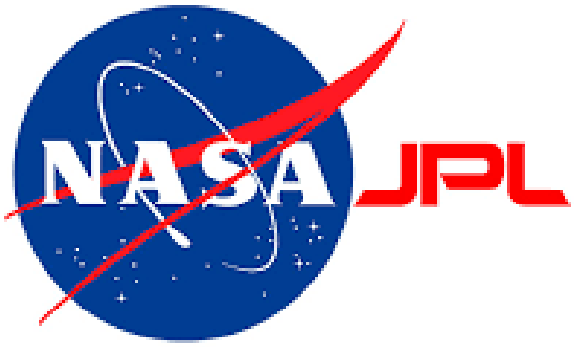
Senior Design Final Report

Collaborative Visualization for Solar System Treks



Version 1.0 - 05/14/2021

Team Members
Montague La France
Stanley Do
Zipeng Guo
Johnny Lee
Jose Garcia
Miguel Sanchez
Abdullah Alshebly
Odasys Soberanes
Christopher Smallwood
David Tang

Faculty Advisor:
Dr. David Krum

Liaisons:
Emily Law
Eddie Arevalo
Shan Malhotra
George Chang
Richard Kim

Table of Contents

1. Introduction

1.1 Background

NASA JPL's Solar System Treks (SST) is a public web application which enables users to access and visualize 3D geospatial data of various planets in our solar system. Users are provided various tools such as distance calculation tools and visual aids, allowing users to calculate the distance between points and helping the user navigate around a planet more easily. While this application is powerful on its own, it does not allow for any means of networked collaboration. Currently, users can only share their discoveries with those physically in the same room. Remotely, they can only share screenshots and links using the SST. In such an unprecedented time and especially during a global pandemic, it is vital that users have a means to effectively collaborate safely over the web.

While there are existing solutions, they are simply not feasible due to their niche target audiences and software requirements. Collaborative Visualization for Solar System Treks (CVSST), will open the floor for collaboration amongst all users on the web without requiring any additional software or fancy equipment other than a simple web browser. This will make it easier for JPL to integrate CVSST into SST. Also, CVSST stands out from previous solutions because it allows people to communicate and share changes made to the terrain in real time. CVSST will allow users to talk to each other and share their discoveries with one another safely over the web. Our solution comes with a multitude of improvements and tools as well, such as a chatroom system, drawing tools to mark up the 3D terrain, and the ability to export and import drawings for later use. Development is not always an easy feat, which is why our team decided to adopt an Agile workflow. We can get instant feedback from the JPL team and re-iterate on our solution at an extremely quick rate. By using a thick client architecture, web socket technology, and scalable back-end services, we can guarantee performance, scalability, and most important of all – access for as many users as possible.

1.2 Design Principles

The main focus of our software was to create a session. A session is defined as an online environment in which users can communicate, exchange data, and work with one another. Then we expanded the collaborative features in the session by allowing users to send messages using a chat box. Once the chat feature worked, we created drawing tools for 2D and 3D terrain. Then we created states, which is defined as the current state of all planetary data such as the planet/moon, layers data, collaborative markups, and waypoints, to store the changes made by users. The goal was to allow users to collaborate in the application.

1.3 Design Benefits

The benefit of the design as a web application was that it allowed a seamless integration into JPL's SST. It also makes the application simpler to use because users would only need a web browser. Users would not have to download any additional software or own special equipment. Being a web application makes it accessible because it has the potential to work across different devices and platforms. These benefits are important because JPL wants to make SST available to the public and to reach as many users as possible.

1.4 Achievements

Some of our major achievements were we added collaborative features. They include the ability to create sessions and type messages in the chat. Before CVSST, collaboration was very limited. The most users could do to collaborate was using SST to share screenshots or share links. Another achievement was adding drawing tools for 3D terrain, which the SST did not have. We also added more drawing tools for 2D terrain. In addition, we created states, which allowed users to save the data and changes made on the planet. Then users can export the state as a file and share it with other users, who can import the file and apply the state on their own version of the planet. The biggest achievement was that the student team, project advisor, and JPL successfully worked together on the project remotely. We never met in-person due to the global pandemic. It was through hard work, patience, and the dedication of excellent team leaders that we were able to persevere through the difficult pandemic and deliver the project to JPL by the end of the academic school year.

2. Related Works and Technologies

2.1 Existing Solutions

A Solar System Treks VR application was built by a previous student as a master's project. This application, called Trek VR, allows a user to visualize the globe in 3D space using virtual reality. However, Trek VR did not include collaborative features and was only available to a niche audience. Initially, we wanted to build off of Trek VR to include collaborative capabilities. We decided not to since we would not satisfy JPL's requirement of targeting the widest audience possible.

There were no previous projects to build upon. We only had JPL's Moon Trek code to work with and to use for testing. We had to create most of the collaboration features from scratch by using WebSockets. We also had to create the drawing tools on 3D terrain using CesiumJS, which the SST did not have.

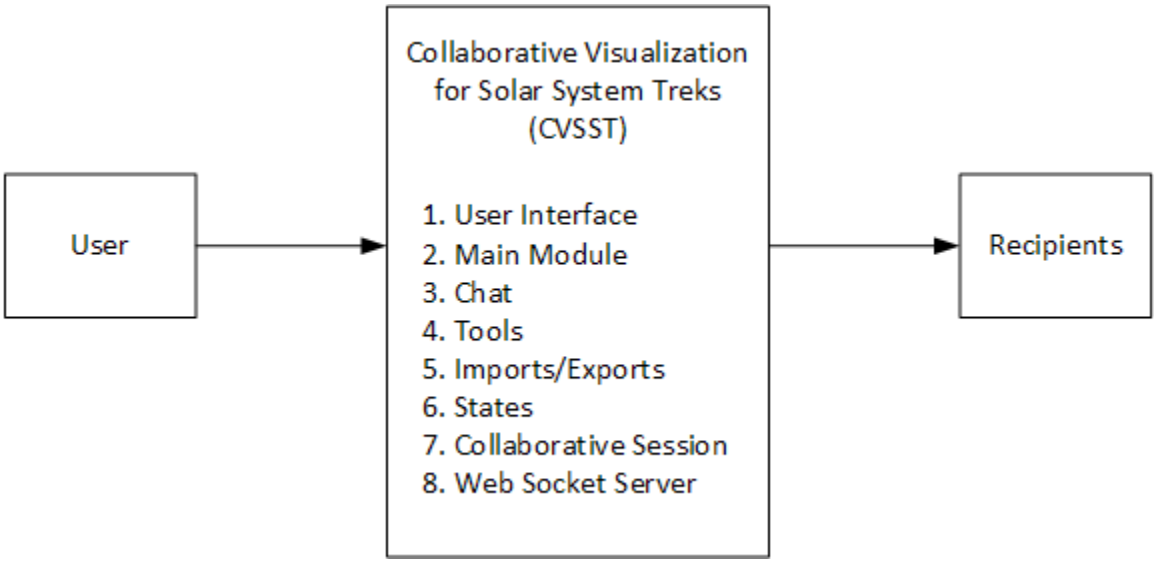
2.2 Reused Products

We reused some of the drawing tools code for 2D terrain. Lines, polylines, and freehand polylines existed in the SST tools, such as the calculate distance tool. The waypoints tool that we created used the Fly To function that was already in SST.

3. System Architecture

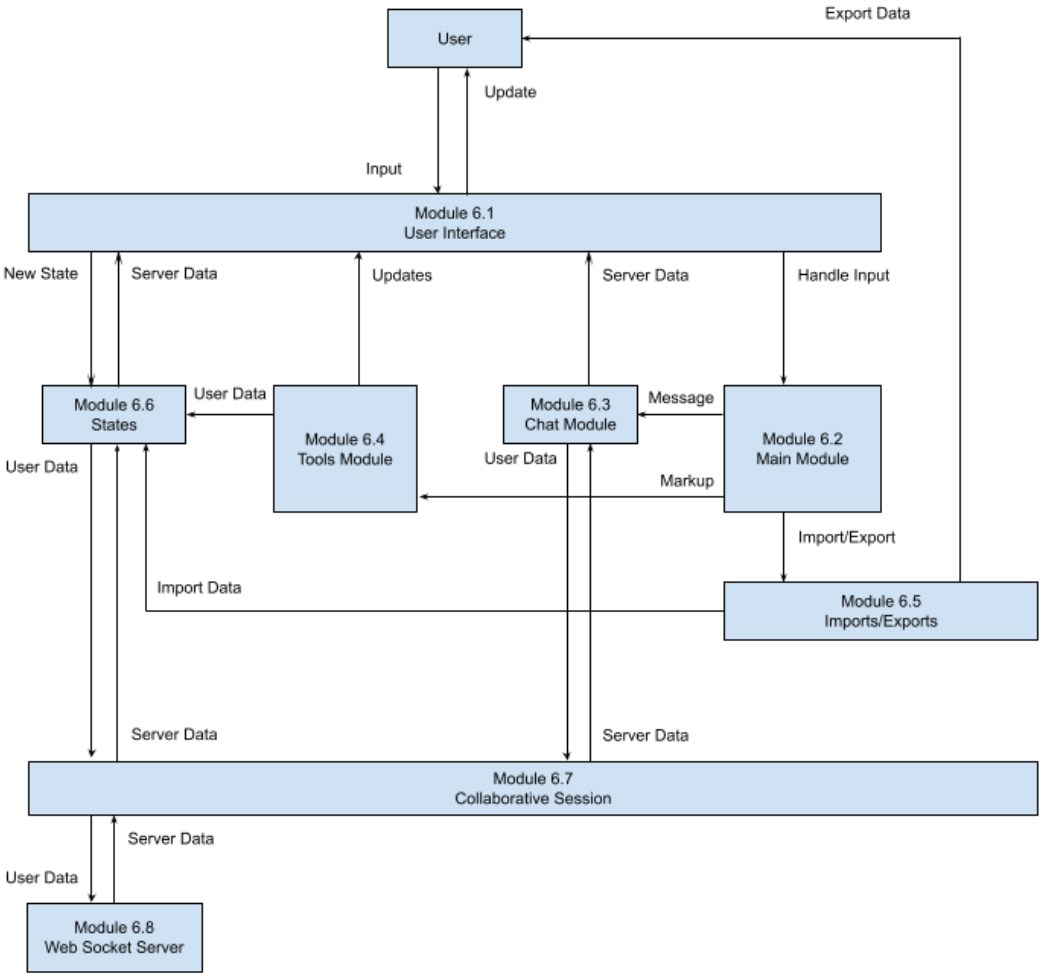
3.1 Overview

Figure 3-1. Level 0 Data-flow Diagram



Since our project was aimed towards providing additional functionality to existing software, we did not impact too much of the existing system architecture. However, we were able to implement additional modules and add new workflows on top. The modules are shown in **Figure 3-1. Level 0 Data-flow Diagram** and the recipients are the other people using the CVSST. All modules are made to interact with each other in their own enclosed code base, as shown in **Figure 3-2. Level 1 Data-flow Diagram** below. However, these modules are all instantiated in a single part of the code, which allows for all of the additional functionality.

Figure 3-2. Level 1 Data-flow Diagram



3.2 Data Flow

The data flow between each of the modules are shown in the **Figure 3-2. Level 1 Data-flow Diagram**.

3.3 Implementation

User Interface Module

- The User Interface Module serves as the messenger between the user and the Main Module. It provides a graphical user interface (GUI) for the user and allows the user to interact with the entire system.

- The User Interface Module consists of buttons, a chat box, and a tools menu used in the application.
- The User Interface Module was implemented with mostly code from the Dojo Toolkit. Some was with HTML and CSS.
- The user inputs data into the system via click events, scroll events, and key press events.

Main Control Module

- The Main Control Module serves as the main component where all of the CVSST modules are instantiated and initialized. It also handles all user input and directs the data to the correct modules for additional processing.
- The Main Control Module handles events from the User Interface Module. Based on the commands, it will interact with the Chat Module, Tools Module, Imports/Exports Module, or States Module.

Chat Module

- The Chat Module obtains data directly from the user interface and updates the user interface with new data from the Collaborative Session Module.
- The Chat Module handles all chat events from the user interface, updates the chat box with data from the session, and sends new data to the session, which will go to the server.
- When there is a message being sent via text, the Main Control Module will handle the event and call functions within the Chat Module, which retrieves user input from the User Interface Module. Once the input is validated, the data is sent to the Collaborative Session Module for processing, then to the WebSocket Server Module for storage into the database. The WebSocket server will recognize the data, store it in the database, and send the new entry to all users in the same collaborative session. When the WebSocket Server Module receives this new entry, the data will be sent back to the Chat Module to update the User Interface Module accordingly.

Tools Module

- The Tools Module provides all of the collaborative functionalities for markups and drawings on 2D and 3D terrain. It obtains user input from the User Interface Module and updates the tools component in the User Interface Module.
- The user selects which tool they would like to mark up the 2D or 3D Solar System terrain with, then clicks on a location on the terrain to place their drawing.
- The Tools Module also keeps track of the tools currently selected, the tools' user-defined attributes, and the location of the desired markup.

- The Tools Module sends all markup data to the States Module for processing and to be rendered in the User Interface Module. Then the data is sent to the Collaborative Session Module to be synchronized with other users.
- The drawing tools for 2D terrain used Esri ArcGIS code.
- The drawing tools for 3D terrain used CesiumJS code.

Imports/Exports Module

- The Imports/Exports Module handles all configuration imports and exports for CVSST states.
- The Imports/Exports Module consists of functions that parse a configuration file and pass it to the States Module for rendering onto the User Interface Module.
- The Imports/Exports Module allows users to export configurations in text format to be imported for future use. It obtains data from the Collaboration Session Module and sends imported data to the States Module.

States Module

- The States Module allows the user to save their session. The changes made during a session can be saved.
- The user can go back to a previous session by choosing a state. Refer to the Software Requirements Specification document for the definition of state.
- The States Module utilizes the WebSocket server's memory and browser memory.

Collaborative Session Module

- A collaborative session is an online environment in which users can communicate, exchange data, and work with one another. Sessions were implemented using WebSockets.
- The Collaborative Session Module serves as the main source of data transfer between the client and public WebSocket server. It transmits data back-and-forth with the WebSocket server. It also transmits data between the Chat Module, Tools Module, States Module, and Import/Exports Module.
- Each room was a session that users joined. If a room did not exist, then a user entered a username, created a room name, and created a password. After that, they click the "create" button to create a room. For others to join a room, they enter a username, the room's name and password, and click the "join" button.

WebSocket Server Module

- The WebSocket Server Module stores any data from the client into the database. It is in charge of all the data coming to and from the clients. It keeps track of all collaborative sessions, validates data, and controls all read and write functions into the database.

- The WebSocket Server Module receives data in the form of JSON or text from the client WebSocket, then processes the data so that it can be written to or read from the database.

The implementation of our code was mostly iterative since our primary goal was to complete the functionality first and foremost. One of the side effects that we have seen using this iterative approach is that oftentimes we strayed away from our best practices and design principles to make a certain functionality work. Even with a design in place, we would often go down a rabbit hole during implementation resulting in an increase in complexity.

4. Results and Conclusions

4.1 Project Results

We were able to add collaborative features. They include the ability to create sessions and type messages in the chat. We also added drawing tools for 3D terrain, which the SST did not have. We also added more drawing tools for 2D terrain. In addition, we created states, which allowed users to save the data and changes made on the planet. Then users can export the state as a file and share it with other users, who can import the file and apply the state on their own version of the planet. These collaborative features will make it easier for users to collaborate and work remotely, especially during the global pandemic.

While virtual reality (VR) does provide a more immersive user experience, web-based VR may prove to be an extremely challenging requirement due to its poor browser support and lack of stable/usable technologies. One of things we looked into was WebXR, but with its limited documentation and usability, it prevented us from accessing our desired range of users. The development team is also unable to meet in person to use and test VR equipment. We would have to create our own web-based VR API or use other existing libraries and integrate them into the SST software. As a result, we decided against implementing VR for the time being and instead, focused on adding collaboration features.

While working on the project, we identified some bugs. Some of those the team was unable to fix, while the others have been fixed. We believe it is worth letting the JPL team know about those bugs. They are listed below:

- Unfixable bugs (very complicated, tricky, and lack of time)
    - Arcing problems
        - Bug: Lines and shapes get distorted at the North/South poles in 3D view.
        - Bug: A problem with converting things from 2D to 3D, especially near the poles.
    - Weird occasional bug on Moon Trek and in CVSST code with 2D polyline
        - Polyline sometimes doesn't work for calculating distance and other tools.
        - Replicate bug: Double clicking draws a line. If you click at a different point, the line you just drew disappears. The polyline drawing seems to not end.
        - Bug disappeared by itself after one to two days
    - Bug: 2D drawing tools don't work properly on North Pole Map and South Pole Map

        - The 2D drawings on these maps do not appear at the correct positions on the 3D globe and sometimes an error message appears.
- Fixed bugs (working, but not the best solution)
    - ToolsModuleToolsComponent.js and ToolsComponent.js are somehow connected
        - The code for the 2D drawing tools is in the ToolsModuleToolsComponent.js file.
        - Bug: When the 2D drawing tools are used, they call some of the code in ToolsComponent.js file, which is an irrelevant file. This can be seen by looking at the console logs after using the 2D drawing tools.
        - Bug: The most recently used tool in ToolsComponent.js (calculate distance, elevation, etc.) will be called every time any 2D line is drawn, which uses the ToolsModuleToolsComponent.js file. The bug was fixed for the other line drawings. The 2D polyline drawing still has this bug.

Despite having satisfied a majority of the requirements, it is likely that some of the code implementation may incur a large amount of technical debt. *Refer to Section 10 of the Software Design Document to see the full list of requirements met and not met.* Since a lot of the programming was iterative, we found ourselves in many situations which moved us farther away from the initial design. Working with a completely new code base also created an extremely large technical cliff amongst the team. A lot of the time was spent researching the actual code. The Dojo Toolkit, CesiumJS, and Esri ArcGIS in the application were also older and outdated versions, meaning that only some documentation would be relevant. For a lot of us, JavaScript was a completely new language, and learning both front-end and back-end development whilst being full-time students was a major challenge.

4.2. Future

A future team can potentially expand and add more collaborative functionalities to CVSST. Some future research is to look into better VR solutions to implement. An alternative is for the JPL team to integrate CVSST into SST. These are some of the team's thoughts for the future, but the final decision is up to the JPL team.

Some major additions for future teams taking on this project would be to restructure how entities in CesiumJS are placed. Currently, CesiumJS creates a new worker thread every time an entity is placed, causing more resources to be fetched from the server. Despite enabling caching, we found that these workers would request a new copy despite the file not having changed. Eliminating these useless requests would improve performance significantly.

Another major addition would be the implementation of a database. *Refer to Section 8 of the Software Design Document for a detailed explanation on the database*. Despite having a database structure for states, entities, chat, and rooms, we were never able to implement a database to save data. All of the data is currently stored in memory on the server and there is no way to clear the memory without restarting the server. Having a database would allow us to actually save the data over long periods of time.

5. References

- WebXR
  - https://immersiveweb.dev/
  - https://github.com/Rufus31415/Simple-WebXR-Unity