Esittely:*
```
void tulostaVuoro(const Pelitilanne &pelitilanne, std::ostream &out);
```

*Parametrit:*
Parametri **pelitilanne** on struct, josta vuoro haetaan.
Parametri **out** on tulostusvirta.

*Paluuarvo:*
Ei paluuarvoa.

*Toiminta:*
Haetaan vuorossa olevan hakkeriryhmän ja hakkerin nimet, ja tulostetaan ne.

*Käyttö:*
Funktio suoritaSiirto kutsuu tätä funktiota vaihdettuaan vuoron.

#### 2.5.10 Funktio: vapautaHakkerit

Vapauttaa dynaamisesti luodut Hakkeri-oliot ja tyhjentää vektorit.

*Esittely:*
```
void vapautaHakkerit(std::vector<Hakkeriryhma> &ryhmat);
```

*Parametrit:*
Parametri **ryhmat** on vektori vapautettavista hakkeriryhmistä. Se tyhjennetään vapautuksen jälkeen.

*Paluuarvo:*
Ei paluuarvoa.

*Toiminta:*
Ryhmät käydään läpi ja hakkerit vapautetaan.

*Käyttö:*
Pääohjelma kutsuu tätä pelin päätyttyä.

#### 2.5.11 Funktio: tulostaTilanne

Tulostaa pelitilanteen TILANNE-komentoa varten.

*Esittely:*
```
void tulostaTilanne(const std::vector<Hakkeriryhma> &ryhmat,
std::ostream &out);
```

*Parametrit:*
Parametri **ryhmat** on vektori hakkeriryhmistä, joiden hakkerit tulostetaan.
Parametri **out** on tulostusvirta.

*Paluuarvo:*
Ei paluuarvoa.

*Toiminta:*
Ryhmien hakkerit käydään läpi ja kutsutaan niiden tulostaTilanne-jäsenfunktiota tilannerivin tulostamiseksi.

*Käyttö:*
Komentotulkki kutsuu tätä TILANNE-komennon yhteydessä.

## 2.6 Luokka: Hakkeri

Yksittäinen hakkeri. Pitää kirjaa hakkerin hermoista ja kolavarannosta.

Sisältää HakkeriTila-tyypin:

```
enum HakkeriTila {PERUS, HYOKKAYS, REITITYS, PUSKUROINTI, OFFLINE};
```

### 2.6.1 Jäsenmuuttujat

```
// Viite yleisasetukset sisältävään structiin.
const Asetukset &asetukset_;

std::string nimi_; // Hakkerin nimi.
HakkeriTila tila_; // Hakkerin tila.

// Hakkerin hallussa olevan viruksen taso. Arvoltaan 0 jos hakkerilla ei ole
// virusta.
int virustaso_;

// Hakkerit, joille viruksia reititetään. Vain REITITYS-tilassa.
std::vector<Hakkeri *> reitityskohteet_;
int reititysvuoro_; // Seuraavan kohteen indeksi.

// Jäljellä olevien toimintokertojen määrä. Vain REITITYS- ja
// PUSKUROINTI-tiloissa.
int maara_;

int kola_; // Pitää kirjaa hakkerin kolavarannoista.
int hermot_; // Pitää kirjaa hakkerin hermoista.
int maxkola_; // Hakkerin kolavarastojen maksimikoko (alkukoko).
int maxhermot_; // Vastaavasti hermojen maksimiarvo.
```

### 2.6.2 Yksinkertaiset jäsenfunktiot

Tässä pelkkä lista jäsenfunktioista, joiden toteutuksessa ei ole mitään ihmeellistä:

```
const std::string &haeNimi() const; // Palauttaa hakkerin nimen
bool online() const; // Onko hakkerin tila jokin muu kuin OFFLINE?

// Lisäksi automaattisesti tuotetut kopiorakentaja ja sijoitusoperaattori.
```

### 2.6.3 Rakentaja, parametrina asetustiedostosta lasketut tiedot

*Esittely:*
```
Hakkeri(const Asetukset &asetukset,
const std::string &nimi, int maxkola, int maxhermot);
```
*Parametrit:*
Parametri **asetukset** on viite pelin yleisasetukset sisältävään structiin.
Parametri **nimi** on hakkerin nimi.
Parametri **maxkola** on hakkerin kolavarastojen suurin koko ja samalla alkukoko.
Parametri **maxhermot** on vastaavasti hakkerin hermojen määrä.

*Jäsenmuuttujien alustus:*
Rakentajan alustuslista:

```
asetukset_(asetukset),
nimi_(nimi),
tila_(PERUS),
virustaso_(0),
reitityskohteet_(),
maara_(0),
kola_(maxkola),
hermot_(maxhermot),
maxkola_(maxkola),
maxhermot_(maxhermot)
```

### 2.6.4 Julkinen jäsenfunktio: lisaaKolaa

Suorittaa vuoron alussa tehtävän kolavaraston täydentämisen.

*Esittely:*
```
void lisaaKolaa();
```
*Parametrit:*
Ei parametreja.

*Paluuarvo:*
Ei paluuarvoa.

*Toiminta:*
Lasketaan lisättävän kolan määrä yleisasetuksen ja Hakkerin maksimikolavarastojen perusteella. Suoritetaan lisäys ja varmistetaan ettei tulos ylitä maksimivarastoja.

### 2.6.5 Julkinen jäsenfunktio: suoritaSiirto

Suorittaa siirron, muuttaen hakkerin tilaa ja tarvittaessa lähettää viruksen siirrossa olevan osoittimen kautta toisen hakkerin otaVirus-jäsenfunktiolle.

*Esittely:*
```
bool suoritaSiirto(const Siirto &siirto, std::ostream &out);
```
*Parametrit:*
Parametri **siirto** on suoritettava siirto, joka on jo tarkistettu lailliseksi.
Parametri **out** on tulostevirta, johon kirjoitetaan ilmoitukset siirron suorituksesta.

*Paluuarvo:*
False, jos hakkerin tila vaihtui, eli vuoron pitäisi päättyä. True muutoin.

*Toiminta:*
Siirron tyypin perusteella tehdään jotain seuraavista:

**KOODAA:** Tulostetaan ilmoitus "*hakkerin nimi* koodaa tason $x$ viruksen." Vaihdetaan tilaksi HYOKKAYS vaihdaTila-jäsenfunktiolla, joka nollaa tilaan liittyvät jäsenmuuttujat. Asetetaan sitten jäsenmuuttujaan `virustaso_` koodatun viruksen taso.

**PUSKUROI:** Tulostetaan ilmoitus, vaihdetaan tilaksi PUSKUROINTI ja asetetaan jäsenmuuttujaan `maara_` siirrosta saatu puskurointien määrä.

**REITITA:** Tulostetaan ilmoitus, vaihdetaan tilaksi REITITYS ja asetetaan jäsenmuuttujiin reititysten määrä ja reitityskohteet.

**RESETOI:** Tulostetaan ilmoitus ja vaihdetaan tilaksi PERUS.

**HYOKKAA:** Tulostetaan ilmoitus ja otetaan talteen jäsenmuuttujasta hallussa olevan viruksen taso. Vaihdetaan tilaksi PERUS ja lähetetään virus hyökkäyskohteelle.

**TANKKAA:** Tulostetaan ilmoitus ja kasvatetaan `hermot_`-jäsenmuuttujaa, kuitenkin varmistaen ettei maksimi ylity.

**ODOTA:** Tulostetaan ilmoitus.

Lopuksi kulutetaan kolaa hakkeripeli-moduulin laskeKulutus-funktiolta saadun tiedon perusteella. Palautetaan **true** jos siirron tyyppi oli TANKKAA, muut siirrot aiheuttavat vuoron vaihtumisen.

*Käyttö:*
Hakkeripeli-moduulin suoritaSiirto kutsuu tietyn hakkerin tätä jäsenfunktiota suorittaakseen komentotulkilta saadun siirron.

### 2.6.6 Julkinen jäsenfunktio: tarkistaSiirto

Tarkistaa siirron kelvollisuuden nykyisessä tilanteessa ja tulostaa mahdolliset virheilmoitukset.

*Esittely:*
```
bool tarkistaSiirto(const Siirto &siirto, std::ostream &err);
```
*Parametrit:*
Parametri **siirto** on tarkistettava siirto.
Parametri **err** on virheiden tulostamiseen käytetty virta.

*Paluuarvo:*
True, jos siirto oli kelvollinen. Ellei siirto ole kelvollinen, tulostetaan virheilmoitus ja palautetaan false.

*Toiminta:*
Tarkistetaan seuraavat virhetilanteet:

- Siirron tyyppi on EISIIRTOA, mikä tarkoittaa virhettä ohjelman sisäisessä rakenteessa. Tällöin tuotetaan poikkeus, jotta virhe varmasti huomataan.
- Siirron tyyppi on HYOKKAA mutta `virustaso_` on 0. Tulostetaan "Virhe: *hakkerin nimi* ei omista virusta." ja palautetaan **false**.
- Siirron tyyppi on TANKKAA mutta `tila_` ei ole PERUS. Tulostetaan "Virhe: Tämän komennon voi suorittaa vain perusmoodissa." ja palautetaan **false**.
- Siirron tyyppi on TANKKAA mutta `kola_` on pienempi kuin annettu `maara`. Tulostetaan "Virhe: Liian vähän kolaa :'(." ja palautetaan **false**.

Muutoin palautetaan **true** eikä tulosteta mitään.

*Käyttö:*
Komentotulkki käyttää tätä käyttäjältä saadun siirron tarkistamiseksi. Komentotulkki suorittaa jo siirtoa tulkitessaan osan tarkastuksista, kuten etsii hakkerit nimien perusteella.

### 2.6.7 Julkinen jäsenfunktio: otaVirus

Kutsutaan, kun tälle hakkerille on lähetetty virus. Suorittaa toimenpiteitä hakkerin tilan perusteella.

*Esittely:*
`void otaVirus(int taso);`

*Parametrit:*
Parametri **taso** on lähetetyn viruksen taso.

*Paluuarvo:*
Ei paluuarvoa.

*Toiminta:*
**Aina:** Tulostetaan ilmoitus "*hakkerin nimi* saa tason *viruksen taso* viruksen koneelleen!"

**PERUS- tai HYOKKAYS-tilassa:** Kutsutaan `kulutaHermoja`-jäsenfunktiota viruksen tasolla. Tämä funktio tulostaa tarvittavat ilmoitukset ja vaihtaa hermojen loppuessa hakkerin tilan.

**PUSKUROINTI-tilassa:** Tulostetaan ilmoitus puskuroinnista ja lisätään taso jäsenmuuttujaan `virustaso_`. Pienennetään `maara_`-jäsenmuuttujaa. Jos uusi `maara_` on 0, vaihdetaan `tila_`:ksi HYOKKAYS ja tulostetaan ilmoitus puskurin täyttymisestä.

**REITITYS-tilassa:** Haetaan seuraava reitityskohde vektorista indeksin perusteella ja päivitetään indeksiä. Tulostetaan ilmoitus reitityksestä.

Pienennetään `maara_`-jäsenmuuttujaa. Jos uusi `maara_` on 0, siirrytään tilaan PERUS ja tulostetaan ilmoitus reititysten purkamisesta.

Kutsutaan kohdehakkerin otaVirus-funktiota viruksen tasolla.

**OFFLINE-tilassa:** Tulostetaan ilmoitus "*hakkerin nimi* on terassilla. Häntä ei kiinnosta."

*Käyttö:*
Toiset hakkerit kutsuvat tätä suorittaessaan hyökkäystä tai reititystä.

### 2.6.8 Julkinen jäsenfunktio: tulostaTiedot

Tulostaa hakkerin tiedot TIEDOT-komentoa varten.

*Esittely:*
`void tulostaTiedot(std::ostream &out) const;`

*Parametrit:*
Parametri **out** on tulostusvirta.

*Paluuarvo:*
Ei paluuarvoa.

*Toiminta:*
Tulostetaan tiedot seuraavassa muodossa:

```
<hakkerin nimi>
Moodi:     <moodin kuvaus>
[Taso:     <reititysten/puskurointien määrä>] // Vain jos tila_ on REITITYS
                                              // tai PUSKUROINTI
[Hermot:   <hermot>/<maksimihermot>] // Ei offline-tilassa
[Kola:     <kola>/maksimikola>] // Ei offline-tilassa

[Virus:    tason <viruksen taso> virus] // Vain jos virustaso_ != 0
[Kohteet:  <kohde 1>, <kohde 2>, ... ] // Vain jos tila_ on REITITYS
```

*Käyttö:*
Komentotulkki kutsuu tätä suorittaessaan TIEDOT-komentoa.

### 2.6.9 Julkinen jäsenfunktio: tulostaTilanne

Tulostaa tilannelistausta varten sopivan tietorivin.

*Esittely:*
`void tulostaTilanne(std::ostream &out) const;`

*Parametrit:*
Parametri **out** on tulostusvirta.

*Paluuarvo:*
Ei paluuarvoa.

*Toiminta:*
Jos `tila_` on OFFLINE, tulostetaan "*hakkerin nimi* Terassilla".

Muutoin tulostetaan "*hakkerin nimi hermot/maxhermot kola/maxkola moodi*". Moodi on Irkkaa, Koodaa, Reitittää tai Puskuroi.

*Käyttö:*
Hakkeripeli-moduulin tulostaTilanne-funktio kutsuu vuorotellen kaikkien hakkerien tätä jäsenfunktiota.

### 2.6.10 Yksityinen jäsenfunktio: kulutaHermoja

Vähentää hakkerin hermoja ja tulostaa siihen liittyvän ilmoituksen. Hermojen loppuessa tulostaa ilmoituksen ja vaihtaa hakkerin tilan. Palauttaa tiedon hermojen riittämisestä.

*Esittely:*
`bool kulutaHermoja(int maara, std::ostream &out);`

*Parametrit:*
Parametri **maara** on hermoista vähennettävä määrä.
Parametri **out** on tulostusvirta.

*Paluuarvo:*
True, jos hermot riittivät ja false muutoin.

*Toiminta:*
Tulostetaan ilmoitus "*hakkerin nimi* menettää hermojaan $x$ pisteen verran!". Pienennetään jäsenmuuttujaa `hermot_` tämän verran.

Jos tulos on negatiivinen tai 0, tulostetaan ilmoitus "*hakkerin nimi* menettää hermonsa ja lähtee terassille." ja palautetaan **false**. Muutoin palautetaan **true**.

*Käyttö:*
Jäsenfunktio otaVirus kutsuu tätä viruksen kuluttaessa hakkerin hermoja. Jäsenfunktio kulutaKolaa kutsuu tätä ellei kolamäärä riitä.

### 2.6.11 Yksityinen jäsenfunktio: kulutaKolaa

Ottaa tarvittavan määrän kolaa hakkerin kolavarastoista. Ellei kola riitä, määrä otetaan hermoista. Tankkauksen yhteydessä puuttuvaa määrää ei pidä ottaa hermoista, mutta kolan riittäminen tankkaukseen tarkistetaan ennen siirron toteutusta.

Palauttaa totuusarvon joka kertoo kolan/hermojen riittämisestä. False tarkoittaa hermojen menneen, eli hakkerin siirtyneen terassille.

*Esittely:*
`bool kulutaKolaa(int maara, std::ostream &out);`

*Parametrit:*
Parametri **maara** on tarvitun kolan määrä.
Parametri **out** on tulostusvirta mahdollista hermojen kulumisesta kertovaa ilmoitusta varten.

*Paluuarvo:*
True, jos kolaa tai hermoja riitti. False jos hakkeri menetti hermonsa.

*Toiminta:*
Tarkistetaan riittääkö kolan määrä. Jos riittää, vähennetään se suoraan jäsenmuuttujasta `kola_` ja palautetaan **true**.

Muutoin lasketaan kolavarastojen ja tarvitun määrän erotus, asetetaan jäsenmuuttujaan `kola_` arvo 0 ja kutsutaan jäsenfunktiota `kulutaHermoja` erotuksella. Palautetaan saatu arvo.

*Käyttö:*
Jäsenfunktio suoritaSiirto käyttää tätä siirron tarvitseman kolan kuluttamiseen.

### 2.6.12 Yksityinen jäsenfunktio: vaihdaTila

Vaihtaa hakkerin tilaa nollaten samalla tilaan liittyvät jäsenmuuttujat.

*Esittely:*
```
void vaihdaTila(HakkeriTila uusitila);
```

*Parametrit:*
Parametri **uusitila** on hakkerin seuraava tila.

*Paluuarvo:*
Ei paluuarvoa.

*Toiminta:*
Asetetaan uusi tila jäsenmuuttujaan `tila_`. Nollataan `virustaso_` ja `maara_`. Tyhjennetään `reitityskohteet_`.

*Käyttö:*
Jäsenfunktio suoritaSiirto käyttää tätä hakkerin tilan vaihtamiseen. Tilaan liittyvät arvot asetetaan tämän funktion kutsumisen jälkeen.

# 3 Loppudokumentaatio

## 3.1 Muutokset alkuperäiseen suunnitelmaan

Alunperin hakkeripeli-moduulin sisältö oli toteutettu yhtenä luokkana. Assari neuvoi muuttamaan tämän, sillä C++:ssa ei ole tapana tehdä luokkia tällaisista asioista.

Siirroissa ja muissa yhteyksissä Hakkeri-olioihin viitattaessa oli käytetty indeksiä HakkeriPeli-olion sisältämässä vektorissa. Tämä neuvottiin muuttamaan pointtereiksi, mikä yksinkertaisti ohjelman toteutusta. Toisaalta tämä monimutkaisti huvikseni tekemääni AI:ta, jonka täytyy suorittaa pelitilanteen kopiointia.

Välipalautussuunnitelmaan lisäsin kaksi moduulia, poikkeukset sekä merkkijonojen käsittelyn. Virheenkäsittelyä muussa ohjelmassa muutettiin käyttämään poikkeuksia. Poikkeusten käytön ansiosta virheilmoitusten tulostusta pystyi keskittämään ja monimutkaisilta if-rakenteilta vältyttiin.

Vuorojen hallinta oli aluksi toteutettu dequeta pyörittämällä. Koska ryhmät, hakkerit ja reitityskohteet pitää kuitenkin tulostaa samassa järjestyksessä kuin ne on annettu, vaihdoin tämän käyttämään vektoria ja indeksimuuttujaa.

Tulosteet suoritettiin alunperin HakkeriPeli-moduulin sisältämien osoittimien kautta. Nyt tulostus- ja virhevirrat välitetään parametreina niitä käyttäville funktioille.

## 3.2 Kuvaus harjoitustyön tekemisestä

Toteuttaminen suunnittelun jälkeen oli helppo, joskin HakkeriPeli-luokasta luopuminen vaati jonkin verran uutta suunnittelua.

Välipalautusdokumenttiin meni noin 12 tuntia. Vaiheen 1 vaatiman asetustiedostojen lukemisen toteuttamiseen meni noin 8 tuntia. Vaiheen 2 komentotulkkiin kului noin 4 tuntia. Vaiheen 3 yhteydessä toteutin jo pelin testausta vaille valmiiksi, mihin meni noin 10 tuntia. Lisäksi kului testaukseen noin 4 tuntia ja loppudokumenttiin 8 tuntia. Yhteensä harjoitustyöhön kului siis 46 tuntia.

Mielestäni harjoitustyöni onnistui hyvin.

## 3.3 Palautetta

Harjoitustyön tehtävänanto yms. dokumentit oli laadittu hyvin, mutta esimerkkiajot olisi saatu tehdä tarkemmin. Nyt niissä oli välilyönneissä ja tyhjissä riveissä vaihtelevaa käyttäytymistä, minkä vuoksi diffaaminen oli työlästä.

Dokumenttipohja oli hyödyllinen, mutta alkusivun otsikko olisi saanut olla ajan tasalla jo valmiiksi.

Tentti oli mielestäni hieman huono, siinä painottui liikaa juuri luennoilla esitetyt käsitykset asioista ja parhaista käytännöistä. Joskin näihinkin kohtiin saa yleisellä tietämyksellä päätellyä vastauksia, mutta tällaista subjektiivista tietoa olisi parempi kysyä “Pohdi syitä, miksi ...” eikä “Miksi ...”. Tällöin ei jäisi tuntumaa, että kysymykseen on yksi oikea prujussa esitetty vastaus.