ΟΙΚΟΝΟΜΙΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ
ΤΜΗΜΑ : ΔΙΟΙΚΗΤΙΚΗΣ ΕΠΙΣΤΗΜΗΣ ΚΑΙ ΤΕΧΝΟΛΟΓΙΑΣ

ΕΙΔΙΚΑ ΘΕΜΑΤΑ ΤΕΧΝΟΛΟΓΙΑΣ ΚΑΙ ΛΟΓΙΣΜΙΚΟΥ

Διδάσκων: Δ. Σπινέλλης

Φοιτήτρια: Καζάκη Αργυρώ (8010044)

1. INTRODUCTION

1.1 A BRIEF PRESENTATION OF SIMULUS

Simulus is a star simulation application, designed mostly for amateurs interested in astronomy. It provides the user with a nice GUI (Graphical User Interface) presenting a simulation of a number of stars distributed in a sphere. The viewpoint circles around the sphere and creates the impression of a rotating sphere.

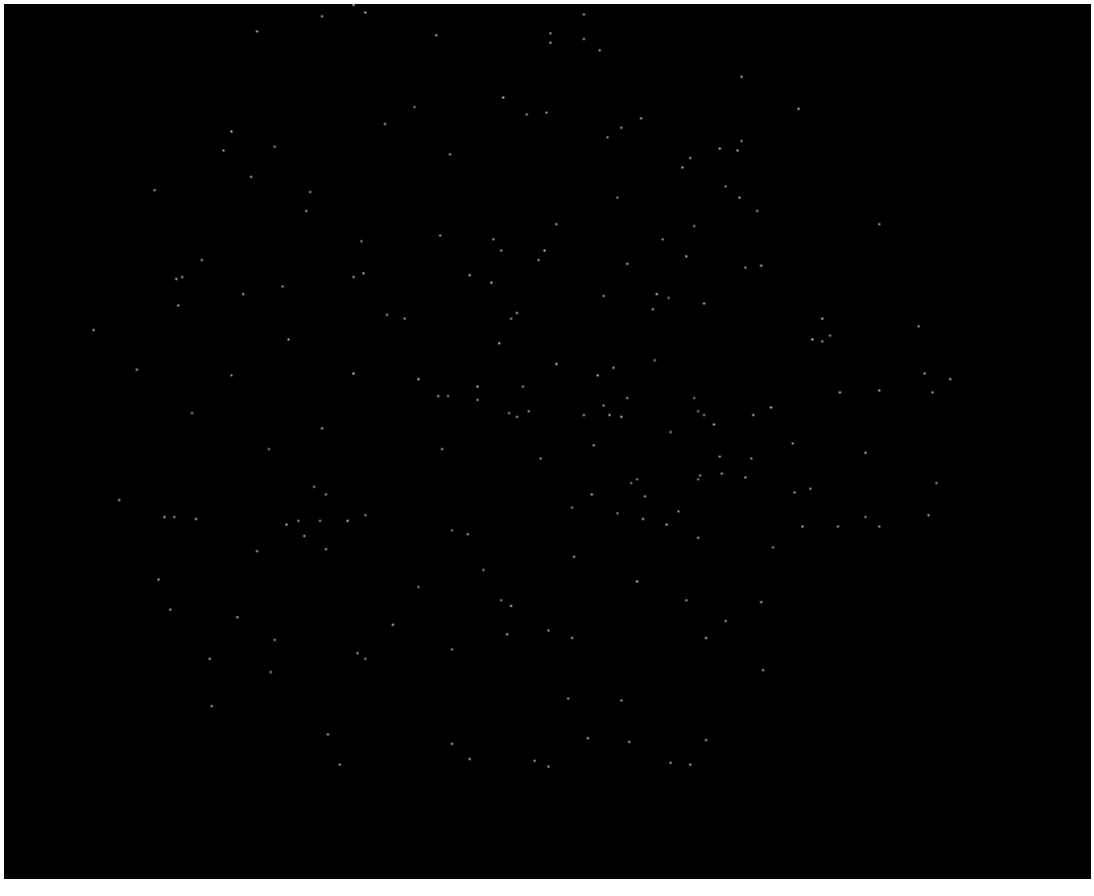

Figure 1

Figure 1 is a screenshot taken for the application (a bigger screenshot of the entire application would make the stars invisible). On the left side of the viewing screen, there is a menu for defining certain attributes like gravity, number of the stars studied, zoom, speed etc, which are essential for the star simulation. When the application runs for the first time, some default values are loaded which have been decided from the designer of Simulus. If the user presses the button marked as "Edit", is allowed to change any of these parameters. During this procedure the viewer stops, and once the new attributes have been set, it starts again loading the new values. Finally, when Simulus is been shut down by the user, the latest values set for the attributes are stored, consequently, simulation next time will begin with these values set.

1.2 TECHNICAL FEATURES

Simulus is entirely written in java and it's a platform independent application. For its graphical interface, designer/developer has used both java.awt and javax.swing packages.

Application, uses 8 main attributes to perform the simulation:

- gamma
- radius
- stars
- sensitivity
- snapshot
- movement
- deltat
- zoom

These are stored in a .properties file named mulumis.properties

Every time a change of those variables takes place, simulation stops, mulumus.properties is resaved. Actually, the .properties file is been reproduced form the beginning even if the smallest change occurs, and overrides the old one. And this happens also, when the application is loaded for the first time (with the default values of course).

2. ENHANCEMENTS

2.1 THE REASON WHY

Although Simulus can prove to be a quite useful tool for the amateurs that want to study star simulation, its design and structure reveals some drawbacks. The application stores and loads only one set of values per time, without giving the user the chance to save some data. Students or astronomers, while experimenting, tend to keep several notes on the observations made, and more often than not, go back on previous experiments to make comparisons. Unfortunately, Simulus lacks these potentials, so in my opinion, a few enhancements should be made, in order to make the application user-friendly or client-based (either way). So, during this project I managed to implement the following features.

2.2 SAVE PARAMETERS

The first thing I changed on Simulus was while shutdown when I asked for the default values to be restored, no matter the changes, so that the application always starts loading its default values. The designer had

already implemented the method I needed, so the only thing I did was just adding the following line:

simulumProperties.resetToDefaultValues();

Furthermore a JButton labeled as "Save" has been added to the application frame (with extra attention to the layout). Using this option, the user is given the possibility to save the set of values for the attributes of the star simulation added. So later on he can refer back to it, without having to type them again. A new .properties file named after the current day and time is been created, storing these values.

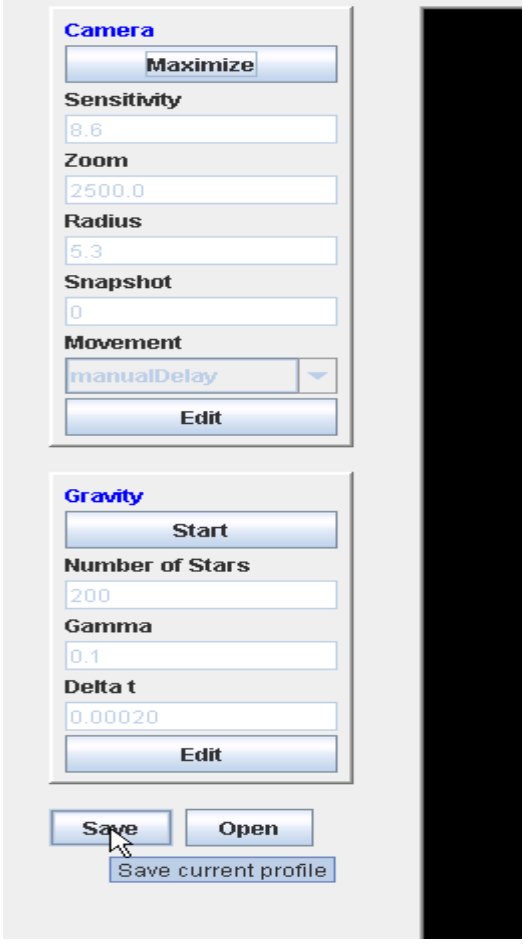

Figure 2

2.3 EXPORT PARAMETERS

Previously, when referring to user-friendly applications, I meant that a developer must try to make things for him as easier as possible and satisfy his demands. From this point of view, think the application simultaneously I realized that saving the parameters studied is indeed very helpful, but not the ideal if the user is performing an essay. In that case having written down every experiment he made is essential for him, so the solution I came up with was exporting the parameters saved to a .txt file. So, once the parameters are saved, a pop up dialog box appears, asking the user if he wishes to export this data.

Figure 3

If the user wishes so, a .txt file is created containing the parameters saved. After some studying the Java API, I realized that exporting data may sound like a difficult or complex process, but is indeed a few well organized lines of code. Mine was:

```
File file = new File(getParams("exportPath") );
FileWriter fw=null;
      try {
         fw =new FileWriter(file);
      } catch (IOException fnf){
         System.out.println("Error opening file: " + fnf);
      }
      BufferedWriter bf = new BufferedWriter(fw);
       try {
     bf.write("New Simulum Properties Profile : " + dateModified);
```

…. and so on. At this point, I will have to remind that I am not letting the user to decide the path that the exported file will be located. I have chosen the path and a standard name for it and placed it in a separate properties file. Of course the user can later change all these and it would be fine by me.

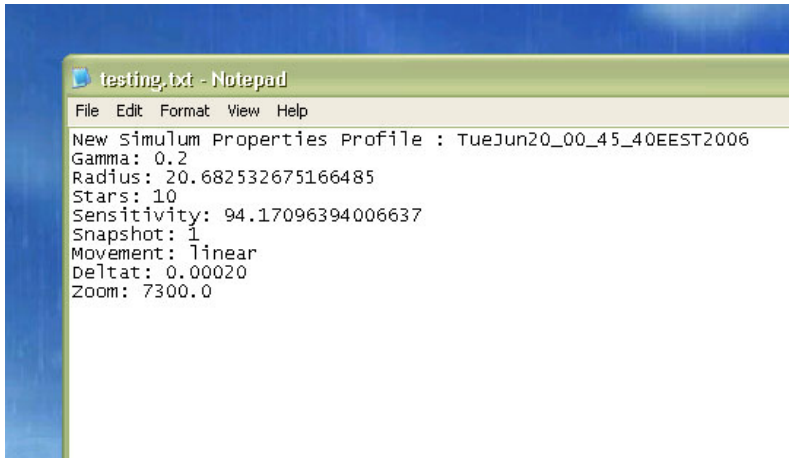

Figure 4

2.4 OPEN PARAMETERS

So, the user is given the possibility to store the attribute values of his experiments, so he must be given a way to use these. A new JButton marked as "Open" was placed next to the "Save" one.

Figure 5

So, the user presses the "Open" button. A JFileChooser appears on screen, and asks for the path of the property file. Once the user provides the application with the path, the attributes are loaded and the simulation is on again.

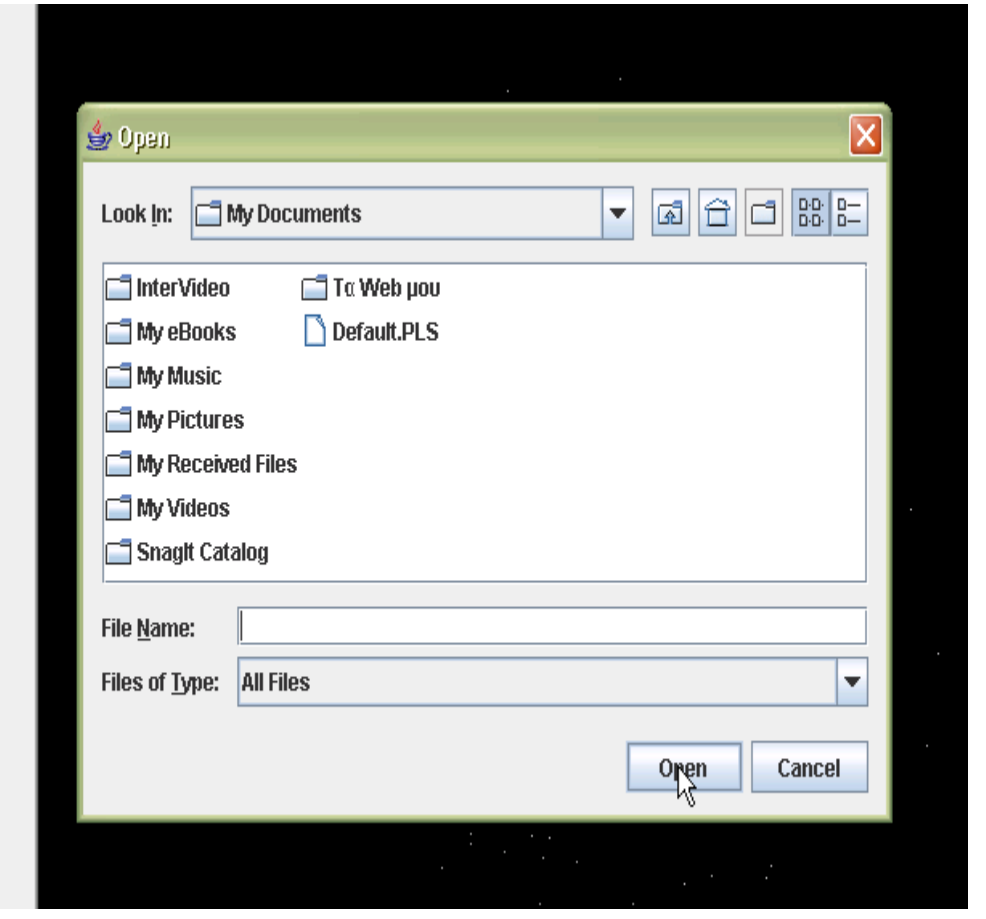

Figure 5

The code used for opening the property file was a small collection of **getParams()** methods which reads the parameter values from the property file. This time to arguments where given: the parameter name, and the path for the .properties file (as collected from the JFileChooser using **getAbsolutePath()** method.

2.5 PRINT PARAMETERS

Finally printing the set of values, saved and exported, through the application, would be ideal for the potential user. The following class is an example of how a printing option could be implemented. It is not currently bound to the rest of the source code and possibly needs some debugging, as I am working at home and I don't own a printer to test this. Compiling it though, seems ok.

```
public class PrintParameters {

    private DocPrintJob job = null;
    private PrintRequestAttributeSet pras = null;
    private DocFlavor flavor = null;
    private PrintService service = null;
    private PrintJobListener pjlistener = null;


    /** Creates a new instance of PrintParameters */
    public PrintParameters() {
        flavor = DocFlavor.INPUT_STREAM.AUTOSENSE;

        pras = new HashPrintRequestAttributeSet();

        PrintService printService[] =
                PrintServiceLookup.lookupPrintServices(null,null);
```

```
PrintService defaultService =
            PrintServiceLookup.lookupDefaultPrintService();

    service = ServiceUI.printDialog(null, 200, 200,
            printService, defaultService, flavor, pras);

    pjlistener = new PrintJobAdapter() {
       public void printDataTransferCompleted(PrintJobEvent e) {
          System.out.println("Printing completed");
       }

       public void printJobCompleted(PrintJobEvent pje) {

       }

    };
  }


  public void print(FileInputStream fis) {

     DocAttributeSet das = new HashDocAttributeSet();
     Doc doc = new SimpleDoc(fis, flavor, das);


 if (service != null) {
```

```
            job = service.createPrintJob();

            job.addPrintJobListener(pjlistener);
        }

        try {
            job.print(doc, pras);
        } catch (PrintException pe) {
            pe.printStackTrace();
        }
    }
}
```

3. NOTES

Reading the existing code and understanding the way things work with Simulus proved to be quite interesting and easy. The API provided along with the application contributed to this fact.

I didn't cooperate with the other members of the team that are currently occupied with Simulus. I believe now that I could have made a better choice concerning the project of the essay. Simulus (though fun and interesting) hardly qualifies as popular or even good application. I wasn't even sure if the team needed those changes, as they were based on my personal opinion and point of view. Maybe the original designer had something else in mind.