Portfolio III

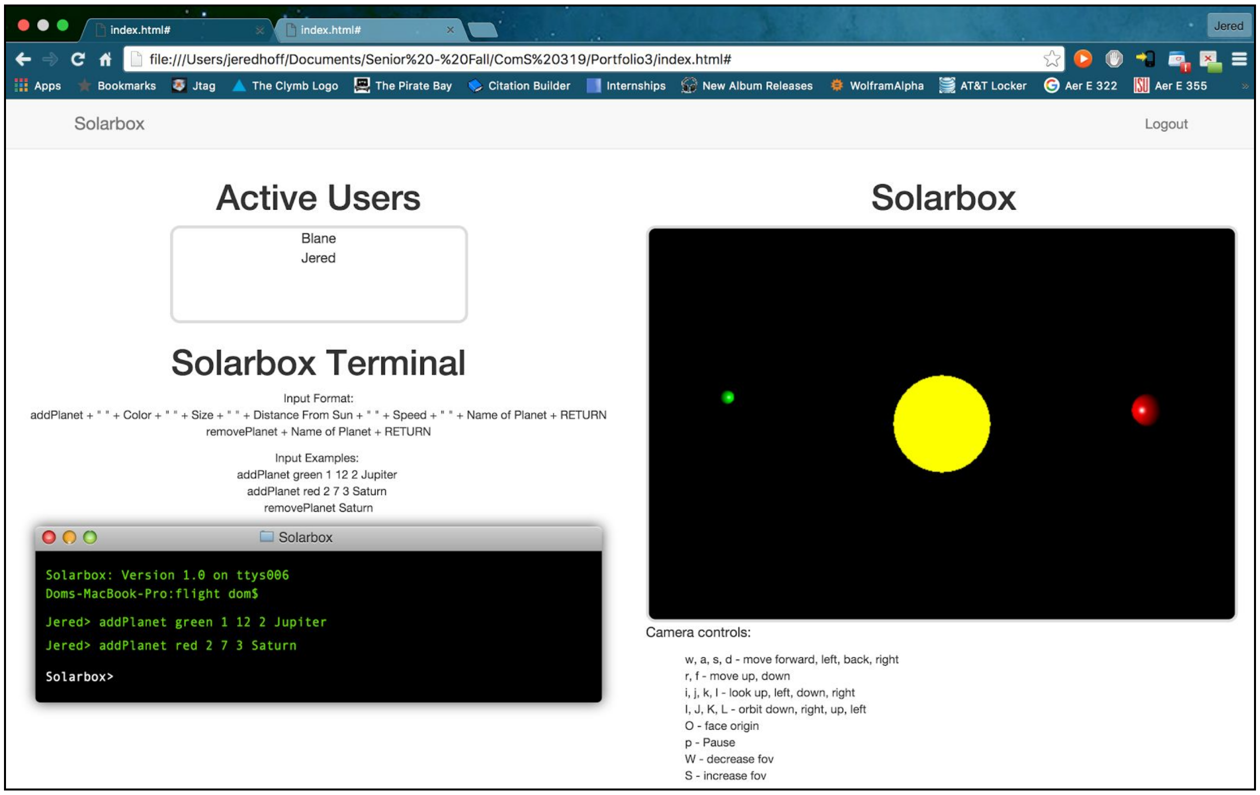
Computer Science 319
Fall 2015

Jered Hoff and Claude Cullen

Table of Contents

Introduction

The purpose of this portfolio was to demonstrate the web development skills we gained in the last few weeks of class and to bolster these skills to develop a useful website. After lots of brainstorming and bouncing ideas off of each other, we decided to build a website that hosted a Solar System Sandbox. This site will allow the user to enter commands into the terminal on the page and create a custom Solar System by either adding or removing planets. We came to this decision because we knew that this website would be a fun and interesting way to demonstrate our newfound skills. With web development being rather new to the both of us, there was lots of learning and research to do. Because our idea was rather complex, multiple issues presented themselves throughout the development process and we had to come up with numerous creative solutions to overcome these issues and achieve our goals.



Active Users

Blane
Jered

Solarbox Terminal

Input Format:
addPlanet + " " + Color + " " + Size + " " + Distance From Sun + " " + Speed + " " + Name of Planet + RETURN
removePlanet + Name of Planet + RETURN

Input Examples:
addPlanet green 1 12 2 Jupiter
addPlanet red 2 7 3 Saturn
removePlanet Saturn

```
Solarbox: Version 1.0 on ttys006
Doms-MacBook-Pro:flight dom$

Jered> addPlanet green 1 12 2 Jupiter

Jered> addPlanet red 2 7 3 Saturn

Solarbox>
```

Solarbox

Camera controls:

w, a, s, d - move forward, left, back, right
r, f - move up, down
i, j, k, l - look up, left, down, right
I, J, K, L - orbit down, right, up, left
O - face origin
p - Pause
W - decrease fov
S - increase fov

Interaction With Material

Questions

- How can we make sure that all active users can see the same data in the Solarbox Terminal, Active Users box, and Solarbox window at the same time?
- How do we create a custom language that allows the user to create their own custom Solar System in their Solarbox?
- How do we check for and handle improper user inputs in the Solarbox Terminal?
- How can we design a rather complex system that is easily used and understood by the user?

Insights

We used a variety of topics that were covered in class, and applied them to things that were useful and interesting. Our project displays our skills with Javascript, Node.js, parsing data with regular expressions, HTML, and CSS. Our use of Node.js allowed us to ensure that all data was being displayed to all users in the same way. This made the site very fluid and interactive which greatly enhanced our user experience. We also parsed the user input using javascript and regular expressions to handle improper user inputs. We also styled a terminal-like user input to make it easily understood that we have built a Solar System building language.

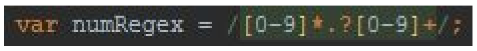
Complex Issues

Parsing Commands

One of the biggest questions we had when coming up with our initial idea was how would we parse the commands the user typed in. This was a concern of ours because we were unsure if antlr would be usable with javascript. After doing a little bit of research we found that there was a javascript target for antlr which would allow grammers to be used in our javascript. When we downloaded and tried to get it up and running we were unable to. We searched online and there was very little documentation on how to setup or use the javascript target for antlr. After we could not find any documentation we decided to give up on using antlr to parse our commands.

Instead of using antlr to parse our commands we used some simple regular expressions and some if statements in javascript. Thankfully we did not have many commands to parse or the parsing code would have gotten long and messy, but with the simple commands we were expecting, using javascript to parse was not a problem. The first thing we did was take everything the user typed and sent it to the server for parsing. When the command was received on the server the first thing we did was split it on spaces so that we could analyze each piece of it. We only had two commands in our application “addPlanet” and “removePlanet”. We had to find a way to only allow these two commands. If the user types anything else it would break our program. To fix this we simply used an if statement. We checked to see if the first piece of the command was “addPlanet” or “removePlanet” we would do the necessary action, otherwise we would alert the user that the command they entered was invalid.

The “addPlanet” command has 5 arguments; color, size, distance, speed, and name. If the distance, size or speed isn’t a number then the command is invalid. We had to write a simple regular expression to make sure that the arguments were correct. The regular expression we used to check for numbers is shown below.



```
var numRegex = /[0-9]*.?[0-9]+/;
```

The “removePlanet” command required only one argument, the name of the planet to remove (which was given it in the “addPlanet” command). It used this argument to find the correct planet to remove from the image and then remove it. Getting this argument did not require any parsing because the name of the planet could be anything. We simply had to check to see if the first part of the command was “removePlanet” then use whatever the second part of the command as the name of the planet to remove.

Using Graphics

In our website we took advantage of a useful and powerful graphics library called threejs. This is how we were able to render the solar system. A member of our group had used this library for another class in the past and thought it would be fun to use in this project. This library allowed us to easily create a scene with a camera, light, and multiple objects.

One of the issues we ran into using this library was adding objects to the scene dynamically. In the past we had used the library to render objects statically so no objects were added to the scene once it had been rendered. In our application we had to render the scene with no planets, and then add and remove planets to the scene as the user entered commands. To do this we had to create a method to add a planet to the scene that could be called externally. We also had to save the planets in a way that we could recognize what a planet’s name was so that we could delete it when the user called enter a remove planet command. To do this we had an array of planet objects which contained the information about the planets including the name, and an array of planet objects that was used in the rendering on the scene. We had to ensure that these arrays matched up so that we could find the index of the planet we needed to remove in the data array and use that index to remove the planet object from the scene.

Multiple Users

One of the most complex issues we had to deal with was allowing multiple users to edit the solar box at the same time. This required us to use node.js and socket.io to setup events and triggers that allowed multiple users to update the solar system at the same time. The basic flow of our mutli-user system starts with a user logging in. When a user logs in the client sends a 'login' event the server. When the user signs on the server will first emit a updatePlanets and updateCommands event to the new user. The server will then update its list of users to include the new user and emit a updateUsers event to all of the users who are connected. Now the new user has the most recent planets and list of previous commands and is ready to start typing commands of their own.

When a user types a command and hits enter the client emits a 'command' event and sends the entire command text to the server. The server then parses this text to determine what it should emit. If the server finds that the command is a correct add planet command then the server will do two things, first it will update its array of planets, then it will emit an add planet event and a makeshift planet object to all of the users that are currently connected. The connected clients will receive this event and then use the data to actually add a planet to their scene. It is important to note that there is a difference between the array of planets on the client side and the array of planets on the server side. The array of planets on the server side is only used to update new users to have the current solar system, after the initial update all planets are added directly to the client side array of planets. The remove planet command works similarly.

Bloom's Taxonomy

Analysis

Creating a Solar System sandbox was a fun way to show our ability to parse user input data and to create our own custom language. We also expanded this knowledge to create a Solar System sandbox that can be modified by multiple users to create a group workplace. This collaborative system could be used to create other workplaces that might be more practical, but our Solar System sandbox allowed us to show all of the skills that we have learned in the last couple weeks of this semester.

Evaluation

We decided that using a custom user input that looks like a terminal window would be a good way to make it clear to the user that our Solar System sandbox is supposed to be used like a programming language. When we started this portfolio, we knew that the user interface was a very important factor in our project’s success. The user can easily figure out how to use our sandbox, which greatly improves the user’s overall experience.

We also decided that creating a group workspace environment was very important. We wanted our website to behave flawlessly between different users, and our use of Node.js made this possible. Making the site change in real time for all users added a whole new dimension to our project. This system could be used to make all sorts of different group workspaces that enhance and promote group collaboration.

Creation

- Our website is a completely new website that was built from scratch and implemented multiple open-source libraries to create an optimal user experience.
- Created multiple data objects in javascript to handle the data moving between the users that are actively engaging with our site.
- Designed the user interface to make the solar system generating language feel much more like a language by adding the styled terminal.