Railway Visualization

Using the Visualization

The railway visualization is created for the KIELER Simulation Visualization (KiVis). When running a simulation of the environment or controller, the visualization will present the current state of the system by positioning trains and changing colors and other graphical attributes of elements in the image. Furthermore it is possible to interact with the visualization.

Interactions

The following presents the possible interactions.

- Power and unpower all tracks using the corresponding buttons
- Power and unpower tracks by clicking the corresponding track
- Toggle state of points via click
- Toggle state of lamps via click
- Toggle state of contacts via click

There are 3 states in which tracks can be powered using the visualization

- When the track is unpowered and clicked, the track will be powered to move in travel direction (120)
- When the track is clicked again, the power is changed to be against the main travel direction (-120)
- The third click will unpower the track (0).

Animations

The following presents how variable values are visualized.

- **Trains** are positioned on the tracks they are standing on
  - The environment visualization moves trains to the correct path and position on that path using the values of *train_track*, *train_branch* and *train_pos*.
  - The controller visualization moves trains to the middle of the track they are standing on, based on the value of the *position* array
- If a **train** is **not setup to stand on the railway** (track is greater than 100), the train will be set to be **invisible**
- Colors show the **track speeds**
  - red, yellow and pink are used for power in the main travel direction
  - red: 100..120, yellow: 80..100, pink: 1..80
  - blue, purple and green are used for power against the main travel direction
  - blue: -100..-120, purple: -80..-100, green: -1..-80
  - a speed value of 0 will show the normal track as it is in the SVG file
  - this animation is created by changing the stroke size and color of the paths on the track
- Colors show the state of **points** and **lamps**
  - points that are STRAIGHT are visualized with a white background
  - points that are BRANCHED are visualized with a green background
  - lamps that are OFF have a transparent background
  - lamps that are ON have a yellow background
- Colors show the state of **contacts**
  - contacts that are ON have a big green stroke added to them (big enough to be visible even if a train is shown above that contact)
  - contacts that are OFF are shown as in the SVG image
- The fill color of **signals** is transparent, if that signal is off and visible if it is on
- A sonic beam is visible around the bell if the **bell** is ON
- **Error labels** are shown for each **train** as well as each **wagon**
- An additional error label is shown if a **collision** occurs

The SVG File

An SVG file has been created, in which the elements that are animated have a unique Id. The Model Railway Track Layout SVG has been choosen for this task, because it shows a more detailed view of the railway, in contrast to the Model Railway Track Scheme.

For details see the Model Railway Layout SVG.

The KiVis File Templates

Another file required for the visualization is a kivis file. It defines how the SVG elements are animated to represent the running simulation. For the railway, this file is rather complex and thus created from Freemarker templates.

The following explains the structure of the Freemarker template files that generate the kivis files.

There are dedicated visualizations for the environment SCChart and controller SCChart. Both reference the same base templates, that provide animations and interactions for common railway elements, e.g., points and contacts. Further the same constant definition file is used by the templates.

This results in the following template files for the visualization:

File	Description	References
EnvironmentVisualization.ftl	The top level template for the environment visualization. Has animations for the trains, based on their exact position from the environment (track, branch, pos). Furthermore, errors detected by the environment are shown via animations.	BasicVisualization.ftl
ControllerVisualization.ftl	The top level template for the controller visualization. Has animations for the trains, based on the track position, that is assumed by the controller.	BasicVisualization.ftl
BasicVisualization.ftl	Has animations and interactions that can be used by both, the environment and controller. This includes animations and interactions for the bell, crossing lights, crossing gates, lamps, points, signals, contacts. Furthermore the power of the tracks is animated and can be changed with a click on the corresponding track.	Constants.ftl Util.ftl
Constants.ftl	Defines railway constants, mostly taken from the wagon.sctx Furthermore the file contains classification of the tracks, to calculate with these. For instance, there are lists with all tracks, only with tracks that have branches, or which branches belong to which tracks.	
Util.ftl	Simple utility macros and functions.	

Constants.ftl

This file contains constant assignmets to various variables such that they can be used in other templates.

A big portion of those variables are the typical railway interface (which track has which number, how many tracks / lamps / points exist, values for STRAIGHT / BRANCH, ON / OFF, etc.).
Another huge portion of constants are actually defined in the *wagon* SCChart from the environment and represent natural constants from the railway. These includes all positions (point positions, contact positions, etc.) as well as track lengths.

The third main part of constants classifies the railway tracks (which tracks are branched / unbranched, have contacts, are unidirectional / bidirectional, etc.). These are used in higher level templates to calculate the final visualization file.
For example there are more signals on bidirectional tracks than on unidirectional tracks that have to be animated differently. These classifications are thus used to easily iterate over suited tracks to easily create a good kivis file.

BasicVisualization.ftl

This file generates the visualization for all elements that both—the environment and controller—use. Normally this is the case for the common interface variables, such as points, lamps, track_speeds, signals, etc.
On the other hand what is not handled here are variables specific to the environment (exact position of trains, errors) and controller (position of trains determined from reed contact signals).
Another animation that is placed here is hiding a train, if that train is not positioned on the railway (train_setup <= 0).

The following shows a more complex example of how signals can be animated, to illustrate the usage of templates and pre-defined variables from Constants.ftl

Bidirectional signals template

```
//-------------------------------------------------------------\\
//--             SIGNALS OF BIDIRECTIONAL TRACKS             --\\
//-------------------------------------------------------------\\
<#list bidir_tracks as track>
<#list ["fwd", "rev"] as dir>
<#assign dirNum = if(dir="fwd", 0, 1) />
<#-- Some tracks don't have signals -->
<#if no_signals?seq_contains(track)><#else>
animate ${track}${dir}Red {
  apply color using signals[${track?eval}][${dirNum}] {
    fillOpacity: 0 is 0, 1 is 1, 2-3 is 0
  }
}
<#if (dir="fwd" && no_fwd_yellow?seq_contains(track))
  || (dir="rev" && no_rev_yellow?seq_contains(track))><#else>
animate ${track}${dir}Yellow {
  apply color using signals[${track?eval}][${dirNum}] {
    fillOpacity: 0-1 is 0, 2 is 1, 3 is 0
  }
}
</#if>
animate ${track}${dir}Green {
  apply color using signals[${track?eval}][${dirNum}] {
    fillOpacity: 0-1 is 0, 2-3 is 1
  }
}
</#if>
</#list>
</#list>
```

The main loops iterate over all bidirectional tracks (defined in Constants.ftl) and over both directions ('fwd' and 'rev'). This makes sense because the index of signals on bidirectional tracks is 0 for the first in main travel direction, and 1 for the second signal in main travel direction.

There are bidirectional tracks that do not have signals (defined in Constants.ftl) thus there is a test to check if the current track is one of those without signals. In this case there is no need to create the animations.

What follows is the animation for the red, yellow and green circles of the signal. The naming scheme of the SVG file has to be considered, to animate the correct element (see Model Railway Layout SVG).
The array with the states of all signals is calld **signals** and is two dimensional (see Interface). The first dimension is for the track, the second is for the direction (0: first in main travel direction, 1: second in main travel direction). Thus **${track?eval}** will evaluate the name of the current track (e.g. KH_LN_7) to find an associated value. As all track constants are defined in Constants.ftl, this statement will result in the index of the track that the current iteration is for.

In conclusion the animation of the red, green and yellow circles is created on all bidirectional tracks that really do have signals, based on the array that holds the corresponding value for that signal. This is the intented behaviour.

ControllerVisualization.ftl

This template positions the trains on the middle of the path for track, that the train is currently standing on. The controller can only calculate the track of a train but not its exact position using the inputs from the reed contacts. Thus this simple animation is created for the controller.

EnvironmentVisualization.ftl

This template creates the visualization of trains on the main tracks as well as branches of tracks (e.g. KH_ST_4branchKH_ST_5). The environment saves the exact position of trains on the track, which is used for the generated animation.

Furthermore, errors calculated in the environment are visualized using labels. This involves two labels in most cases, one for the error message, one for a corresponding number (e.g. a message such as "point error", and a number with the actual point that has this issue).