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
16 sivua

Turun yliopisto
Tulevaisuuden teknologioiden laitos
Tietotekniikka
Olio-ohjelmoinnin metodiikka

Sisältö

1 Tehtävä 1: Mato pelin mallinnus

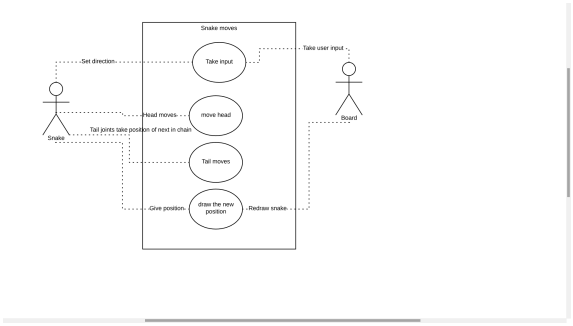
Tehtävänä oli malintaa kolme matopelin tapausta käyttötapauskaavioilla

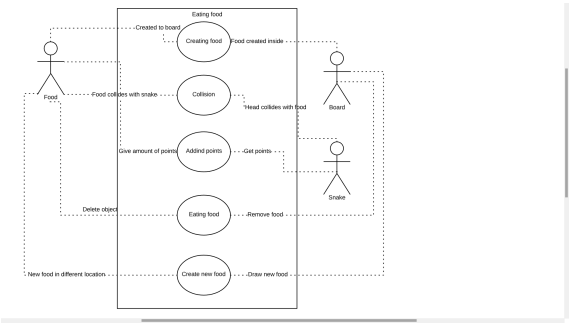
1.1 Maton liikkuminen

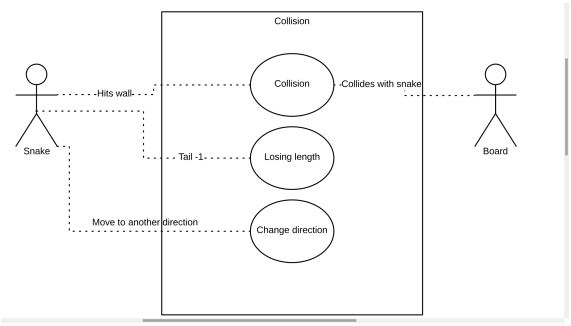
Madon liikkuminen kehyksessä tapahtuu niin että, käyttäjä antaa suunnan mihin liikutaan ja mato liikkuu jokaisella tickillä vähäsen siihen suuntaan kunnes pelaaja vaihtaa suuntaa. Pelaaja ohjaa madon päätä ja madon häntä tulee perässä niin että jokainen hännän osa saa koordinaateiksi sitä edeltävän osan vanhan paikan.

1.2 Evään syönti

Kun madon pään koordinaatit ovat samat kuin evään, eväs syödään eli poistetaan kehyksestä. Mato saa pisteitä eväältä ja sen häntä kasvaa yhden pykälän. Uusi eväs asetetaan jonnekin kehykseen.







1.3 Törmäys

Jos mato osuu kehyksen seinään se menettää pituutta ja pysähtyy kunnes sille annetaan uusi suunta.

- Janina Kuosmanen (516580)

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)

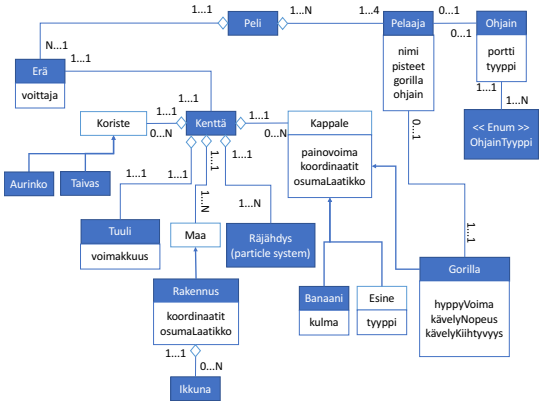
3 Tehtävä 3: Gorilla -peli

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 mallinnus OOM-kurssin aiheuttaman tuskan minimoivasta metodista.

4.1 Opiskelija-luokat (kohdat a)

Kohdassa a tuli määrittää ja toteuttaa luokat TavallinenOpiskelija ja ValeasuOpiskelija. Tulin lopputulokseen että TavallinenOpiskelija on vain Opiskelija-rajapinnan toteutus mutta ValeasuOpiskelija on nimenomaan valeasu, jonka ainoa arvo on TavallinenOpiskelija ja kaikki metodit suunnataan suoraan tälle opiskelijalle.

TavallinenOpiskelija-luokka

```
1  public class TavallinenOpiskelija implements Opiskelija {
2    String nimi;
3    int opNumero;
4    OOMTilanne oomTilanne;
5    public TavallinenOpiskelija(String nimi, int opNumero, boolean hereillä) {
6      this.nimi = nimi;
7      this.opNumero = opNumero;
8      this.oomTilanne = new OOMTilanne();
9      oomTilanne.hereillä = hereillä;
10   }
11   /**
12    * @.pre true
13    * @.post RESULT == (Palauttaa tilanneolion)
14    */
15   @Override
```

```
16  public OOMTilanne annaOOMTilanne() {
17    return oomTilanne;
18  }
19  /**
20   * Asettaa opiskelijan nimen ja op. numeron
21   *
22   * @.pre nimi != null && opNumero>0
23   * @.post (nimi & op.numero asetettu)
24   **/
25  @Override
26  public void asetaNimiJaOpNumero(String nimi, int opNumero) {
27    this.nimi = nimi;
28    this.opNumero = opNumero;
29  }
30  /**
31   * @.pre true
32   * @.post annaOOMTilanne().ilmoittautunut == true && annaOOMTilanne().hereillä
33   *         == false && annaOOMTilanne().luennolla == false &&
34   *         (OLD(annaOOMTilanne().ilmoittautunut) || Maailma.tuskanMäärä ==
35   *         OLD(Maailma.tuskanMäärä) + 1000)
36   */
37  @Override
38  public void ilmoittauduOOMKurssille() {
39    if(!oomTilanne.ilmoittautunut) {
40      oomTilanne.ilmoittautunut = true;
41      Maailma.lisääTuskaa(1000);
42    }
43  }
44  /**
45   * @.pre annaOOMTilanne().ilmoittautunut == true && annaOOMTilanne().hereillä ==
46   *       false
47   * @.post Maailma.tuskanMäärä == OLD(Maailma.tuskanMäärä) + 10 &&
48   *         annaOOMTilanne().luennolla == true
49   */
50  @Override
51  public void osallistuLuennolle() {
52    oomTilanne.luennolla = true;
53    Maailma.lisääTuskaa(10);
54    if(oomTilanne.hereillä) {
55      Maailma.lisääTuskaa(90);
56    }
```

```
57    }
58    /**
59     * @.pre 0     aikaaLuennonAlusta < 90
60     * @.post annaOOMTilanne().hereillä == true
61     */
62    @Override
63    public void herää(int aikaaLuennonAlusta) {
64      oomTilanne.hereillä = true;
65    }
66    /**
67     * @.pre annaOOMTilanne().luennolla == true
68     * @.post annaOOMTilanne().luennolla == false && annaOOMTilanne().hereillä ==
69     *        false
70     */
71    @Override
72    public void poistuLuennolta() {
73      oomTilanne.luennolla = false;
74      oomTilanne.hereillä = false;
75    }
76    /**
77     * @.pre annaOOMTilanne().ilmoittautunut == true && annaOOMTilanne().hereillä ==
78     *       true
79     * @.post Maailma.tuskanMäärä == OLD(tuskanMäärä) + 90 -
80     *        hereilläAlkaenMinuutista && annaOOMTilanne().hereillä == true
81     */
82    @Override
83    public void vastaaKysymykseen(int aikaaLuennonAlusta) {
84      herää(aikaaLuennonAlusta);
85      Maailma.lisääTuskaa(90-aikaaLuennonAlusta);
86    }
87  }
```

ValeasuOpiskelija-luokka

```
1  public class ValeasuOpiskelija implements Opiskelija {
2    Opiskelija opiskelija;
3    public ValeasuOpiskelija(Opiskelija opiskelija) {
4      this.opiskelija = opiskelija;
5    }
6    /**
7     * @.pre true
```

```
8    * @.post RESULT == (Palauttaa tilanneolion)
9    */
10   @Override
11   public OOMTilanne annaOOMTilanne() {
12     return opiskelija.annaOOMTilanne();
13   }
14   /**
15    * Asettaa opiskelijan nimen ja op. numeron
16    *
17    * @.pre nimi != null && opNumero>0
18    * @.post (nimi & op.numero asetettu)
19    **/
20   @Override
21   public void asetaNimiJaOpNumero(String nimi, int opNumero) {
22     opiskelija.asetaNimiJaOpNumero(nimi, opNumero);
23   }
24   /**
25    * @.pre true
26    * @.post annaOOMTilanne().ilmoittautunut == true && annaOOMTilanne().hereillä
27    *         == false && annaOOMTilanne().luennolla == false &&
28    *         (OLD(annaOOMTilanne().ilmoittautunut) || Maailma.tuskanMäärä ==
29    *         OLD(Maailma.tuskanMäärä) + 1000)
30    */
31   @Override
32   public void ilmoittauduOOMKurssille() {
33     opiskelija.ilmoittauduOOMKurssille();
34   }
35   /**
36    * @.pre annaOOMTilanne().ilmoittautunut == true && annaOOMTilanne().hereillä ==
37    *        false
38    * @.post Maailma.tuskanMäärä == OLD(Maailma.tuskanMäärä) + 10 &&
39    *         annaOOMTilanne().luennolla == true
40    */
41   @Override
42   public void osallistuLuennolle() {
43     opiskelija.osallistuLuennolle();
44   }
45   /**
46    * @.pre 0     aikaaLuennonAlusta < 90
47    * @.post annaOOMTilanne().hereillä == true
48    */
```

```
49    @Override
50    public void herää(int aikaaLuennonAlusta) {
51      opiskelija.herää(aikaaLuennonAlusta);
52    }
53    /**
54     * @.pre annaOOMTilanne().luennolla == true
55     * @.post annaOOMTilanne().luennolla == false && annaOOMTilanne().hereillä ==
56     *        false
57     */
58    @Override
59    public void poistuLuennolta() {
60      opiskelija.poistuLuennolta();
61    }
62    /**
63     * @.pre annaOOMTilanne().ilmoittautunut == true && annaOOMTilanne().hereillä ==
64     *       true
65     * @.post Maailma.tuskanMäärä == OLD(tuskanMäärä) + 90 -
66     *        hereilläAlkaenMinuutista && annaOOMTilanne().hereillä == true
67     */
68    @Override
69    public void vastaaKysymykseen(int aikaaLuennonAlusta) {
70      opiskelija.vastaaKysymykseen(aikaaLuennonAlusta);
71    }
72  }
```

4.2 OpiskelijaAllas-luokka (kohta c)

Kohdassa c pyydetään toteuttamaan luokka, jonka avulla kierrätetään opiskelijoita, jottei kaikkien tarvitse ilmoittautua tälle tuskaa tuottavalle kurssille.

OpiskelijaAllas-luokka

```
1   public class OpiskelijaAllas {
2     private static Opiskelija[] opiskelijat = new Opiskelija[0];
3     /**
4      * Metodi joka luo uusia opiskelijoita vain tarpeen vaatiessa.
5      * @param uhriMäärä
6      * @param vuosi
```

```
7     */
8    public static void hankiOpiskelijat(int uhriMäärä, int vuosi) {
9      Opiskelija[] uusiAllas = new Opiskelija[uhriMäärä];
10     if(uhriMäärä > opiskelijat.length) {
11       for(int i = 0; i < opiskelijat.length; i++) {
12         uusiAllas[i] = opiskelijat[i];
13       }
14       for(int i = opiskelijat.length; i < uhriMäärä; i++) {
15         uusiAllas[i] = new ValeasuOpiskelija(new TavallinenOpiskelija("Uniikki␣Lumihiutale", 1234567
16       }
17     }
18     else {
19       for(int i = 0; i < uhriMäärä; i++) {
20         uusiAllas[i] = opiskelijat[i];
21       }
22     }
23     for (int i = 0; i < uhriMäärä; i++) {
24       uusiAllas[i].asetaNimiJaOpNumero("Uniikki␣Lumihiutale", 1234567 + i + vuosi);
25     }
26     opiskelijat = uusiAllas;
27   }
28   /**
29    * Metodi joka luo aina uudet opiskelijat.
30    * @param uhriMäärä
31    */
32   public static void hankiOpiskelijat(int uhriMäärä) {
33     Opiskelija[] uusiAllas = new Opiskelija[uhriMäärä];
34     for (int i = 0; i<uhriMäärä; i++)
35     uusiAllas[i] = new TavallinenOpiskelija("Uniikki␣Lumihiutale", 1234567+i, true);
36     opiskelijat = uusiAllas;
37   }
38   /**
39    * Palauttaa opiskelijat.
40    * @return
41    */
42   public static Opiskelija[] opiskelijat() {
43     return opiskelijat;
44   }
45 }
```

4.3 Opiskelija testi (kohta b ja d)

Kohdassa b testataan luokkien eroavaisuuksia. Opiskelijoita 100 per vuosi.

Valeasulla pienenevä tuska.

TavallinenOpiskelija-luokalla

```
Tuska yhteensä: 117840
```

ValeasuOpiskelija-luokalla

```
Tuska yhteensä: 108840
```

Altailla pienenevä tuska.

Ilman altaita

```
Vuoden 1 tuska: 108840
Vuoden 2 tuska: 108840
Vuoden 3 tuska: 108840
Vuoden 4 tuska: 108840
Vuoden 5 tuska: 108840
Tuska yhteensä: 544200
```

Altailla

```
Vuoden 1 tuska: 108840
Vuoden 2 tuska: 8840
Vuoden 3 tuska: 8840
Vuoden 4 tuska: 8840
Vuoden 5 tuska: 8840
Tuska yhteensä: 144200
```

ValeasuOpiskelija-allas luo vähiten tuskaa eli koodi toimii.

Tekijänä Santeri Loitomaa (516587)