Turun yliopisto
University of Turku

Olio-ohjelmoinnin metodiikka
Viikkoharjoitustyö 3

Ryhmä:
Pasi Toivanen (517487)
Janina Kuosmanen (516580)
Santeri Loitomaa (516587)
Tommi Heikkinen (517749)

Viikkoharjoitustyö 3
6. lokakuuta 2018
18 sivua

Turun yliopisto
Tulevaisuuden teknologioiden laitos
Tietotekniikka
Olio-ohjelmoinnin metodiikka

Sisältö

1 Tehtävä 1

Tehtävässä tutustutaan oom-kit -kehykseen ja siitä löytyviin luokkiin

1.1 fi.utu.oomkit -luokat

fi.utu.oomkit.gui -paketista löytyy luokat:

```
1  public interface Console {}
2  class DefaultDialogFactory implements DialogFactory {}
3  public interface DialogFactory {}
4  class FXConsole extends OutputStream implements Console {}
5  public interface MainWindow extends WindowContent {}
6  public abstract class MainWindowBase implements MainWindow {}
7  class MergedStream extends OutputStream {}
8  public abstract class OOMApp extends Application {}
9  public class ReactiveCanvas<X> extends SimpleCanvas implements Observer<X> {}
10 public class ReactiveLabel<T> extends Label implements Observer<T> {
11         public interface LabelHandler<T> {}
12 }
13 public class SimpleCanvas extends Canvas {}
14 class StreamWrapper extends OutputStream {}
15 public interface WindowContent {}
```

Joista kaikki ovat julkisia paketin sisällä, mutta paketin ulkopuolella näkyvistä on jätetty luokat DefaultDialogFactory, FXConsole, MergedStream ja StreamWrapper. fi.utu.oomkit -paketista löytyy luokat:

```
1  public final class AppConfiguration {}
2  public interface AppLogic extends KeyHandler, Scheduled {}
```

Joista kaikki ovat julkisia luokkia niin paketin sisällä kuin ulkopuolella.

1.2 OOMApp -luokka sovelluksen ikkunan luomiseksi

OOMApp -luokka peritään ja sille kirjoitetaan generateMainWindow -metodi.

GenerateMainWindow -metodilla määritellään mikä on ohjelman (=app) nimi ja kuinka ison alueen ohjelma käyttää. Metodin tulee palauttaa MainWindow -alaluokka. OOMApp -luokan käyttöä on esitetty DemoApp1 -tutoriaalissa seuraavasti:

```
1  public class DemoApp1 extends OOMApp {
2      // alustaa pelilogiikan
3      final static LaatikkoGameLogic gameLogic = new LaatikkoGameLogic();
4
5      // kytkee piirtopinnan käyttöliittymään
6      @Override
7      protected MainWindow generateMainWindow(String appName, double width,
8      double height) { return new SimpleMainWindow(appName, width, height) {
9              @Override public SimpleCanvas mainContent() {
10                 return gameLogic.piirtoPinta;
11             }
12         };
13     }
14 }
```

EmptyApp -luokassa generateMainWindow -metodille on tehty mielekäs esimerkki-implementaatio, jolloin sitä voi käyttää sellaisenaan muodostaessa ohjelman ikkunaa. Siitä löytyy valmiiksi ajettava main-metodi.

1.3 Ohjelman ikkunan sisällön hallitseminen

Ikkuna luodaan käyttämällä luokkaa MainWindow. Huomionarvoista on, että MainWindow -olion sisältö koostuu ylä- ja alapalkista, sekä Canvas-piirtoalueesta. Oletuksena nämä ovat tyhjiä, jolloin oman sisällön voi luoda ylikirjoittamalla metodit topBarContent(), mainContent(), bottomBarContent().

```
1  public class EmptyApp extends OOMApp {
2          @Override
3          protected MainWindow generateMainWindow(){
```

```
4                 return new SimpleMainWindow(appName, width, height) {
5
6                         @Override
7                         public List<Node> topBarContent() {
8                                 //return yläpalkin sisältö
9                         }
10
11                        @Override
12                        public Canvas mainContent() {
13                                //return pääsisältö
14                        }
15
16                        @Override
17                        public List<Node> bottomBarContent() {
18                                //return alapalkin sisältö
19                        }
20                }
21        }
22}
```

Ylä- ja alapalkkiin voidaan lisätä mitä tahansa graafisia solmuja, jotka voivat olla erilaisia graafisia käyttöliittymäkomponentteja. Ylikirjoittamalla topBarContent() -metodin voidaan sijoittaa käyttöliittymään lista mitä tahansa elementtejä kuten esimerkiksi luokkien Canvas, ImageView, Shape tai Button luokkien oliot. Yläpalkin lisäämisessä voitaisiin käyttää esimerkiksi seuraavaa metodia:

```
1 @Override
2 public List<Node> topBarContent() {
3         return Arrays.asList(
4                 new Button("File"),
5                 new Button("Edit"),
6                 new Button("Window"),
7                 new Button("Help")
8         );
9 }
```

Tämä lisää työkalurivin kaltaiset painikkeet yläpalkkiin

1.4 Ohjelmanaikaiset tapahtumat

OOMApp -luokan olion konstruktorilla pystytään luomaan useanlaisia logiikoita ohjelman suoritukseksi. Oma logiikka voidaan lisätä käyttämällä OOMApp konstruktoria, johon syötetään ohjelmalogiikka. Ohjelmalogiikkaluokka peritään AppLogic -luokalta. Seuraavassa esimerkissä on esitetty, kuinka MyLogic -luokalla voidaan hallita ohjelmalogiikkaa niin, että asiakkaan koodi suoritetaan joka 25ms.

```
1  public class MyApp extends OOMApp {
2          public MyApp() {
3                  super(new MyLogic());
4          }
5  }
6
7  public class MyLogic extends AppLogic {
8          @override
9          AppConfiguration configuration() {
10                 return new AppConfiguration(25, "Asiakkaan ohjelma", true);
11         }
12
13         @override
14         void tick() {
15                 //asiakkaan koodi
16         }
17 }
```

- Pasi Toivanen (517487)

2 Tehtävä 2

2.1 a-kohta

Määrittelyn muutoksen aloittaisin metodista lisääSolmu, koska solmuja ei enää luoda irrallisina vaan osana muita prosesseja, voi sen tyypin vaihtaa privateksi.

```
1 /**
2 * @.pre leima != null && !sisältääSolmun(leima)
3 * @.post RESULT.sisältääSolmun(leima) &
4 * RESULT.poistaSolmu(leima).equals(this)
5 */
6 private lisääSolmu(String leima)
```

Vaihtoehtoisesti tyypin voi pitää publicina, jos metodia kuitenkin haluaa luokan ulkopuolelta kutsua.

Tärkeämpi osuus on muuttaa SuunnattuGraafi luokan constructoria siten, että uutta graafia luodessa pitää sille antaa syötteenä luotava solmu, josta rakentaminen lähtee liikkeelle (ns. juurisolmu).

```
1 /**
2 * @.pre juuri != null
3 * @.post solmumäärä() == 1 & kaarimäärä() == 0 &
4 *         RESULT.sisältääSolmun(juuri) &
5 *         RESULT.poistaSolmu(juuri).equals(this)
6 */
7 public SuunnattuGraafi(String juuri)
```

Konstruktori siis kutsuu aiemmin muokattua lisääSolmu(leima) metodia ja luo sen pohjalta juurisolmun.

Uuden kaaren lisäämisessä pitää huomioida, että kohdejuuren ei ole pakko olla vielä olemassa, vaan se voidaan luoda kaaren yhteydessä, mutta muita muutoksia metodi ei juuri vaadi.

```
1  /**
2  * @.pre (lähtöleima != null & tuloleima != null) &&
3  * sisältääSolmun(lähtöleima) & !sisältääKaaren(lähtöleima, tuloleima)
4  * @.post RESULT.sisältääKaaren(lähtöleima, tuloleima) &
5  * RESULT.poista(lähtöleima, tuloleima).equals(this) &
6  * RESULT.annaPaino(lähtöleima, tuloleima) == paino &
7  * if(!sisältääSolmun(tuloleima))  RESULT.sisältääSolmun(tuloleima) &
8  *                                 RESULT.poistaSolmu(tuloleima).equals(this)
9  */
10 public SuunnattuGraafi lisääKaari(String lähtöleima, String tuloleima,
11 double paino)
```

Jos tuloleima nimistä solmua ei ole vielä olemassa metodi luo sellaisen, jatkaen sitten toimintaansa normaalisti. Jos solmu on jo olemassa toiminta ei eroa aiemmasta.

2.2 b-kohta

Luokan SuunnattuGraafi nimi kannattaisi alkajaisiksi muuttaa, esim. SuunnatonGraafi, vastaavasti kaikki metodit jotka viittaavat tähän nimeen yms. tulisi muuttaa viitaamaan uuteen nimeen. Terminologian selkeyttämiseksi vaihtaisin myös muuttujat lähtöleima ja tuloleima, vaikkapa korvaaviksi päätysolmu1 ja päätysolmu2, kuvastamaan ettei ole alkua ja loppua vaan kaaren kaksi päätyä.

Terminologian muutosten lisäksi on luokassa vain kaksi metodia jotka ovat kiinnostuneita kaaren edeltäjist ja seuraajista, metodit Solmujoukko edeltjät Solmujoukko seuraajat. Nämä voisi poistaa ja korvata uudella versiolla Solmujoukko parit

```
1  /**
2  * @.pre leima != null && sisältääSolmun(leima)
3  * @.post RESULT == (solmuun linkitettyjen kaarien toisissa päissä olevat solmut)
4  */
5  public Solmujoukko parit(String leima)
```

Solmujoukko parit palauttaa kaikki solmut, jotka ovat yhteydessä kyseiseen slomuun jo olemassaolevien kaarien kautta.

Tekijänä Tommi Heikkinen (517749)

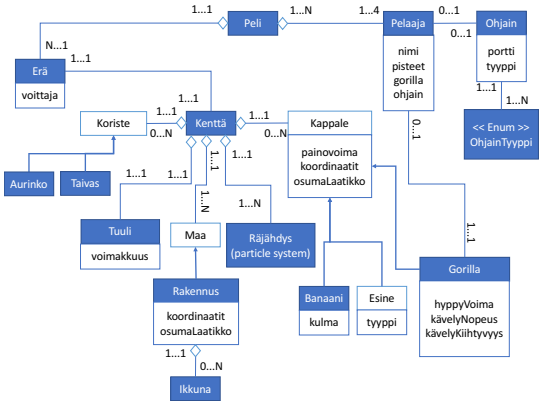
3 Tehtävä 3: Gorilla -peli

3.1 Peliä kuvaavat luokkakokonaisuudet (a,b)

Kuvausten ja tehtävien pohjalta on tehty luokkakaavio kuvassa 3.1, jolla pyrin kuvaamaan lopullista peliä. Luokkakaaviossa on olioiden hierarkian lisäksi kuvattu niiden määräsuhteita ja abstraktit luokat on merkitty sinisen sijaan valkoisella pohjalla.

Kuvasta 3.1 on jätetty selkeyden vuoksi merkitsemättä luokat vektori, osumalaatikko ja rajapinnat aktiivinen, piirrettävä, heitettävä ja kerättävä. Nämä kaikki luokat voidaan jakaa suunnittelu, analyysi ja toteutusluokkiin (kts 3.2), jossa suunnitteluluokkiin kuuluu pääasiassa abstrakteja luokkia. Analyysiluokista löytyy pelin rakenteeseen liittyviä luokkia ja tuuli-luokka jota ei ruudulla suoraan näe vaan liittyy enemmän pelimoottorin toimintaan.

Toteutusluokista löytyy selkeitä käsitteitä, joita pelissä nähdään visuaalisina ele-



Kuva 3.1: Luokkakaavio Gorilla-Pelistä

Suunnittelu	Analyysi	Toteutus
Aktiivinen	Kenttä	Rakennus, Ikkuna
Piirrettävä	Erä	Gorilla
Kappale	Peli	Pelaaja
Heitettävä	Tuuli	Aurinko
Kerättävä		Räjähdys (particle system)
Koriste		Pisteet
Maa		Banaani

Kuva 3.2: Suunnittelu-, analyysi ja toteutusluokkiin jako

mentteinä ja näitä näkyviä olioita yhdistää myös rajapinta Piirrettävä, joka määrittelee millaiset metodit tulee luokasta löytyä, että se voidaan piirtää ruudulle. Aktiivisten luokkien oliot ovat sellaisia, joille löytyy oma päivitysmetodi mikä toteutetaan jokaisen tick() -metodin tai toisin sanoen jokaisen pelin päivitysiteraation aikana.

Esineet ovat kerättäviä asioita pelikentiltä, jotka voisivat olla esimerkiksi pisteitä, lisä elämiä tai banaaneja, joita voi heitellä toisten pelaajien niskaan. Itse rajapinta Heitettävä kuvaa kaikkia olioita, joita pelaaja voi singota omalla gorillallaan. Nämä kaikki rajapinnat kuvaavat pelissä tapahtuvia vuorovaikuttavia elementtejä, mutta sen lisäksi tarvitaan koristeita, jotka tuovat pelin taustaksi grafiikkaa ja mallintavat selkeästi pelaajalle missä rakennukset peliruudulla ovat.

3.2 Pakettikokonaisuudet ja uudelleenkäytettävyys (c, d)

Kuvan 3.1 luokkia voidaan jakaa ryhmiin, jotka kuvaavat pääsääntöisesti jotain pelin osa-aluetta niin, että vuorovaikutukset ryhmän sisällä on suurempaa kuin ryhmien välillä. Ryhmät voitaisiin nimetä paketeiksi seuraavalla tavalla:

- Player (Pelaaja, Ohjain)
- Engine (Kenttä, Kappale, Maa, Esine)

- Map (Rakennus, Ikkuna)
- Character (Banaani, Gorilla)
- Graphics (Aurinko, Taivas)
- Menu (Peli)

Player-paketissa olisi työkaluja pelaajien ja pelaajien käyttämien ohjainten hallintaan. Engineä pystytään uudelleenkäyttämään, kun halutaan tehdä samanlaiseen pelimekaniikkaan liittyvä peli tai modi. Menu-pakettilla voidaan muodostaa toinen moninpeli, jossa 1-4 pelaajaa kamppailee toisiaan vastaan erissä, joiden pisteitä lasketaan. Map, Character, Graphics paketit liittyvät tiiviisti pelin ulkonäköön ja olemukseen, joten muita pelejä tehdessä näitä paketteja tuskin pystyy paljon käyttämään hyväksi.

3.3 Fysiikan mallinnuksen soveltaminen (e)

Fysiikanmallinnuksen soveltaminen pelimoottoriin voidaan tehdä monella tavalla ja se riippuu pääsääntöisesti pelistä. Joissain arcade -peleissä fysiikan lakeja halutaan venyttää ja muokata niin, että pelikokemus on viihdyttävämpi. Tässä tapauksessa voimille ja kentille voi tehdä omat oliot, jolloin pelin logiikan kehityksessä näitä voimia tai impulsseja voidaan säätää yliampuviksi tarpeen mukaan.

Jos pelissä on kyseessä enemmän simulointi, jossa halutaan, että samat fysiikan lait pätevät kaikkialla pelikentällä kaikille olioille, niin esimerkiksi Painovoima, Voima ja Impulssi -olioiden tekeminen on turhaa. Fysiikan lait pystytään silloin kuvaamaan esimerkiksi kuvan 3.1 kaltaisessa luokkahierarkiassa pelkästään Kappale -luokan sisällä, jolloin yksikään aliluokka tai olio ei pysty tästä säännöstä poikkeamaan.

Esimerkiksi painovoiman, tai pikemminkin normaalikiihtyvyyden suuruus voitaisiin määrätä final static -muodossa vektoriksi, jolloin kaikki kappaleet pelissä joutuisi-

vat sitä noudattamaan. Normaalikiihtyvyyden vektori kannattaisi pitää vähintään static -muotoisena, jolloin jos maan painovoimakentästä siirryttäisiin kuun painovoimaan, niin kaikki pelialueen oliot välittömästi päivittyisivät tähän. Pelimoottori ei näytä välttämättä eheältä, jos jotkin kappaleet liikkuvat kuin 'eri planeetalla'.

Voima tai Painovoima -luokista luopuminen ei kumminkaan tarkoita sitä, että pelimoottorin mekaniikan toteuttavissa metodeissa nämä käsitteet ei tarvitsisi olla hyvin dokumentoituja. Päinvastoin: metodien toteutuksessa tulee nimetä hyvin selkeästi milloin on kyse kiihtyvyydestä, milloin voimasta tai milloin liiketilasta. Dokumentaatiossa kannattaa myös merkitä, että mikä rivi toteuttaa minkäkin fysiikan lain: Newtonin lait, Galileon suhteellisuus, voimien resultanttivektorin summaus ja jotta fysiikan mallinnuksen tuottava algoritmi on selkeästi luettavissa.

Tekijänä Pasi Toivanen (517487)

4 Tehtävä 4

Tehtävän tarkoituksena on luoda lukitusmenetelmä OOMKit-ohjelmiin, joka estää kaiken näkyvyyden ohjelmaan kunnes ohjelman avaava koodi syötetään täysin oikein. Aikarajaa salasanan syöttöön ei ole.

4.1 Lukitus-luokka (kohdat a ja b)

Kohdassa a tuli määrittää luokka Lukitus, jolla voisi lukita OOMKit-ohjelman. Kohdassa b se taas tuli toteuttaa. Kirjaan molemmat kohdat tähän samaan osioon.

Lukitus-luokka

```
1  /**
2   * @author Santeri Loitomaa
3   */
4  public class Lukitus {
5    private ArrayList<Key> codeLukituskoodi;
6    private ArrayList<Key> yritysLukituskoodi;
7    private boolean lukossa;
8    /**
9     * Konstruktori lukolle.
10    * @param codeLukituskoodi
11    */
12   public Lukitus(ArrayList<Key> codeLukituskoodi) {
13     this.codeLukituskoodi = codeLukituskoodi;
14     this.yritysLukituskoodi = new ArrayList<Key>();
15     this.lukossa = false;
16   }
```

```
17  /**
18   * Asettaa boolean lukossa-arvon oikeaksi.
19   * @.pre Tätä metodia tulee kutsua vain jos ohjelma on lukossa.
20   * @.post   if(merkki == oikein) lukossa = true;
21   *       if(merkki != oikein) lukossa = true;
22   *       (kun kaikki merkit oikein) lukossa = false;
23   * @param Key merkki
24   */
25  public void tarkista(Key merkki){
26    yritysLukituskoodi.add(merkki);
27    if(codeLukituskoodi.equals(new ArrayList<Key>())) {
28      yritysLukituskoodi = new ArrayList<Key>();
29      lukossa = false;
30    }
31    else if(codeLukituskoodi.get(yritysLukituskoodi.size()-1) !=
32        yritysLukituskoodi.get(yritysLukituskoodi.size()-1)) {
33      System.out.println("Nyt meni väärin. Aloita alusta.");
34      yritysLukituskoodi = new ArrayList<Key>();
35      lukossa = true;
36    }
37    else if(codeLukituskoodi.size() == yritysLukituskoodi.size()) {
38      System.out.println("Koodi on syötetty oikein.");
39      yritysLukituskoodi = new ArrayList<Key>();
40      lukossa = false;
41    }
42  }
43  /**
44   * Kertoo onko lukko lukossa.
45   * @return true jos lukossa.
46   */
47  public boolean onLukossa() {
48    return lukossa;
49  }
50  /**
51   * Merkitsee lukon lukituksi.
52   * @.post lukossa = true;
53   */
54  public void lukitse() {
55    lukossa = true;
56  }
57  /**
```

```
58      * Vaihtaa salasanan.
59      * @.post codeLukituskoodi = codeLukituskoodi
60      * @param codeLukituskoodi
61      */
62    public void setCodeLukituskoodi(ArrayList<Key> codeLukituskoodi) {
63      this.codeLukituskoodi = codeLukituskoodi;
64    }
65    /**
66      * Kertoo nykyisen lukituskoodin.
67      * @return codeLukituskoodi
68      */
69    public ArrayList<Key> getCodeLukituskoodi(){
70      return codeLukituskoodi;
71    }
72  }
```

Tulin tällaisen lopputulokseen. Tämä toimii melko hyvin lisättynä OOMKitin DemoApp3:een pienten muutosten kera.

4.2 DemoApp3-luokka (kohta c)

DemoApp3:een tein seuraavat muutokset jotta Lukitus-luokkaa voitaisiin käyttää hyvin.

```
1  class LaatikkoGameLogic2 implements AppLogic {
2    ...
3    private Lukitus lukko = new Lukitus(new ArrayList<Key>());
4    private boolean vaihdetaankoSalasana = false;
5    private boolean vaihdetaanSalasanaa = false;
6    private ArrayList<Key> uusiSalasana = new ArrayList<Key>();
7    private boolean lukitaanko = false;
8    ...
9    @Override
10   public void tick() {
11     ...
12     // piilottaa näkymän jos lukossa
13     piirtoPinta.asetaPiilotus(lukko.onLukossa());
14     ...
15   }
16   // käsittele näppäimen painallus
17   @Override
18   public void handleKey(Key k) {
19     System.out.println(k);
20     if(lukko.onLukossa()) {
21       lukko.tarkista(k);
22     }
23     else if(vaihdetaanSalasanaa) {
24       uusiSalasana.add(k);
25       if(k == Key.Backspace) {
26         uusiSalasana = new ArrayList<Key>();
27         System.out.println("Salasanan␣vaihto␣peruutettu.");
28         vaihdetaanSalasanaa = false;
29         vaihdetaankoSalasana = false;
30       }
31       if(k == Key.Enter) {
32         lukko.setCodeLukituskoodi(uusiSalasana);
33         uusiSalasana = new ArrayList<Key>();
34         System.out.println("Salasana␣vaihdettu.");
35         vaihdetaanSalasanaa = false;
36         vaihdetaankoSalasana = false;
```

```
37          }
38      }
39      else if(lukitaanko && k == Key.N) {
40        lukitaanko = false;
41        System.out.println("Lukitus␣peruutettiin.");
42      }
43      else if(lukitaanko && k == Key.Y) {
44        lukitaanko = false;
45        lukko.lukitse();
46        System.out.println("Lukittu.");
47      }
48      else if(vaihdetaankoSalasana && k == Key.N) {
49        vaihdetaankoSalasana = false;
50      }
51      else if(vaihdetaankoSalasana && k == Key.Y) {
52        vaihdetaanSalasanaa = true;
53        System.out.println("Anna␣uusi␣salasana␣merkki␣kerrallaan.");
54        System.out.println("Tallenna␣painamalla␣Enter.");
55        System.out.println("Peruuta␣painamalla␣Backspace.");
56      }
57      else if(k == Key.Space) {
58        System.out.println("Haluatko␣lukita␣ohjelman?␣Y/N");
59        lukitaanko = true;
60      }
61      else if(k == Key.Backspace) {
62        System.out.println("Nykyinen␣salasana:");
63        System.out.println(lukko.getCodeLukituskoodi());
64      }
65      else if(k == Key.Enter) {
66        System.out.println("Haluatko␣asettaa␣uuden␣salasanan?␣Y/N");
67        vaihdetaankoSalasana = true;
68      }
69    }
70  }
```

Lukitusta voidaan hallita handleKey(Key k)-metodilla.

4.3 Testi-luokka (kohta d)

En juuri tiedä, miten testiluokka tulisi tehdä Jqwikilla, joten teen sen JUnitilla. Ensin testataan oikealla salasanalla ja sitten uudelleen lukittuna väärällä.

```
1  class Testi {
2    static Lukitus lukko;
3    static ArrayList<Key> salasana = new ArrayList<Key>();
4    @BeforeAll
5    static void setUpBeforeClass() throws Exception {
6      salasana.add(Key.T);
7      salasana.add(Key.E);
8      salasana.add(Key.S);
9      salasana.add(Key.T);
10     salasana.add(Key.I);
11     lukko = new Lukitus(salasana);
12     lukko.lukitse();
13   }
14   @Test
15   void test() {
16     lukko.tarkista(Key.T);
17     lukko.tarkista(Key.E);
18     lukko.tarkista(Key.S);
19     lukko.tarkista(Key.T);
20     lukko.tarkista(Key.I);
21     if(lukko.onLukossa()) {
22       fail("Salasana␣ei␣toiminut");
23     }
24     lukko.lukitse();
25     lukko.tarkista(Key.T);
26     lukko.tarkista(Key.E);
27     lukko.tarkista(Key.S);
28     lukko.tarkista(Key.Y);
29     lukko.tarkista(Key.I);
30     if(!lukko.onLukossa()) {
31       fail("Salasana␣toimi␣vaikkei␣pitänyt");
32     }
33   }
34 }
```

Testi ei tuottanut yhtään virhettä ja konsolin tulosteet näyttävät oikealta.

```
1 Koodi on syötetty oikein.
2 Nyt meni väärin. Aloita alusta.
3 Nyt meni väärin. Aloita alusta.
```

Tekijänä Santeri Loitomaa (516587)