CSCI205 | Spring 2017

SOLAR SYSTEM SIMULATOR

User Manual

Authors:

Gabe Gomez

John Venditti

Zach Brill

Tyler DiBartolo

Our program attempts to create a scaled, virtual representation of the solar system. Our solar system comprises 9 planets, one massive star, and millions of other objects floating around in space. At the center of it all, is the Sun. The planets, starting with the closest to the Sun, are as follows...Mercury, Venus, Earth, Mars, Jupiter, Saturn, Uranus, Neptune, Pluto. These planets revolve around the Sun, in what are called orbits. The expansive nature of our solar system makes it extremely difficult to display all of these objects, given that they are incomprehensibly far away from each other. Our team wanted to develop some way to fit the entire solar system on a computer screen, but still keep the representation as accurate as possible. This of course presented us with plenty of challenges. The first order of business was to simply research the mathematics behind how our solar system operates. What makes the planets move the way they do? Is there some way to calculate how long it should take a planet to orbit the Sun, or is it arbitrary? What's the best way to render each planet, while remaining true to scale? These questions intrigued us, and spurred us on to create our Solar System Simulator.

Our group, Superfish LLC, felt inspired to create a piece of software that wasn't a game. Instead, we opted for something that felt more rewarding, and more satisfying to our curiosities. This led to us making a solar system simulator. We knew right off the bat, plenty of research had to be done. Within the first few days, we had amassed a research repertoire, stockpiling any equations we thought may be even the slightest bit relevant to our project. It quickly became clear what sort of "objects" our solar system would need. Planets, the Sun, orbits, and relevant information were all things that needed to be present. So, we began to create classes for each of these things. Each planet contained almost a dozen constants, some entered by hand, others

calculated within our Algorithms class. We learned what makes each planet's orbit so unique. Its Semi-Major and Semi-Minor axes. Each planet's mass plays a huge part in how the planet moves around its orbit. As a team, we began to understand the relationship between how far a planet is from the Sun, and how long it will take that planet to complete one orbit. This can be calculated simply using Kepler's Laws/Kepler's Equation. Kepler's equation states a very clear-cut way to calculate a planet's orbital period, based on its semi major axis. The period, 'P', can be found using this equation: $P^2 = A^3$, where A is the average distance the planet is from the Sun (Semi Major Axis). This orbital period is how long it takes each planet to orbit the Sun once, and it is described in Earth years. A planet with a period of 1 takes approximately 365 days to orbit its star. Calculating the orbital period of a planet is just one of many equations we had to code in our Algorithms class.

Once we were able to figure out the calculations behind the orbital period, we began to tackle the challenge of actually getting an object to behave with this orbit in mind...how could we create an orbit with code? Our team knew this would require some more research. A few key classes were found. Path, and PathTransition. These would enable us to represent these orbits within our code. To construct each planet's Path (orbit), our program requires information such as the planet's orbital eccentricity (how circular is it), the planet's period, and the planet's semi-major and semi-minor axes. With these created, we now had 'to-scale' orbital traces for every planet in the solar system. Next comes PathTransition, which allows for a node, in our case a planet, to be added to the Path. Once this is complete, we can control and render the planet actually moving through its orbit in the GUI. This is done with various PathTransition settings, including the duration of the animation, and at what speed the planet should travel. We begin to

see real progress in achieving a proper simulation of our solar system. However, one large problem still remains.

The orbits of each planet are much too far apart to show on the screen all at once, using a static camera view. In addition, some of the planets are simply too large to show on the screen, in comparison to some of the smaller, inner planets. This is where we introduced the ability for the user to zoom in and out of the frame, changing how much of the solar system they'd like to see. All of this is done through key handlers...if the user presses 'i' for instance, the main root pane will rescale its height and width, giving the effect of zooming in. Perhaps the user presses the 'o' key. This would again, rescale the pane to give the effect of zooming out. Through these various zooming features, we are able to display the solar system in its entirety. Everything is to scale, and the user is able to get a bird's eye view of how vast our solar system really is. When zoomed out enough to see Pluto's orbit, the planet itself will be almost invisible to the naked eye...while Jupiter and Saturn are plain as day. When seeing this, our team felt we truly accomplished our goal of displaying the solar system to scale, and impressing upon the user just how vast and large our neighboring celestial objects are.

To use our program, we simply require you to run the JavaFXMain.java file. Once the program has started, there are a few basic controls. They are as follows:

Key:

I - Zooms In

O - Zooms Out

R - Resets the frame so the Sun is at the center

W - Moves the frame up.

A - Moves the frame left.

S - Moves the frame down.

D - Moves the frame right.