# Final Project Design v. 1.1

Sumner Evans, Robbie Merillat, Sam Sartor

November 14, 2017

## 1 Design History

| Version Number | Date | Change Description |
|---|---|---|
| 1.0 | 2017–11–07 | First Draft |
| 1.1 | 2017–11–15 | Second Draft |

## 2 Overview

VR Technology has been developing rapidly in the 21st century. Current solutions such as Unity attempt to use old programming languages and paradigms to implement VR environments and thus limit developers' abilities to create new and unique environments. With every cutting edge technology, new paradigms and design patterns must be invented. This project intends to explore and implement these paradigms and design patterns.

For this project, we will utilize a 3D space to implement an intuitive UI for (a) loading programs, (b) saving program state, and (c) customizing the environment (described in Section 3). We believe that implementing a shell-like environment is the most effective method for us to explore those patterns.

## 3 Design

There are three main goals for our project (referred to as *VRsh* for now):

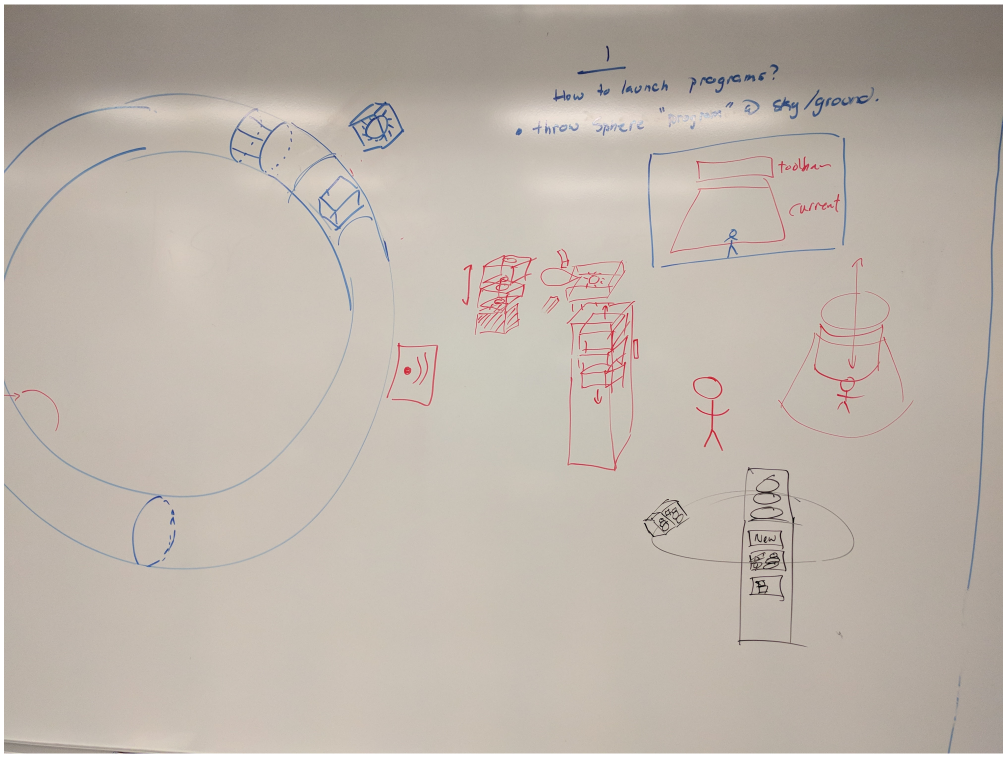
1. **Loading Programs:** the user will be able to load our three individual programs from our VRsh as described in Section 3.1.1.
2. **Saving Program State:** the user will be able to save the state of an environment (program). This is elaborated in Section 3.2.
3. **Customizing the Environment:** the user will be able to customize the VRsh environment. See Section 3.1.2 for details.

### 3.1 Visual Design

The visual design of this program is dynamic in that both the general settings and environment can be modified. This would look and feel similar to the Steam VR personal "desktop" with added options for settings and environment customization. "Running" a program will switch from the VRsh home environment to the desired program environment.

### 3.1.1 Global UI

There will be a persistent user interface that will be visible from the VRsh home environment and from within the individual environments that can be accessed from within VRsh. This UI will be a "halo" centered at the user, located above the user's head. The halo will have components attached to allow the user to perform actions such as running programs and moving between environments. We have not yet determined the most intuitive design motif for these controls. However, we have brainstormed a few innovative ideas for these controls including grab-and-pull-down selectors, Rolodex selectors, and system setting asteroid field of options. Some of these concepts can bee visualized in the image below.



### 3.1.2 Customizable Environment

Several components will enable the user to customize their VRsh home environment. Mentioned above, the system setting asteroid field will allow the user to create a new entity within their home environment. Some entities may last for a set amount of time without being interacted with, while others remain persistent. Examples of these entities could be:

- Brightness Bar: changes ambient brightness
- Skymap Texture: Throw to apply image as background
- Color Wheel: modify colors
- Mode: sitting vs. standing modes

- Various Objects: place objects around the home environment, allowing customization
  - Clouds/Mist
  - Boxes
  - Penguin
  - Mjolnir
  - Rugs
  - Plushes
  - Weapons (Pikes, Axes, Swords, Bow/Arrow)

### 3.1.3 Interactions

Interactions will include:

- Object surface/node snapping
- A yank control to bring objects closer/further
- Grabbing an object at a point
- Throwing objects (grab maintains momentum)
- Expanding fixed menus
- Applying tool objects to other objects
- Inter-object collisions

### 3.1.4 Lighting Model

We have several lighting/rendering models available, however most assets will use physically based rendering.

## 3.2 System State Storage

To facilitate the storage of environment state, VRsh will create a directory in the `.config` folder of the user's home directory called `vrsh` and store all necessary information in that directory.

Our preliminary folder structure is as follows:

```
.vrsh
|-> environments                        // all environment state data
|   |-> home                            // for the savefiles
|   |   |-> 2017-11-14-172312.save      // data format of these files can be
|   |   |-> 2017-11-14-173708.save      // determined by the program that is
|   |   |-> 2017-11-14-184937.save      // associated with this environment
|   |   |-> ...
|   |-> snowflakes
|   |   |-> 2017-11-14-172312.save
|   |   |-> 2017-11-14-173708.save
|   |   |-> 2017-11-14-184937.save
|   |   |-> ...
|   |-> lets-get-physical
|   |   |-> 2017-11-14-172312.save
|   |   |-> 2017-11-14-173708.save
|   |   |-> 2017-11-14-184937.save
|   |   |-> ...
|   |-> workbench
|-> config                              // for general configs (if necessary)
```

Individual programs are responsible for (a) opening these environments, (b) creating 3D "thumbnails" for the environments, and (c) creating the file's binary data. We became

# 4 Management

## 4.1 Project Scope

Our original goal was to implement a fully featured Linux shell that is able to wrap Linux commands in a VR environment. We realized that this was infeasible with our limited experience with VR and quickly changed the scope. At this point, we have created an achievable and measurable plan to implement the features that we want to include in the final project. Our plan consists of three milestones (see Table 2) with specific goals.

Table 1: Milestone Delivery Schedule

| **Milestone** | **Date** |
|---|---|
| Milestone I: Design Document | 2017/11/08 |
| Milestone II: Individual Project Program Picking (Demo) | 2017/11/27 |
| Milestone III: Final Code Submission<br>• Individual project improvements<br>• Saving Program State<br>• Customizable home environment | 2017/12/13 |

## 4.2 Risk Analysis

While we have done a significant amount of brainstorming and design work for our VRsh environment, we have not begun to implement any of the environment. This is a large risk for us, as there will be many unanticipated problems which may alter the complexity of our goals. Scope creep is another risk for our team, as a few of us are notorious for adding unnecessary features to our applications.

Currently, we have a decent idea of how to store state, however, we have not begun to implement it. This is another major risk for us. Another risk that we must mitigate is settling for less than what our goals because of deadlines. To counter this, we will will be proactive with scheduling to help ensure that we meet our goals.

Our dependency tree is relatively thick, so our code is vulnerable to external dependency disruption making the cost of refactoring high.

## 4.3 Test Plan

We will have an appropriate number of unit tests in our code, especially our libraries (`flight` and `picea`). Additionally, we plan to iterate quickly on the program, testing as we develop to help ensure that program behaves as predicted.

## 4.4 Demo Plan

We will be demoing our intermediate milestones during our meetings. Our goal is to have this project ready for the C-MAPP event on January 18, 2018.

# 5 Dependencies

These are the libraries that we are using in our application. Dependencies marked with a `*` are made by this team.

Table 2: Dependency List

| **Dependency** | **Purpose** |
|---|---|
| `flight*` | Rendering, graphical user interface, asset loading, and VR hardware library |
| `picea*` | Dynamic, event-oriented object trees |
| `serde` | Data serialization and deserialization |
| `nphysics` | Physics engine |
| `ncollide` | Vector and matrix math |
| `nalgebra` | Geometric/volumetric operations and queries |