CSCI205 | Spring 2017

SOLAR SYSTEM SIMULATOR

Design Manual

Authors:

Gabe Gomez

John Venditti

Zach Brill

Tyler DiBartolo

Our program is a simulation of the Solar System complete with accurate scaling based off the standard units used within the scientific community. We completed this task using solely JavaFX. Our main display is shown in a Borderpane to hold the entirety of our project. We then add another Borderpane and a Split Pane. The second borderpane holds the actual depiction of our simulation and is the size of the entire first borderpane. With this set up we are able to put the split pane holding our text with information on the planets over part of the simulation with opacity that is more aesthetically pleasing. Our simulation consists of spheres from each planet and the sun. There are elliptical paths created and drawn on this borderpane to show the path that the planets take around the sun. Each path is used to then create a path transition, an object in JavaFX, whose node is the sphere of a planet. These nodes traverse the path creating the visual effect of orbiting planets. Many of our classes hold more information than they have methods. This is due to our Algorithms class that acts as our calculator to be used by classes such as Planet, Sun, and Solar System. With this type of design it became increasingly easier to add to our simulation due to the distinct and clear partitions of the code.

User Stories

We began early on understanding that we wanted our Solar System to show the planet's motions as they moved through through their orbits. We were also a bit apprehensive as to whether or not we could fit the planets all on one screen, which we knew would throw the idea of scale out the window. However, within a few days we agreed as a team that we wanted our simulator to be as accurate as possible, so with this in mind we began writing software that would attempt to be remain true to the size of our solar system. This meant implementing zooming capabilities was a key user story throughout our development process.

Another very important user story to us was displaying the planets in an eye-catching way. To ensure this, we learned a good bit about the Sphere class in JavaFX, and how to wrap an image around these spheres using a Material. This enabled us to display a surface mapping image of each planet around the sphere object that represented the planet. However, when doing this we realized the entire surface of the planet was illuminated. This gave a false visual of the day and night cycles on each planet. For this, we had to develop a new key user story...lighting.

After a bit of Googling, our group was able to find an extremely useful JavaFX object. PointLight. This allows the programmer to specify a light source within the scene. From here, we simply construct the PointLight to use the Sun's current position in the frame as the location of the light source. Once this is completed, the planets become shaded based on what side is facing the sun. As the planets rotate through their orbits, only the side adjacent to the Sun can be seen. The other half, is dark, as it would be in real life. This is our best attempt at representing what each planet looks like during the daytime.

Object Oriented Design

The design of our project revolves around the objects we created such as our Sun and Planet class. These two classes hold information along with important methods needed. Our primary object were our Planet objects we created. These objects call on our Constants class which hold all the numbers that will not be changed such as mass or radii of planets. Being able to get its information from the Constants class, the Planet class constructor, when called, is much cleaner and isn't a jumble of ugly looking numbers. Instead it is just the name of the planet with the constant required following. Creating the Constants class keeps all of our vital information in one place that is so easy to access and makes

changing those constants, if the need arises, even easier. Our Planet object was a parent class for a couple of our other classes. Both Sun and Moon inherited from Planet and had their own distinctions. This became useful when instantiating a Sun and producing the graphic sphere because we could use the same methods as a planet just without the path. Having inheritance on classes that are similar was one of the ways we used object oriented design to our advantage.

We began our implementation of our simulation by adding all the planets and planet paths for into the main. This worked when we were in our testing phase and gave us a visual, but the result was a cluttered and bogged down main function. We concluded that we needed to find a way that would draw our simulation without looking like a mess. In the end we added another class Solar System to hold all the planets and their paths. With that we added some methods to draw our solar system from the Solar System class creating a cleaner, more optimal code. With Solar System as a class we were then able to create an easier way of selecting planets with a simple method to search through our array of planets.

Our GUI implementation was a mix of JavaFX and Scene Builder. The main visual portion of the simulator is written entirely using JavaFX. The additional information displayed at the bottom of our GUI was created using Scene Builder. We found Scene Builder much simpler to use when it came to actually constructing the layout for this information pane. Getting things aligned and spaced correctly is much easier to pull off in Scene Builder, since it’s a much more visual approach to creating the GUI. With JavaFX, you have no idea what the code is actually creating until the program is run. With Scene Builder however, you have an in your face example of exactly how the FXML will render your GUI.