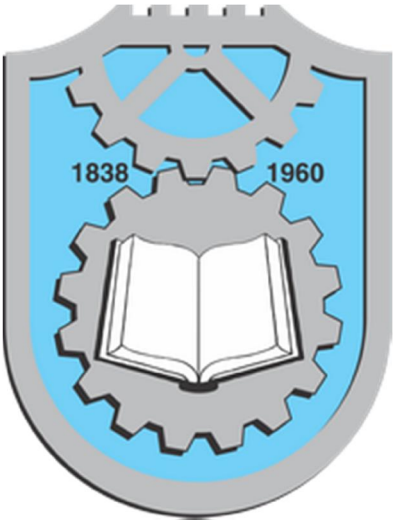
Универзитет у Крагујевцу

Факултет инжењерских наука



Пројектни рад

Предмет:

Софтверско инжењерство

Тема:

Графичка симулација Соларног система

Студент:

Никола Самарџић 562/2015

Професор:

Ненад Филиповић

Тијана Шуштершич

Садржај

Поставка задатка ..... - 3 -
Опис изворног кода ..... - 4 -
Класа *Планета* ..... - 4 -
Класа *Соларни систем* ..... - 5 -
UML дијаграми ..... - 10 -
Use Case Diagram ..... - 10 -
Дијаграм секвенци ..... - 11 -
Дијаграм активности ..... - 12 -
Дијаграм стања ..... - 13 -
Дијаграм класа ..... - 14 -
Литература ..... - 15 -

Поставка задатка

Пројектни задак је графичка репрезентација Соларног система. Дакле, све планете треба да буду у реалном односу величина и брзина. Када су у питању полупречници планета, направљен је компромис, јер реалне величине излазе ван оквира рачунарског монитора. Нпр. Јупитер је највећа планета у Сунчевом систему, а 100 пута је мањег пречника од Сунца, што би значило да за *реалну* величину Сунца, Јупитер се не би видео на екрану, јер би био сувише удаљен, а за мању величину Сунца, 100 пута мањи Јупитер се не би ни видео, а самим тим ни остале планете. Одабрана је оптимална вредност полупречника Сунца и на основу исте су скалирани полупречници осталих планета. Брзине су у реалним односима, док су растојања измењена због ограничења монитора (растојање између Марса и Јупитера је такође огромна - 550 390 000 км, док је растојање између Земље и Марса 78 340 000). Величине су дате у следећој табели:

Планета	Пречник екватора	Маса	Велика полуоса	Орбитални период	Период ротације
Меркур	0,382	0,06	0,38	0,241	58,6
Венера	0,949	0,82	0,72	0,615	243
Земља	1,00	1,00	1,00	1,00	1,00
Марс	0,53	0,11	1,52	1,88	1,03
Јупитер	11,2	318	5,20	11,86	0,414
Сатурн	9,41	95	9,54	29,46	0,426
Уран	3,98	14,6	19,22	84,01	0,718
Нептун	3,81	17,2	30,06	164,79	0,671

Корисник може мануелно да мења брзине планета, уз помоћ *user-friendly* графичког интерфејса. Једноставно, из опадајуће листе, изабере жељену планету, затим у поље текста унесе брзину, кликом на једно дугме ажурира систем, кликом на друго дугме добија репрезентацију наведеног система. Звезде су обичне беле тачке пречника 1 пиксел, које су псеудо-случајно *разбацане* по екрану. Пројектни задатак је имплементиран у Јава програмском језику, коришћењем *java-swing* библиотеке за ГУИ.

Опис изворног кода

Централна класе пројекта су класе:

- Соларни систем, и
- Планета

Класа *Планета*

Коришћењем ове класе, апликација конструише и исцртава планете. Све планете имају своје пречнике, почетну позицију, као и брзину (која заправо представља угао померања у кружном кретању, што је тај угао већи, то је планета бржа). Конструктор класе:

```
5  public class Planeta {
6
7      private int basex, basey, x, y, R;
8      private double angle, d, increment;
9      Image img;
10
11
12     public Planeta(int basex, int basey, int R, Image img,double angle) {
13         this.basex = basex;
14         this.basey = basey;
15         this.R = R;
16
17         this.angle = angle;
18         this.increment = angle;
19         this.img = img;
20         this.x = basex;
21         this.y = basey;
22
23     }
24
```

Уз помоћ промењивих *x* и *y* ажурира се положај планете, у почетном тренутку те координате су, редом, *basex* и *basey. Increment* се у каснијем делу кода користи за инкрементирање угла у кружном кретању планете, и та вредност је константна, за разлику од координата.

Поред типичних *gettera* и *settera,* битно је напоменути још две функције, за цртање планете и кружно кретање исте.

```
55     public void draw(Graphics g)
56     {
57         g.drawImage(img, x, y, null);
58     }
59
```

Функција *drawImage* узима као аргументе слику и тренутне координате планете, док је *Observer* постављен на *NULL.*

Функција *update* је комплекснија.

```
60      public void update(int SunceX, int SunceY, int SunceR) {
61          angle += increment;
62          d = Math.sqrt((SunceX - basex)*(SunceX - basex) + (SunceY - basey)*(SunceY - basey));
63
64          x  = (int)(SunceX + d*Math.cos(angle)+SunceR/2);
65          y  = (int)(SunceY - d*Math.sin(angle)+SunceR/2);
66          //repaint();
67      }
68
69
70  }
```

Аргументи функције су координате Сунца, и полупречник наведеног небеског тела. У свакој итерацији, угао се повећава за инкремент, треба напоменути да су у почетном трернутку ове промењиве једнаке. Планете се кружно крећу око Сунца, те се тражи растојање дате планете од Сунца, и координате се ажурирају математичком формулом за кружно кретање. Треба напоменути, формула је измењена итеративном, експерименталном методом. Овај део кода је представљао енигму кроз дужи период рада, најпре, инкрементирањем само координата, путања није круг, већ ромб. Једначини је придодат сабирак полупречника Сунца, како би се планета кретала у односу на „површину“ Сунца (тј. кружницу), а не у односу на сам центар.

Треба напоменути, постоји још једна класа, под називом Сунце. Заправо, она је иста као Планета, осим што нема последњу наведену функцију *update,* јер се Сунце не креће (осим око своје осе).

Класа *Соларни систем*

У овој класи све долази на своје место, буквално и фигуративно. Дакле, у овој класи убацују се сви параметри планета, као и слике, у бесконачној петљи се непрестано ажурирају координате сваке планете.

Коишћена су два низа, један за слике планета, а друга за саме планете. Дакле, један низ садржи типове података *img,* док други садржи инстанце класа Планета.

```
17      Model model;
18      Sunce s;
19      Planeta[] Planete = new Planeta[8];
20      Image[] img = new Image[9];
21      double vMerkur, vVenera, vZemlja, vMars, vJupiter, vSaturn, vUran, vNeptun;
```

Наведена класа има два конструктора:

- један који не прима аргументе, користи се за подразумевана величине и параметре, и
- други који прима аргументе, користи се за мануелно уношење параметара

Први конструктор:

```
24    public SolarniSistem() {
25
26
27        model = new Model();
28        model.setPreferredSize(new Dimension(1680, 1050));
29        add(model);
30
31        try {
32            img[0] = ImageIO.read(new File("sunce.jpg"));
33            img[1] = ImageIO.read(new File("merkur.png"));
34            img[2] = ImageIO.read(new File("venera.png"));
35            img[3] = ImageIO.read(new File("zemlja.png"));
36            img[4] = ImageIO.read(new File("mars.png"));
37            img[5] = ImageIO.read(new File("jupiter.png"));
38            img[6] = ImageIO.read(new File("saturn.png"));
39            img[7] = ImageIO.read(new File("uran.png"));
40            img[8] = ImageIO.read(new File("neptun.png"));
41
42        } catch (IOException e) {
43            // TODO Auto-generated catch block
44            e.printStackTrace();
45        }
46
47        s = new Sunce(0, 600, 100, img[0]); //Sunce
48
49        Planete[0] = new Planeta(64, 664, 3, img[1], 0.1);   //Merkur
50        Planete[1] = new Planeta(114, 714, 9, img[2], 0.04); //Venera
51        Planete[2] = new Planeta(170, 770, 10, img[3],0.025); //Zemlja
52        Planete[3] = new Planeta(227, 827, 5, img[4], 0.014); //Mars
53        Planete[4] = new Planeta(454, 1054, 30, img[5], 0.002); //Jupiter
54        Planete[5] = new Planeta(600, 1200, 25, img[6], 0.0008); //Saturn
55        Planete[6] = new Planeta(800, 1400, 15, img[7],  0.0002); //Uran
56        Planete[7] = new Planeta(1000, 1600, 20, img[8],  0.0001);//Neptun
57
```

```
61        setBackground(Color.BLACK);
62
63        Thread thread =  new Thread() {
64
65            @Override
66             public void run() {
67                Petlja();
68             }
69          };
70
71          thread.start();
72
73
74
75    };
```

Дакле, модел је постављен на највећу величину, 1680 пиксела x 1050 пиксела. Између линија 31 и 45, учитавају се слике, уколико дође до грешке, „хвата“ се изузетак. На линији 47, конструише се Сунце, са својим координатама и одгаварајућом сликком, касније, између линија 49 и 56, иницијалиују се планете, у одговарајућем редоследу. Може се приметити, да је Меркур „најбржа“ планета, из разлога што је најближа Сунцу, те на њу делује највећа гравитациона сила. За пречник Сунца, узето је 100 пиксела, за нпр. Јупитер, пречник је 30. Дакле, око 3 пута мање, што, наравно одступа од правог односа, јер је Сунце 100 пута веће. Такве величине су узете ради оприналности, што је већ раније наведено.

Линија 61 поставља црну позадину на модел, док се у помоћ нити извршава бесконачна петља, о њој више касније.

Други конструктор:

```
77  public SolarniSistem(double vMerkur, double vVenera, double vZemlja, double vMars, double vJupiter, double vSaturn, double vUran, double vNeptun) {
78
79      this.vMerkur = vMerkur;
80      this.vVenera = vVenera;
81      this.vZemlja = vZemlja;
82      this.vMars = vMars;
83      this.vJupiter = vJupiter;
84      this.vSaturn = vSaturn;
85      this.vUran = vUran;
86      this.vNeptun = vNeptun;
87
```

Дати конструктор прима аргументе, ти аргументи се уносе са тастатуре, преко графичког интерфејса. Промењиве са слике, су заправо брзине, тј. инкременти, који су раније објашњени.

```
139     public void Petlja() {
140
141           while (true) {
142
143
144              for(int i = 0; i <= Planete.length-1; i++)
145                 {
146
147                  Planete[i].update(s.getX(), s.getY(), s.getR());
148
149                 }
150
151
152              repaint();
153
154              try {
155                 Thread.sleep(DELAY);
156              } catch (InterruptedException ex) { }
157           }
158        }
159
```

Функција петља, је заправо бесконачна петља, у којој се константно ажурирају координате планета. Линија 152 је задужена за поновно цртање планета (тј. свих објеката у *JFrame-y).*

```
160    class Model extends JPanel {
161
162        private static final long serialVersionUID = 1L;
163
164        public Model() {
165
166             setFocusable(true);
167             requestFocus();
168          }
169
170        public void paintComponent(Graphics g) {
171
172            s.draw(g);
173
174            for(Planeta p: Planete) {
175                p.draw(g);
176            }
177
178            //zvezde
179
180
181            for(int count=0;count<=1000;count++) {
182             g.setColor(Color.WHITE);
183
184             g.drawOval(50*count,100*count,1,1);
185            g.drawOval(75*count,100*count,1,1);
186
187             g.drawOval(100*count,200*count,1,1);
188             g.drawOval(150*count,200*count,1,1);
189             g.drawOval(200*count,200*count,1,1);
190             g.drawOval(250*count,200*count,1,1);
191             g.drawOval(300*count,200*count,1,1);
192             g.drawOval(350*count,200*count,1,1);
193             g.drawOval(400*count,100*count,1,1);
194             g.drawOval(450*count,100*count,1,1);
195             g.drawOval(500*count,100*count,1,1);
196             g.drawOval(550*count,300*count,1,1);
197             g.drawOval(600*count,300*count,1,1);
198             g.drawOval(700*count,300*count,1,1);
199             g.drawOval(800*count,300*count,1,1);
200             g.drawOval(900*count,300-count,1,1);
201             g.drawOval(1000*count,300-count,1,1);
202
203
204
205             }
206        }
207
```

Класа Модел је заправо прва класа која се позива, још у конструктору Соларног система. Она је задужена за исцртававање Планета и Сунца (линије 172 до 177). Уз помоћ *for* петље се цртају „звезде“ (на псеудо случајним позицијама). До позива функције *Петља* на

екрану ће бити исцртане само звезде, јер су све инстанце класе иницијализоване на *NULL.* Међутим, све се дешава таквом брзином да је голим оком то неприметно.

За крај, цела класа изгледа овако:

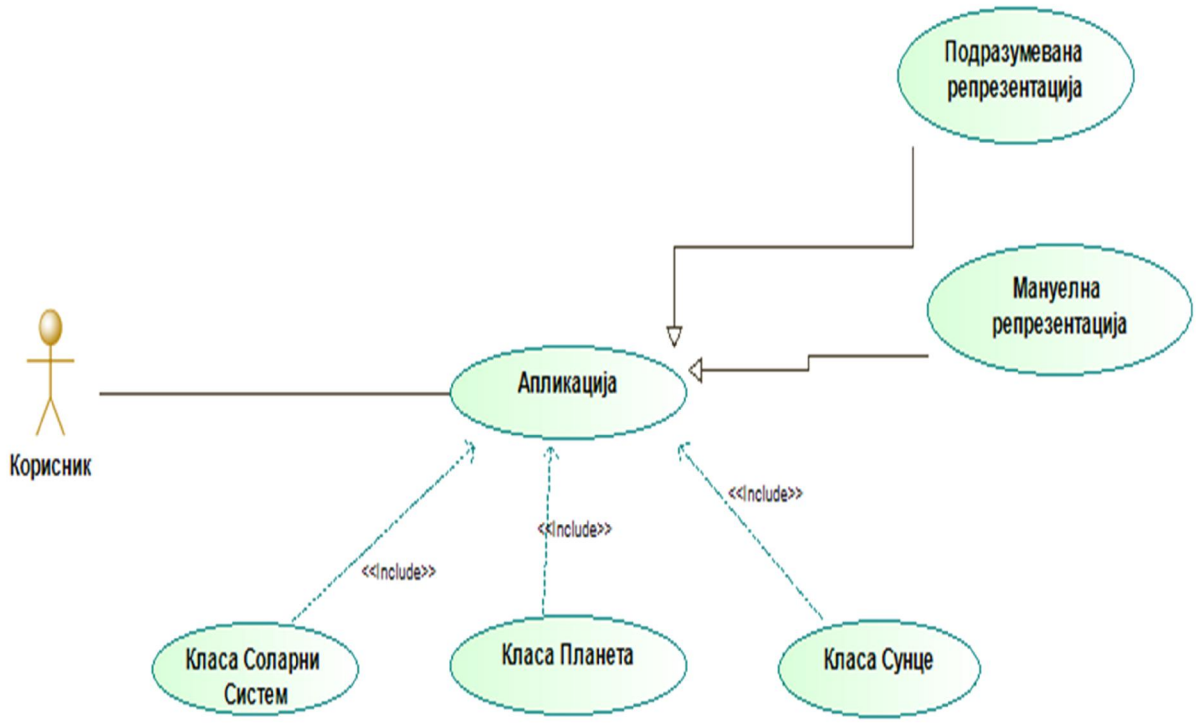
```
1  import java.awt.*;
6
7  public class SolarniSistem extends JPanel {
8
9      /**
10     *
11     */
12     private static final long serialVersionUID = 1L;
13     final static int DELAY = 50;
14     double size = 1;
15
16
17     Model model;
18     Sunce s;
19     Planeta[] Planete = new Planeta[8];
20     Image[] img = new Image[9];
21     double vMerkur, vVenera, vZemlja, vMars, vJupiter, vSaturn, vUran, vNeptun;
22
23
24     public SolarniSistem() {...
76
77     public SolarniSistem(double vMerkur, double vVenera, double vZemlja, double vMars, double vJupiter, double vSaturn, double vUran, double vNeptun) {...
138
139    public void Petlja() {...
159
160    class Model extends JPanel {...
208
209
210 }
211
```

UML дијаграми

Use Case Diagram

- Дијаграм случајева коришћења приказује скуп случајева коришћења и актера.
- Визуелизује понашање система, подстистема или интерфејса.

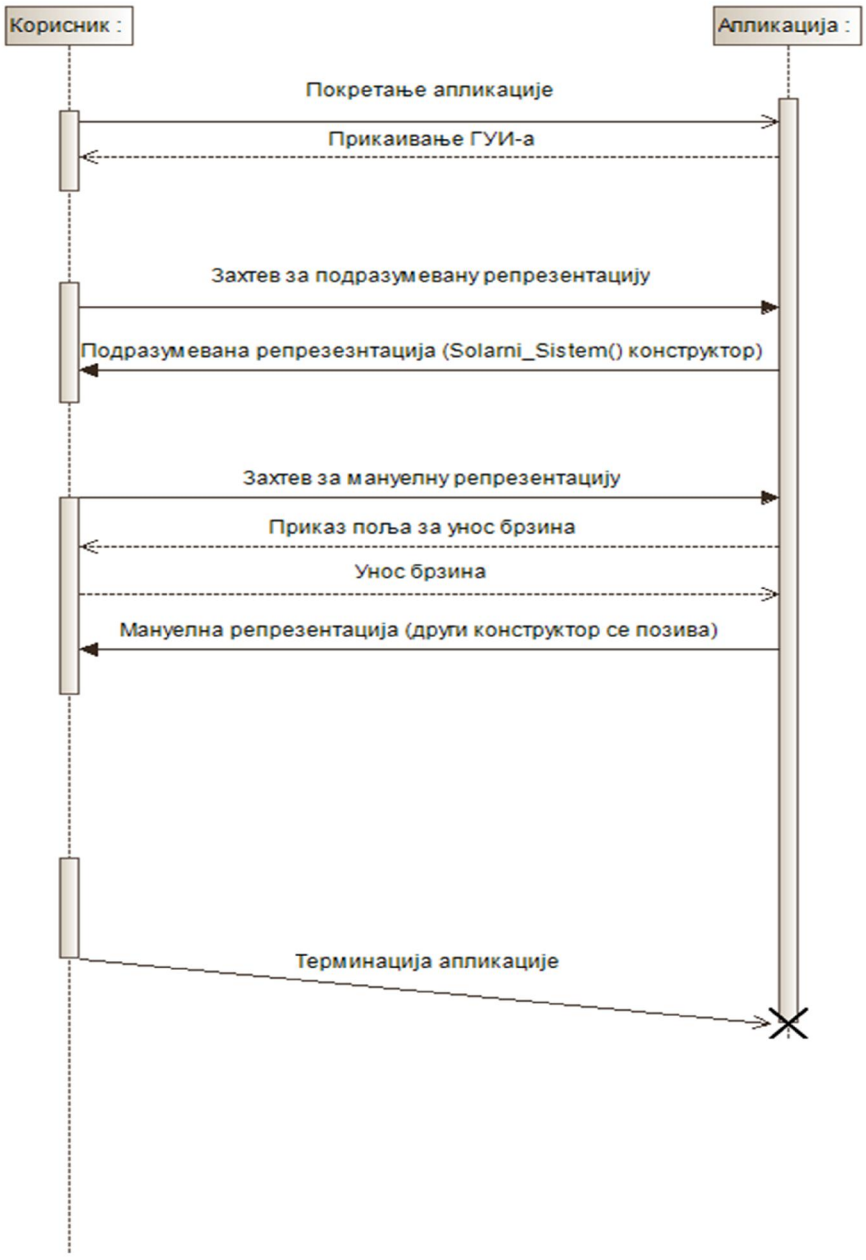
Дијаграм случајева, пројектован на датом задатку, приказан је на следећој слици.



Дијаграм секвенци

- Приказује комуникацију између скупа објеката, која се остварује порукама које објекти међусобно размењују у циљу остваривања очекиваног понашања.
- Детаљно описује како се операције изводе – које поруке се шаљу и када.
- Користи се за приказ једног или више сценарија.

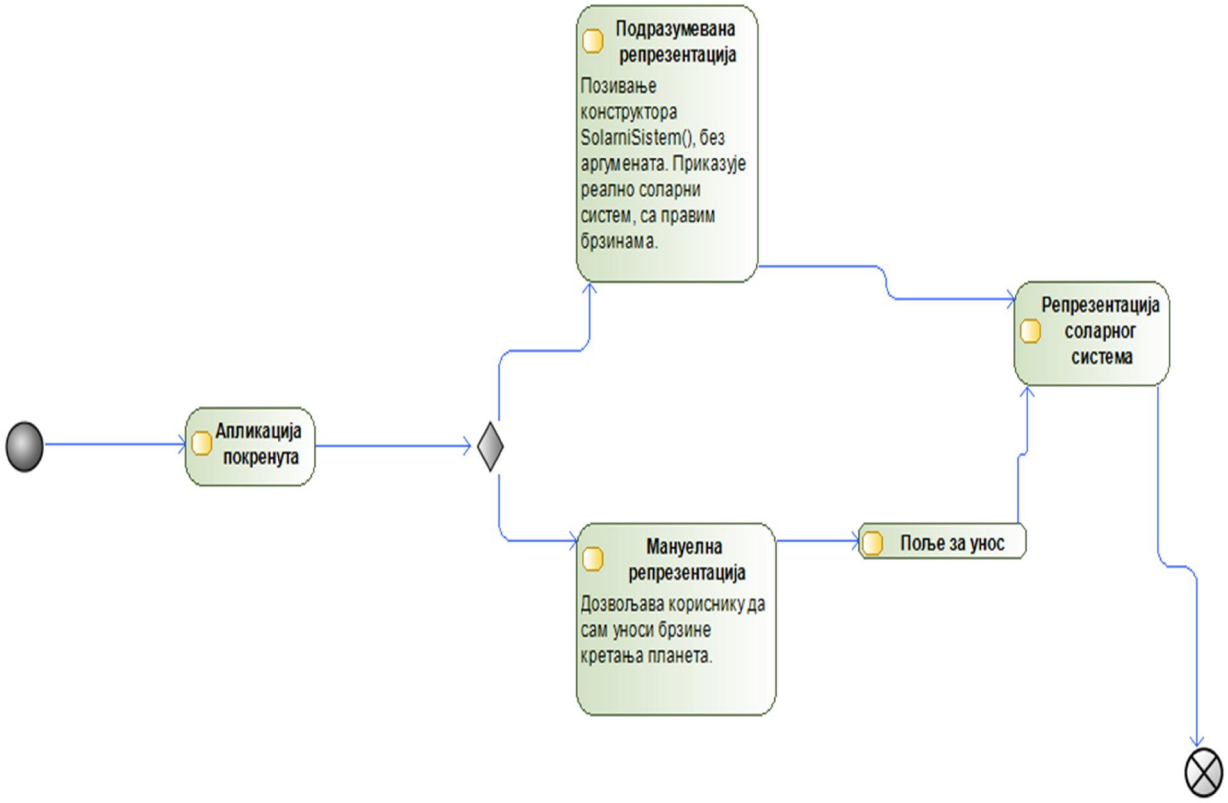
Дијаграм секвенци, за пројектни задатак, дат је на следећој слици.



Дијаграм активности

- Дијаграми активности су намењени моделирању динамичких понашања система
- Дијаграм активности приказује:
    1. Ток активности коју извршавају објекти
    2. Евентуално и ток објеката између корака активности

Дијаграм активности дат је на следећој слици.



Дијаграм стања

Аутомат стања:

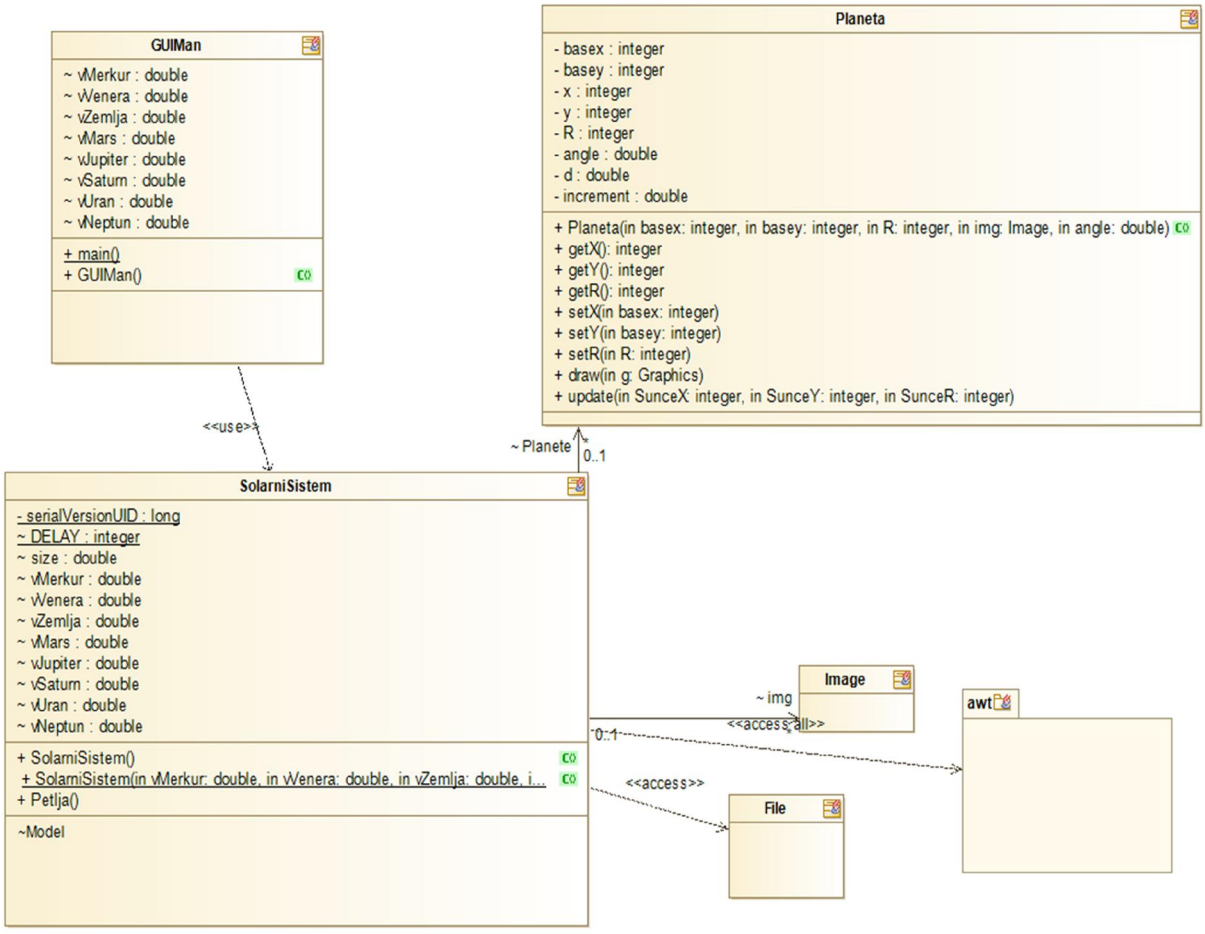
- Понашање које специфицира секвенце стања кроз која пролази.
- Моделира понашање неког ентитета или протокол интеракције.

Дијаграм стања је граф који приказује аутомат стања:

- Чворови су стања.
- Гране су прелази.

Дијаграм стања, за дати пројектни задатак, приказан је на слици.

Дијаграм класа



Литература

I. http://moodle.mfkg.rs/course/view.php?id=524
II. https://docs.oracle.com/javase/tutorial/uiswing/
III. https://www.eclipse.org/documentation/
IV. https://www.modelio.org/documentation-menu/user-manuals.html
V. https://sr.wikipedia.org/wiki/%D0%A1%D1%83%D0%BD%D1%87%D0%B5%D0%B2_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC