Group 5

Conor Green

Dr. Asghari ENGR 160-03

Solar System Simulation Project

**I. Abstract**

The purpose of this report is to inform the reader on how the project is designed and the development process behind it. Originally, this project was intended to improve on the code given from last year by improving speed, allowing for customizable planets, displaying information on mouse hover, and/or adding in comets. However, after working through the code, my goals changed to create a program that runs any variable input faster than before. To accomplish this, a new GUI, new methods to attain user information, and a different driver were necessary.

**II. Introduction**

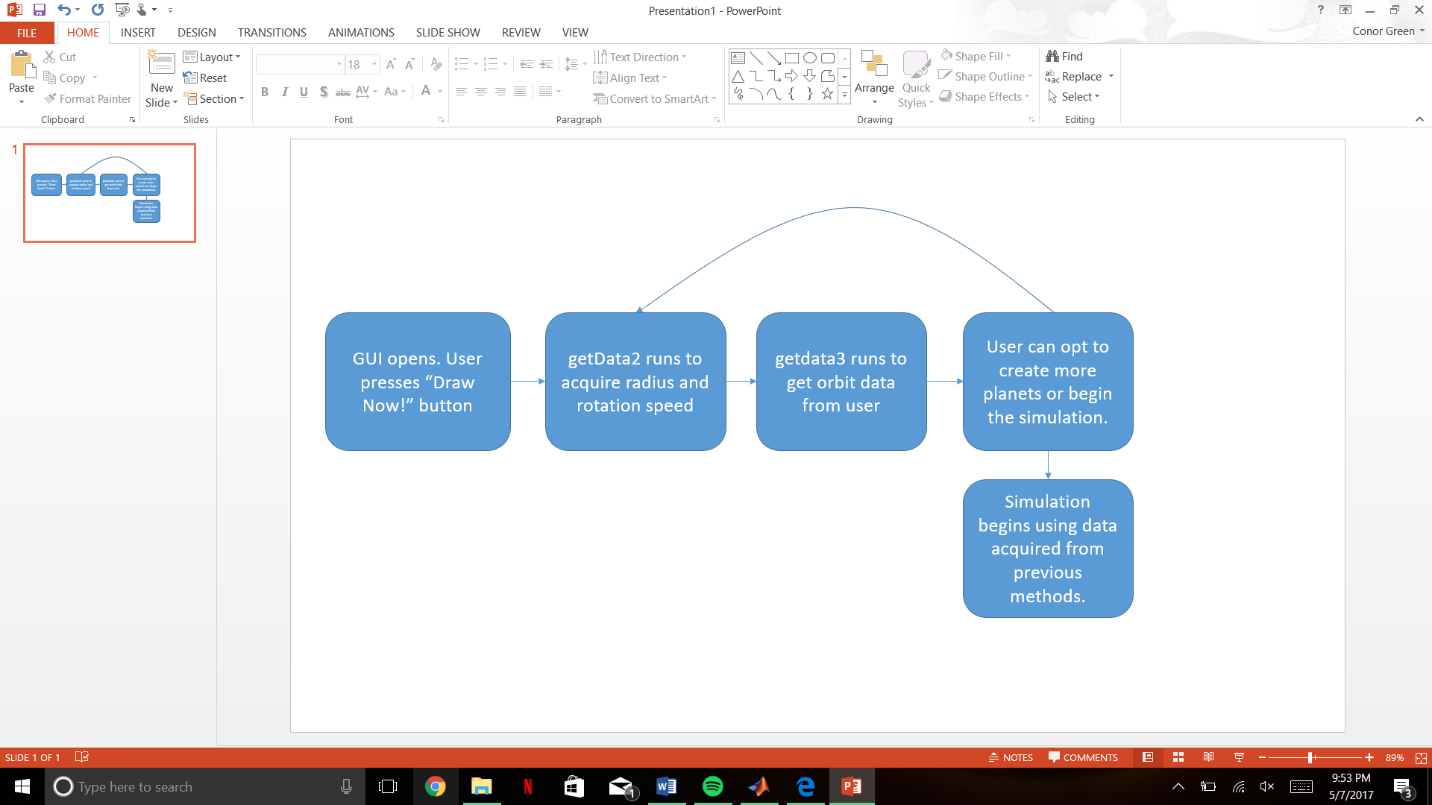
The code was given in a file, containing 119 Matlab files, 36 JPG images, and a Matlab figure, that ran by executing “Solar\_Sim\_GUI.” Upon execution, a figure pops up, with an axes (galaxy background image), three drop down menus for selecting the view, magnification, and speed, and button for re-orienting the view. To view the solar system, inner-solar system, the sun, or any of the eighth planets and their moons the user selects the desired view from the first drop down menu. The program then simulates the orbits in the axes. The user can also control the magnification by selecting “actual” or “magnified” from the second drop down menu and the user can control speed by selecting “slow,” “fast,” or “hyper-speed” from the third drop down menu.

The program ran by having a hierarchical organization that used multiple selector functions to take the users input and throw it to the right specific simulating function. The main running function is the GUI. The GUI initializes the figure and adds each file image into the handles structure. It defines the callback functions for all UIcontrols (3 drop down menus and a button). Each callback updates the handles structure, which holds all the graphical and user data, and then calls the function “Sim\_Selector,” passing the handles through. In “Sim\_Selector,” the program would use the value of the first drop down to go to another selector function (there is one for each view). In the next selector function, the function directs to the next function based on the second drop down menu’s value. After that it selects the right speed function. Finally, after going through three selector functions, the program ends up at the simulating functions (i.e. “Planet\_Mars\_A\_F” for Mars, actual size, fast speed). At this level, the program first sets values for center of orbit, radius of planet, orbit size, etc. Next, the simulating function calculates all the points for that the planet and it’s moons will go through and plots that orbit. After that, the function creates a surface and puts corresponding images on it. Then the program goes to a long, simulating for loop that changes the position of the surface and rotates the image about the surface incrementally to simulate a planets movement.

Initially, I tried to use the same methods and structure as the previous build; however, I ran into specific problems that would not work with the hierarchical structure of the code. I firstly wanted to have a size and speed slider that would affect the simulation in real time. Unfortunately, since the GUI runs through the selections on each callback of UIControls, every time the user wanted to change the speed, the program would reset the simulation from the beginning, not only making the program look bad but taking up a lot of data processing considering the program would recalculate everything on each call. Moreover, since the last group hard coded every planet simulation, it was nearly impossible to allow the user to input variable number of planets with corresponding variable values for radius, orbit, spin, etc. Therefore, I decided to recode everything, from the GUI to the process of information gathering to the process of simulation. I needed a GUI that would not callback on every value change of sliders, a more efficient selecting process, a method to get data from the user, and a new method to simulate any variable number of planets/moons. I wanted the data acquisition method to involve real time simulations of the planet in the process of being created.

**III. Design, Flow Chart, Algorithm, Hardware, etc.**

A. Base Code for Simulation

This flowchart describes the flow of methods as the program runs. The whole program only contains four methods, “Second\_Solar\_GUI,” “getData2,” “getdata3” (capitalization mistake that is too deeply written) and “beginSim.” Second\_Solar\_GUI, or the GUI, only links to getData2 on the press of the button “Draw Now!” At getData2, the program links to getdata3 on the press of the button, “Describe Orbits.” In getdata3, the user has two options, to create more planets by pressing “Make another planet!” or to begin the simulation by pressing “Begin Simulation!”

3.A.1 A new GUI

In order to create slider bars and change the callback process, I decided to modify the last GUI figure by putting two slider bars (for speed and magnification) in place of the three drop down menus. I deleted the old callback functions and added my own. The new callback functions updated the handles structure like before but did not begin a selecting process. I also decided to change the button from re-orienting the user to instead beginning the simulation process. Other than renaming the figure and whole code to “Second\_Solar\_GUI,” I did not change the GUI much.

3.A.2 Getting Data

In order to create customizable planets, the user has to input the information; however, just inputting the numerical data for radius, apoapsis, orbit duration, etc. is boring and does not give the user a good picture of the planet until it is plotted. Therefore, I sought to create methods that plotted the current planet as the user inputs data through various sliders. In getData2, the code opens up a new figure and puts various UIControl and a subplot onto the figure. The first, a slider, varies the radius of the planet from 1,000 to 25,000 miles and has accompanying textboxes that describe how to work the slider and the current value of the slider. The second slider varies the rotation speed of the planet (like the length of a day) and has accompanying textboxes like the aforementioned slider. The next UIControl object is a pushbutton that changes the current image on the planet. It references a local callback function, “newImage,” that simply increments a global counter, “imCounter.” imCounter is an integer that decides which image in a vector of images, called “images,” to use (i.e. imCounter= 3 means using the Earth image). As for real time graphing, getData2 plots the planet by plotting a surface with an image on it in a for loop. In the for loop, getData2 checks the values of the sliders and imCounter and creates a surface, with radius specified by the slider, and an image, as specified by imCounter, that rotates using circshift based on the value of the rotation slider. This allows the user real time feedback like before. Moreover, in the for loop, the code constantly updates a matrix, “planetMat,” that holds all the values for radius, apoapsis, periapsis, etc. that has been entered so far.

3.A.2.1 Getting Data cont.

After describing the radius, day length, and image to use, the user presses the pushbutton, “Describe Orbit” to move on to getdata3. That callback passes the usual hObject, eventdata, and handles as well as planetMat and the last figure handle. In getdata3, it firstly closes the last figure using the handle. Next, it creates more UIControl objects on a new figure. These objects are very similar to before, except that here there are four sliders and two pushbuttons. The four sliders determine the “right” distance, “left” distance, semi-minor axis, and orbits per minute, respectively. “Right” and “left” refer to the distances on the major axis that are right and left as seen by the user. That may sound weird but while using the program, it becomes obvious. The two buttons, “Make another planet!” and “Begin Simulation!” callback to getData2 and beginSim, respectively. Both pass through planetMat and the “Begin Simulation!” also passes the last figure (to be closed in beginSim).

As for plotting, getdata3 plots the current orbit using a for loop, similar to getData2. In the for loop, the code constantly references the values for the “right,” “left,” and semi-mionr distances, and the orbit period. After getting the values, the code replots the elliptical orbit in a subplot. This allows the user real time feedback like before.

3.A.3 Simulating the Planets

As mentioned before, I had to redo the simulation code so that it can simulate a variable number of planets. In order to accomplish this, I used the same calculations to describe the orbit and movement of the planets as the previous group; however, I did it using multiple for loops and a handles vector. I described the orbits using a for loop that looped by the number of planets to describe and created three vectors/matrices, xPoints, yPoints, and zPoints, which describe the coordinates for the orbit. Each has thousands of columns to describe each coordinate for a specified time (i.e. 2 seconds in references column 20 or so) and each row describes the coordinates for a planet. For example, if I wanted to plot two planets, xPoints would have dimensions 2x300,000. Next, I used a for loop to plot all the points for all the planets. After that the code does some visual adjustments like setting the window and colors. Then in another for loop, I describe each planet as a surface –a collection of points in 3D that consist a surface- with image as described by imCounter, held in planetMat column eight. To make the surface, I took the coordinates X,Y, and Z for a basic sphere and multiplied them by the radius for each planet. Finally, the code ends in long, nested for loops that change the surface over time. The first for loop (using counter “v”) iterates through time and the second for loop (using counter “b”) iterates through the various planets. Within the second for loop, the code uses xPoints, yPoints, and zPoints to change the x, y, and z position of the surface. As the first for loop iterates, the second for loop uses larger indexes within xPoints, yPoints, and zPoints; therefore, moving the surface. Moreover, the for loop references indexes in xPoints, yPoints, and zPoints by a scale factor of the orbit length. Essentially, the shorter the orbit length, the faster the for loop goes through xPoints, yPoints, and zPoints. To rotate the planet, the second for loop circshifts the image along the second dimension (columns), by a ratio of v over 2\*pi, to make the image appear to rotate. Every time the iterator v gets to 2\*pi, the circshift returns to the original image. The day length, “dayL,” modifies the shift by multiplication to make it shift faster.

3.A.3.1 Simulating the Planets cont.

To make the GUI sliders for magnification and speed affect the simulation, I had the simulation adjust the axes by a factor of magnification and pause by the inverse of speed. The magnification slider has a minimum of 1 and a maximum of 3, thus, I made the axes grow larger when the value was less than 2 and vice versa. For the pause, the program pauses by a split second as decided by the speed slider value. This way, it does not affect the actual calculations of the orbits and only affects the figure/timing.

**IV Results**

At the beginning of this project, my lack of understanding as to how the program worked impeded making improvements and changing the immensely. For my first couple improvements, I was just trying to see what changing each variable did and making a few choice changes. However, there came a point where I realized I needed to actually sit down and figure it all out; after learning how the code functioned, my ability to modify it changed completely. After knowing that the code used hardcoded values and simulated the planets/systems as surfaces with changing coordinates, I realized I needed to create a simulator that could handle various inputs but also rely on the stellar math and plotting methods.

Many things came from creating the new simulator. Firstly, I decided to remove the theta and phi angles, which decide the tilt of the orbit, because it was not working as planned and I did not know how to explain to the user to change it. Adding in tilt would be the first improvement on this code one should make. Although tilt was put on the back burner, the new simulator was slightly faster because it did not have to calculate odd directional and normal vectors. Additionally, the simulator seemed faster due to the new streamlined code structure, as compared to the huge, hierarchical structure of the previous code. In coding the new simulator, I also realized I needed a way to get data from the user. For those, I created a preliminary data method, “getData,” that got data from the user using a while loop and message boxes; however, I realized typing in numbers was boring and abstract. Therefore, I created “getData2” and soon after, “getdata3” that created figures, put on UIControls, and plotted on a subplot.

Creating getData2 and getdata3 proved to be quite hard to code. I had to learn how to pass parameters through the call back, adjust the current subplot by the current values of the UIControls, and create various UIControls with numerous callbacks.

I also learned a lot about passing parameters and images. I nearly took out the handles structure because it seemed clunky, except I kept it to pass the values of the original GUI sliders. The other use of the handles structure was to pass the images of the planets so that they only have to be loaded once. I took that feature out because I only load the images twice and I would rather not pass them through for every time I passed the handles, which was often. Moreover, to make the circshift work the same for every image used, I needed to normalize the images. I decided to resize them all to 500x500. This required a cell array, “images,” to hold all the loaded images and a quick for loop resized each element. Loading images like this is not too slow and it only occurs twice.

After finally finishing this project, I have learned a lot about how Matlab works and have a deeper understanding of computer science in general. This project covered many concepts in Matlab from local functions to plotting intricate functions over time, all contributing to learning. Looking forward, if another group were to take over the project, areas for improvement would include the speed, accuracy to real life, and ability to plot pre-defined solar systems. The program runs a little slow; it could probably be tidied up. To make the program user friendly, I sacrificed a lot in accuracy to the orbits of planets in real life. Everything is magnified and closer together. Somehow making it more realistic would be cool. Moreover, the timing of the program might be off at points due to slow computations. I do not know how to adjust the program for operating speed, hopefully one could figure things out. Additionally, another improvement would be to have buttons for the user to create planets from our solar system with a single click. This would also add to the realness of the simulation.