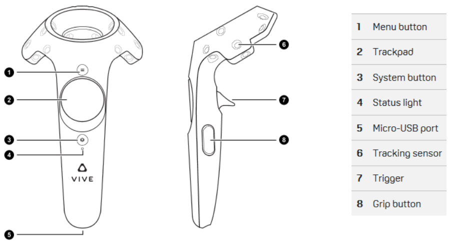
Framework: Current Features

State: September 19th , 2019

Note:
The following information does not ensure that these features still exist in future versions.
In case they change, please make sure to update either this document or create a new one.

1. Movement

Movement is mainly done by teleportation. If you calibrated an area, you can of course simply move around freely inside it.
To travel long distances, you have to use the teleportation feature.
At this point, you can achieve teleportation by using the ***trackpad*** button ❷.
Hold it down and aim at a location of your choice.
An indicator will tell you if the position is valid or not, mainly based on its color.
After you aimed at a valid position, simply release the button to teleport.

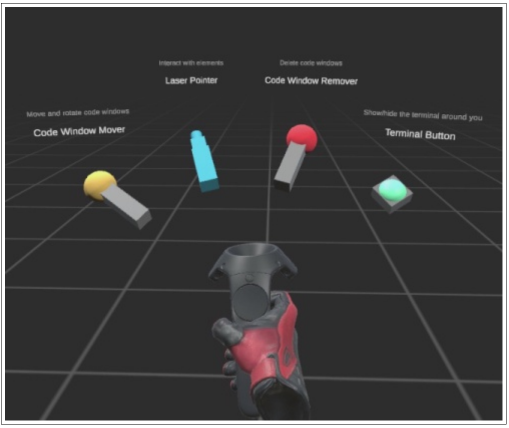


1	Menu button
2	Trackpad
3	System button
4	Status light
5	Micro-USB port
6	Tracking sensor
7	Trigger
8	Grip button

Adapted from https://docs.unity3d.com/Manual/OpenVRControllers.html

2. Controller Menu

To bring different functionality to the single controller, we decided to use a controller menu that allows you to switch between different types of controllers.
To achieve this, we made use of the SteamVR ItemPackage feature.
It allows to apply a different functionality to the controller, while also allowing to change the visual appearance of it. To open the menu, make sure your controller does not touch any possible other object in the scene and hold down the ***grip*** button ❽. The menu will spawn at the position of your hand.
While holding down the button, reach for any of the controllers and make sure your virtual controller touches them. They will light up if you do this step right.
Once done, simply release the button and your virtual controller will change.
Depending on your selection, buttons can now have different functionality.
For instance, you wont be able to teleport any longer, using the trackpad.
To switch back to the ***default controller***, which is capable of teleporting, you have to open the menu as described previously and just close it afterwards without selecting anything.



3. Controller Functionality

This section will briefly tell you about the functionality that each controller offers.

<u>Default</u>

The default controller is what you start with. These are its main features:

- Teleportation – using the ***Trackpad*** button
- Interaction with physical objects (e.g. to rotate the code city or turn the terminal) – grap using the ***Trigger***
- Open controller menu – using the ***Grip*** button

<u>Laser Pointer</u>

It provides the main interaction with other elements, terminal interfaces and more.

- Enable/Disable pointer – using the ***Trigger***
- Teleportation – using the ***Trackpad*** button when the laser is ***disabled***
- Interact with buttons and similar components (e.g. files, code city) – aim at them and using the ***Trackpad*** button
- Scrolling – aim at scrollable components and do a scroll gesture on the ***Trackpad***

<u>Code Window Mover</u>

Allows you to move and rotate already spawned code windows and over movable components inside the world.

- Pick up / Drop of movable elements – using the ***Trigger***
- Rotation – holding down the ***Trackpad*** button and moving the finger on the ***Trackpad*** to the left or right
- Distance change – holding down the ***Trackpad*** button and moving the finger on the ***Trackpad*** to the top or bottom

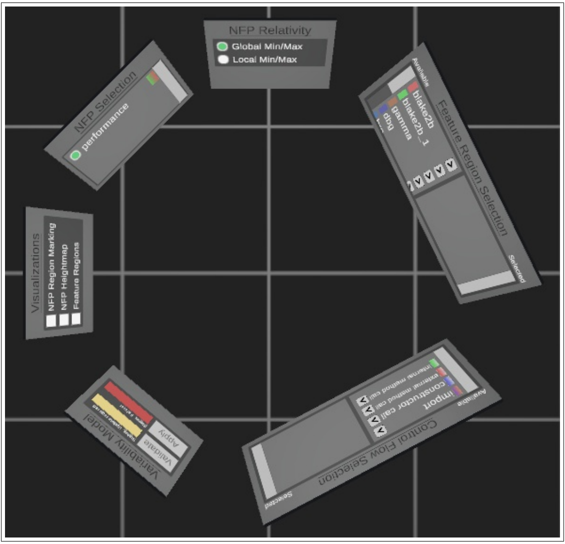
<u>Code Window Remove</u>

Allows you to remove spawned code windows.

- (Un-) Mark object for deletion – aim at a code window and click the ***Trackpad*** button
- Perform deletion – pull the ***Trigger*** (only short)

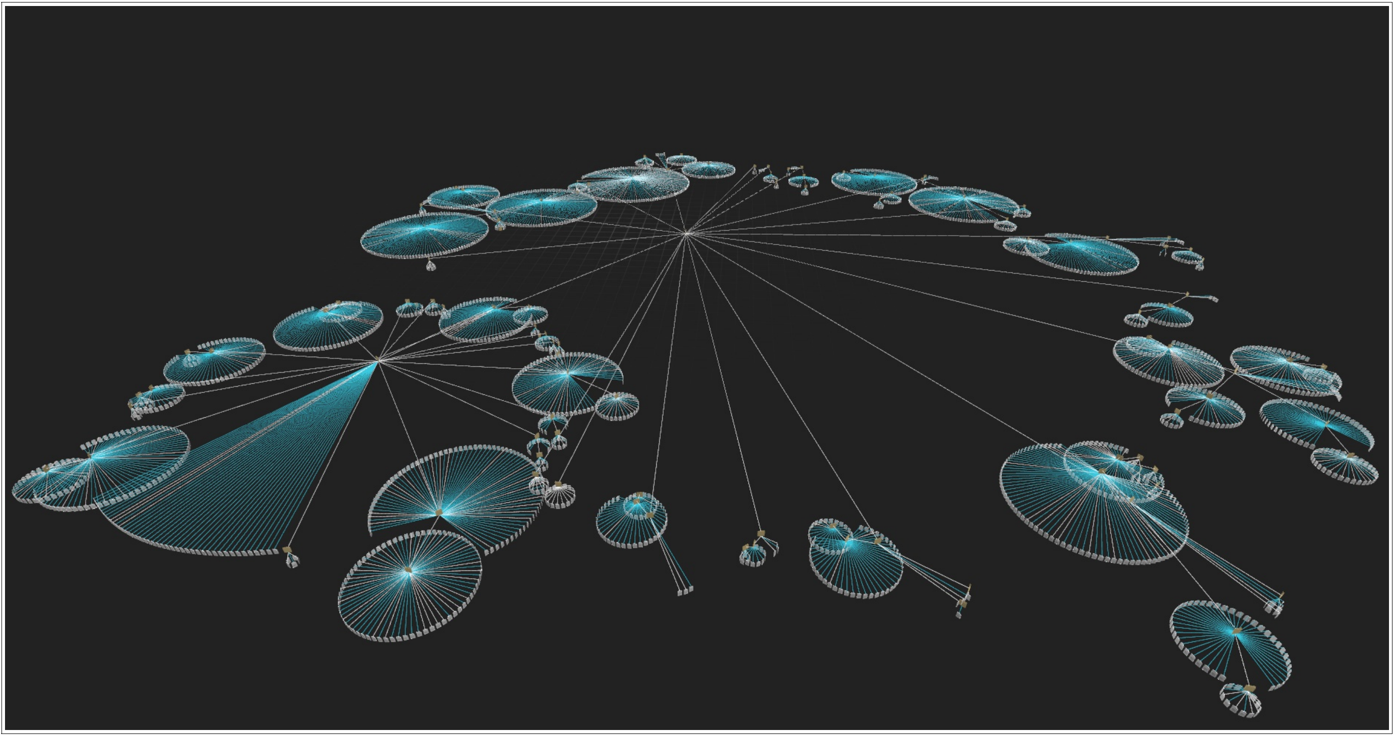
4. Terminal

The terminal is the current UI that allows you to change
the data that is used for visualizing specific parts in the scene.
For instance, it allows you to change the visibility of visualizations
(currently not shown on the figure) or to enable and disable them.
Furthermore, the variability / feature model can be validated, and its
configuration can be applied. Regarding non-functional properties (NFP),
it is possible to change the currently shown type of NFP or its relativity.
Relativity in this case means, that global shows the values of a region
with respect to all other regions that are currently taken into account.
Local modifies, for instance, the colors of regions with respect to the
min/max value of that file. Moreover, the terminal allows to select which
annotated feature regions should be shown in opened files and the
control flow selection provides the ability to enable/disable the control
flow visualization shown between opened code windows.



5. Software Structure / -Graph

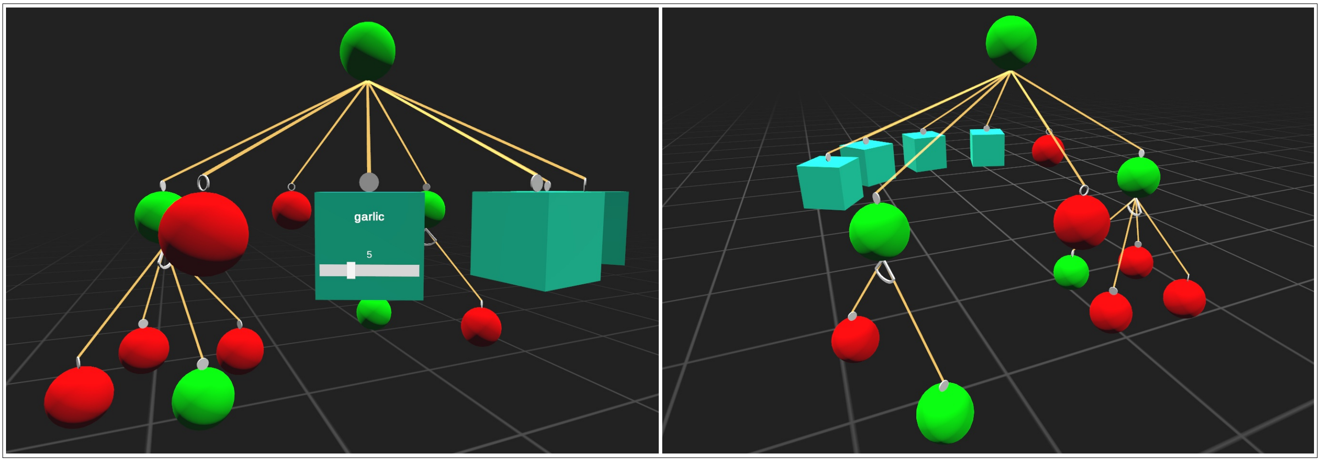
A graph that shows the directory hierarchy of the software that is currently visualized by the framework.



This graph is inspired by the cone-tree layout and currently uses the bubble-tree layout for visualization of the hierarchy.
For detailed information about its implementation, please refer to the “cone-tree.pdf” provided in the documentation.
You can interact with files and folders. By pointing at them, you get basic information like the path and name, how many files a folder contains or if a file is spawned and how many regions it has. By clicking at a file, it is also possible to spawn them.
This generates a code window at the desired location. The color-coding of the graph's edges is customizable and defined in the “mappings_extensions.json” file of the project examples. Using it, it is possible to quickly identify files by their file extension.

6. Variability Model Visualization - Feature Diagram

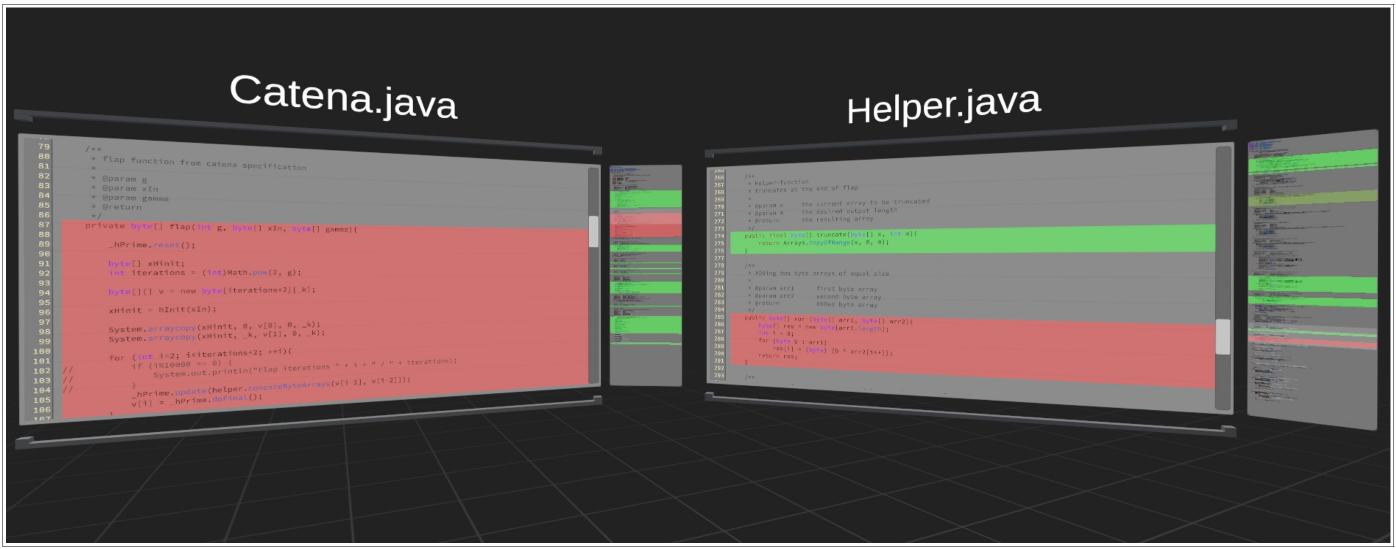
This is a visualization of the feature model, also using a cone-tree layout. In fact, both graphs now use the same code base.



Spherical objects represent binary options and numerical options are represented by the cubes.
Future tasks aim to improve this visualization by cross-tree constraints and similar features that are not yet available.
For now, you can see information about the options by hovering over it with the laser pointer equipped.
Furthermore, you can change their state/value. In case of binary options, simply click at them and they change their color and state accordingly. For numerical options, you have to click at them to open the menu shown in the figure left. You can them simply edit their values by modifying the slider. The slider makes sure that only possible values will be selected. Nonetheless, they are currently limited to a maximum of 10 to 11 different values (ticks), as the possible set of values can be very large.
For all options in general, it is also worth mentioning that they have the notation attached to them, telling you which features are mandatory (required) or which are just optional and don't have to be enabled at all. You should also consider the groups, shown by the notation below some options (e.g. "ALT" or "OR" group).

7. Code Window

The code window allows you to view the content of a file, analyse its regions, the influence of your feature model configuration, feature regions and control flow. Some visualizations can be computational intensive, so do not wonder if some have a significant performance impact on activation or modification. This also applies for a change of the feature model configuration.



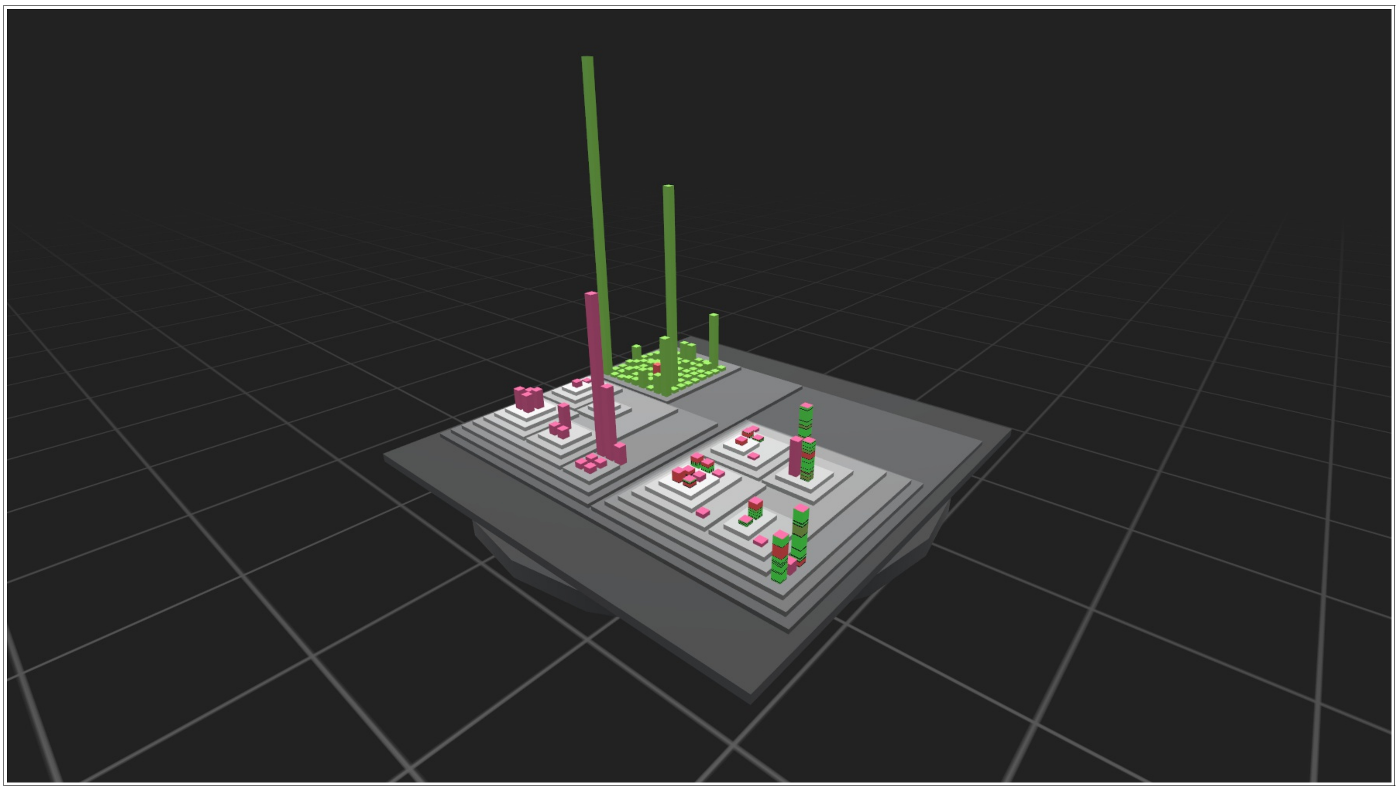
```
    /**
     * flap function from catena specification
     *
     * @param g
     * @param xIn
     * @param gamma
     * @return
     */
    private byte[] flap(int g, byte[] xIn, byte[] gamma){

        _hPrime.reset();

        byte[] xHinit;
        int iterations = (int)Math.pow(2, g);

        byte[][] v = new byte[iterations+2][_k];

        xHinit = hInit(xIn);

        System.arraycopy(xHinit, 0, v[0], 0, _k);
        System.arraycopy(xHinit, _k, v[1], 0, _k);

        for (int i=2; i<iterations+2; ++i){
//          if (i%10000 == 0) {
//              System.out.println("Flap iterations " + i + " / " + iterations);
//          }
            _hPrime.update(helper.concateByteArrays(v[i-1], v[i-2]));
            v[i] = _hPrime.doFinal();
```

```
    /**
     * Helper-function
     * truncates at the end of flap
     *
     * @param x      the current array to be truncated
     * @param m      the desired output length
     * @return       the resulting array
     */
    public final byte[] truncate(byte[] x, int m){
        return Arrays.copyOfRange(x, 0, m);
    }

    /**
     * XORing two byte arrays of equal size
     *
     * @param arr1       first byte array
     * @param arr2       second byte array
     * @return           XORed byte array
     */
    public byte[] xor (byte[] arr1, byte[] arr2){
        byte[] res = new byte[arr1.length];
        int i = 0;
        for (byte b : arr1)
            res[i] = (byte) (b ^ arr2[i++]);
        return res;
    }

    /**
```

Code windows can be placed anywhere in the available area. There is also no limit for how many you can spawn at a time.
Enabling control flow will draw edges between annotated files. Therefore, it may be required to disable some visualizations (e.g. the one on the left/right or both) by using the terminal. This also applies if you want to see absolute values of the regions, which can be achieved by enabling the heightmap visualization on the right, which will disable the overview you currently see there.

8. Code City

The code city is one of the latest major features added. Its task is similar to that of the software graph.
It shows the whole software projects hierarchy in a city metaphor and allows to quickly identify specific properties of files.

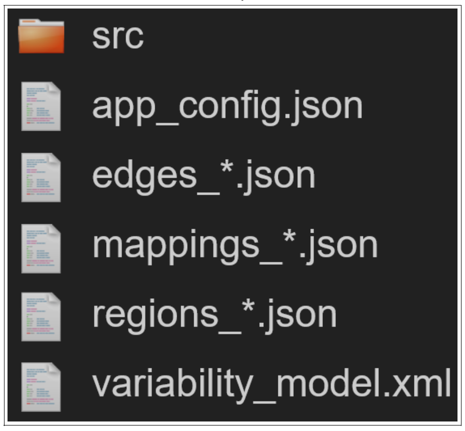


Its implementation is based on the official research by Richard Wettel (https://wettel.github.io/codecity.html).
We have extended it with texturing of the buildings. As you can see in the figure above, some buildings have textures that show the regions and their relative values encoded by color. As this is still the first version of this visualization, it offers great opportunity for improvements and extensions, for which we have already gathered many ideas. An obvious feature that is currently missing is the possibility to show the control flow edges. We look forward to see this implemented in future versions.
As a user, you can interact with it as usual. Pointing at elements shows you quick information about them. Clicking at buildings allows to spawn the code window as done similar using the software graph. The platform below currently serves as a tool to lift the whole code city (modify its height) and to rotate it around its center.

9. Configuration / Customization

The whole framework is highly customizable. This is currently achieved by using the “workspace” functionality. All the files (including the source code of the analysed project) are stored in a main folder. A rough overview is shown by the figure on the right. A global application configuration (“app_config.json”) allows you to provide information like which folder includes the source code, which files should be ignored or what extensions should be removed and more. The other files allow to provide the edges, mappings and regions, which are described in detail in this documentation. These descriptions can be found in the “file-specs” folder. All in all, you are able to modify what is shown and how. A future task is to bring this configuration inside the application. This is not yet done so that is why you have to create and edit the files before running the app.

Workspace



src
app_config.json
edges_*.json
mappings_*.json
regions_*.json
variability_model.xml