Train driving interface using Statecharts

General information

- The due date is December 5th, before 23:55.
- Submissions must be done via BlackBoard. Beware that BlackBoard's clock may differ slightly from yours. *All* results must be uploaded to BlackBoard and accessible from links in the main (index.html) file.
- The assignment must be made in groups of maximum 2 people. It is understood that all partners will understand the complete assignment (and will be able to answer questions about it). Clearly identify who did what.
- Grading will be done based on correctness and completeness of the solution. Do not forget to document your requirements, assumptions, design, implementation and modelling and simulation results in detail!
- To get feedback about the assignment workload, provide the number of hours you spent on this assignment.
- Contact: Simon Van Mierlo.

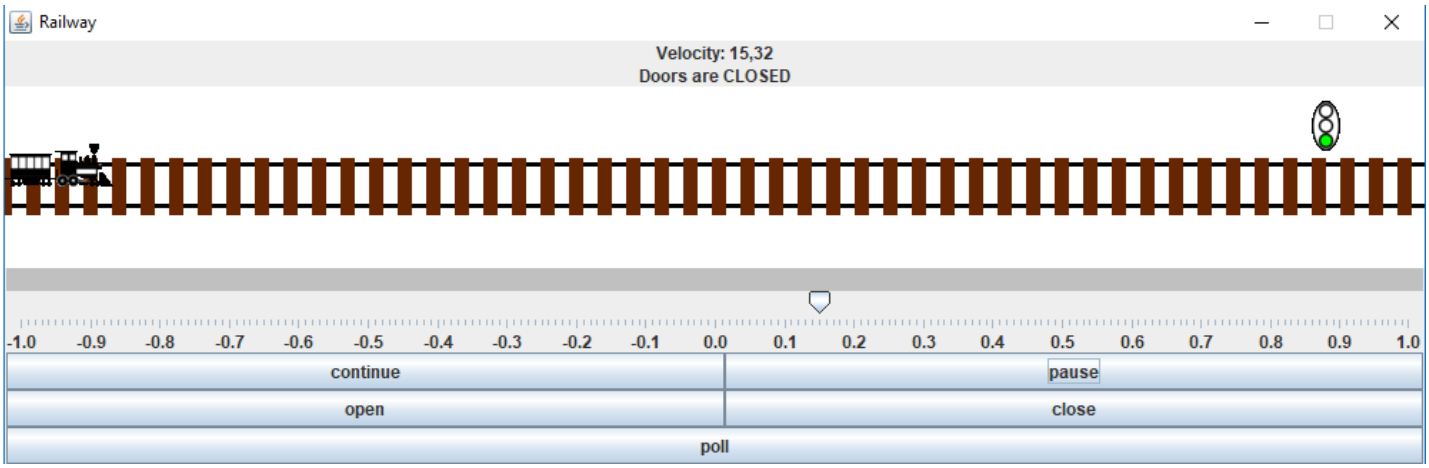
Goals

This assignment will make you familiar with modelling in Statecharts, simulation, and code synthesis.

Problem statement

In this assignment, you will design a Statechart model spcifying the reactive behaviour of the interface for driving a train.

You are provided with a simple GUI, which will send events to your statechart, and which will respond to the commands issued by the statechart. The GUI itself is therefore stateless and non-intelligent, as all state and knowledge is encoded in your statechart.



The GUI presents a simple interface with a slider to set the acceleration, thus automatically controlling the engine and brakes. The slider ranges from -1 (emergency brake) to 1 (full throttle). There are additional buttons to open and close the doors. Buttons are also present to pause and resume the simulation. Finally, there is a button which the system uses to poll whether or not the train driver is still able to drive, the so called "dead man's button".

In this assignment, you will create the statechart which listens to the events generated by this interface, and modifies the behaviour of the train. Instead of directly linking these buttons to their respective actions, some safety measures should be encoded to guarantee that unsafe things (*e.g.*, opening the doors at full speed) will never happen.

Requirements

Your statechart should comply with the following requirements:

- A train is either standing still (either in a station, or outside of it), driving, or cruising at full speed.
- The train can only open its doors when it is standing still in a station. Pressing the 'open' button at any other time will just be ignored.
- The train can only open its doors once in every station: when the doors have been closed, they cannot be opened again until the next station is reached.
- After opening the doors, they must remain open for at least 5 seconds to allow people to unboard and board. Pressing the 'close' button before that will just be ignored.
- Standing still does not necessarily imply that you are at a station, so the doors should remain closed if the train is outside of a station.
- When reaching the maximally allowed speed (100 kilometers per hour), further positive acceleration will be ignored.
- Likewise, when standing still, further (negative) acceleration should be ignored. Make sure that in all cases, it is impossible to go backwards!
- The train can go in emergency brake mode when dangerous situations occur. In this mode, the interface no longer listens to the driver, and the train will decelerate as hard as possible (i.e., -1), before turning itself off when speed has reached 0.
- After an emergency brake, when the speed has reached 0, the interface becomes responsive again after a cooldown period of 5 seconds. That is, speed must remain equal to 0 for 5 seconds before the interface can be used again.
- When driving in a train station at a speed above 20 kilometers per hour, the train automatically goes into emergency brake mode. This holds at all points in time while in a station.
- When passing a red light, the train automatically goes into emergency brake mode.
- When passing a yellow light, the speed must be lower than or equal to 50 kilometers per hour. If it is higher, an emergency brake should be triggered.
- After passing a yellow light, the speed must be limited to 50 kilometers per hour until the next light is reached. Only when a green light is reached,

can the train drive at the maximum speed again.
- The driver needs to press the "dead man's button" every 30 seconds. If the button is not pressed within 5 seconds of the prompt, the train goes into emergency brake mode. Pressing the button before the prompt occurs will have no effect.
- When a train has opened its doors, it ignores all acceleration requests until its doors have been closed again. This to prevent the train from leaving with its doors open.
- Pressing the pause button will stop the simulation. Pressing the continue button afterwards will make it resume.

Events send to the statechart

The following events will be raised by the environment to the statechart to indicate a button press, slider change, or event happening:

- `close`: the "close" button was pressed.
- `open`: the "open" button was pressed.
- `leave`: leaving a station.
- `enter`: entering a station.
- `pause`: the "pause" button was pressed.
- `continue`: the "continue" button was pressed.
- `awake`: the "poll" button was pressed.
- `yellow_light`: passed a yellow traffic light.
- `red_light`: passed a red traffic light.
- `green_light`: passed a green traffic light.
- `update_acceleration`: the 'acceleration' slider was modified. It has a parameter containing the new acceleration.

Interface of the GUI to the statechart

The following output events are sensed by the environment:

- `openDoors`: opens the doors.
- `closeDoors`: closes the doors.
- `error`: displays a notification (string parameter) as an error.
- `warning`: displays a notification (string parameter) as a warning.
- `clearWarning`: clears the error or warning currently shown.

The following operations are defined:

- `updateGUI`: updates the visualization. You should not call this method from your statechart, as it is automatically called every 20ms.
- `print`: prints a message to the console. Use this for debugging purposes.

Additionally, the following attributes can be read and/or modified.

- `velocity`: the current speed of the train. You can modify this value to round of errors, for example, but do note that the velocity is automatically calculated every 20ms based on the current acceleration of the train.
- `acceleration`: the current acceleration of the train. This should be set to the value passed by the slider if acceleration is allowed.

Starting point

As you should only focus on the reactive behaviour of the interface, we have provided a starting point for you, which you can download here. To import the zip-file, in Eclipse go to `File > Import... > General > Existing Projects into Workspace`. Then browse to the zip-file and import the `Train` project. The starting point already includes the binding with the GUI, which is also provided. You should only have to modify the statechart model in the `model.sct` file. Do not change the UI code and/or the interface already defined in the provided model. To generate the code, right-click the `model.sgen` file and click `Generate Code Artefacts`. Run the main `TrainDemo.java` file in the `src` directory to start the UI.

Practical information

Modelling of the Statechart model can be done using Yakindu SCT. Follow the installation instructions presented on the website. to install the tool and run it.

Please hand in your modified project (zipped), and a report (in HTML or PDF). In your report, please include your model as an image. You can use the export functionality of Yakindu. Alternatively, just take a screenshot.

You are strongly encouraged to use all functionality that Statecharts offer, such as history states, after events, and orthogonal components. Yakindu's extensive documentation can be found here.

Maintained by Hans Vangheluwe. Last Modified: 2018/11/19 05:23:39.