Victoria University of Wellington
School of Engineering and Computer Science

SWEN326: Safety-Critical Systems

Assignment 1

Due: Monday 27th April @ 23:59

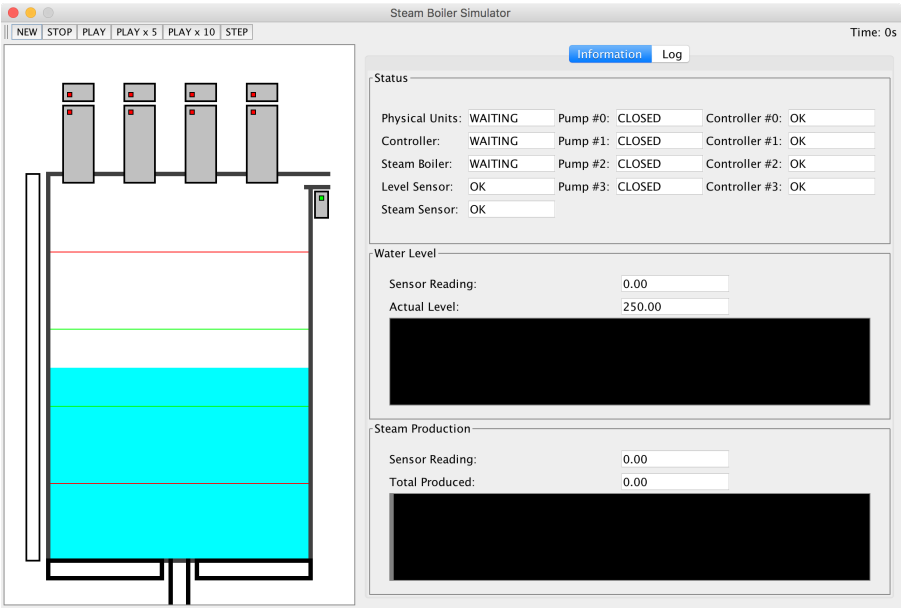
NOTE: For this assignment, you must use the giblab repository created specifically for you: `http://gitlab.ecs.vuw.ac.nz/course-work/SWEN326/2020/<username>`

Overview

This assignment is concerned with the *steam boiler specification* and you should familiarise yourself with this. The purpose of the assignment is to gain experience developing a safety-critical system under the restrictions imposed by a realistic coding standard. In addition, this will help you develop familiarity with the steam boiler specification.

Simulation

The main component of this assignment is the implementation of a control system for the steam boiler system. This is to be written in Java in the context of an existing *simulation framework*. The following illustrates the framework:



The simulation framework provides an implementation of the "physical units" which form part of the steam boiler system. In addition, the graphical user interface provides a means to see and control the overall functioning of the system. For example, one can tell when the boiler controller is malfunctioning. Likewise, one can configure a given physical component to itself malfunction and, hence, to test the boiler controller responds appropriately.

What to Do

In this assignment, you will be implementing the steam boiler control software. This must communicate with the physical units according to the protocol set out in the steam boiler specification document. There are two requirements placed on your implementation:

1. **Coding Standard**. Your submitted source code must adhere to the "SWEN326: Java Coding Standard". This means, for example, it should compile successfully with all warnings set as errors. Likewise, all field, method and class declarations should have appropriate JavaDoc comments. A copy of the standard should be available on the assignments page, *and you should read this carefully.*

2. **Version Control**. You are required to use version control in an appropriate fashion for this project. Specifically, you should use the GitLab created specifically for you at `http://gitlab.ecs.vuw.ac.nz/course-work/SWEN326/2020/<username>`.

To get started, you should download the accompanying software artifacts: `sb-simulation.jar` and `sb-controller.jar`. The former provides a binary library with which you can run the simulation. You should add this to your project's "Build Path". The latter provides a code template for the controller and accompanying tests. *It is not expected that you use more than one source file for this assignment!*

HINT: Don't forget to activate `checkstyle`!

HINT: Don't forget to enable assertions using the "`-ea`" switch from within Eclipse.

HINT: You might find it helpful to worry about memory allocation once you have a prototype working. Then, you can go back through and figure out how to allocate memory only during initialisation.

HINT: You should definitely write your own test cases! The tests provided cover only a relatively small range of scenarios and you should expect the hidden tests used by the automark system to be much more complete.

HINT: The steam boiler specification is ambiguous and lacks clarity in places. You should be prepared to ask clarifying questions on the forum and/or consider the implications of design choices made in the simulation framework.

Questions

A careful analysis of the steam boiler specification reveals a surprising number of unanswered question. To help you get started on thinking about this, a number of questions arising are given below.

- *How to know how many pumps to enable?*
- *What if evacuation rate is too high?*
- *How to tell there is a valve failure?*
- *How to tell if pump has degraded performance?*
- *What to do if a pump is locked on?*
- *How should physical units respond to emergency stop? open valve; heat off; pumps off.*
- *What is difference between degraded mode and normal mode? Presumably only what is transmitted!*
- *What happens if steam sensor failure?*
- *What happens if physical units fails to acknowledge a failure detection message? How long should it take to respond?*
- *Does a rogue* `REPAIRED\_XX` *message count as a transmission failure?*
- *What does the operator desk do?*
- *How to distinguish a pump failure from a controller failure?*
- *How should an emergency stop be handled by the physical units?*
- *What constitutes a transmission failure?*

To help, a detailed failure analysis of the steam boiler problem is provided in the Appendix. For the purpose of this assignment, this failure analysis should be considered sufficient.

Marking

This assignment will be marked out of **100**, with marks based on four distinct areas:

- **Correctness (40 marks)**. The correctness of your implementation will be assessed based on the number of passing (hidden) tests *as determined by the automated marking system*.
- **Code Quality**. The overall quality of your code will be assessed manually (by tutors). The marks for this section are broken down as follows:
  - **JavaDoc (5 marks)**. The quality of the JavaDoc comments you have written will be assessed. Comments are expected to be both *concise*, and to provide *accurate* summaries of the methods and/or fields in question.
  - **Decomposition (5 marks)**. The manner in which your code decomposes the problem will also be assessed. This means that methods should provide *logical* and *coherent* units which can be *easily understood*.
- **Coding Standard**. The degree to which your source code adheres to the required Java Coding Standard will be assessed using a combination of manual inspection (by tutors) and automated checking (e.g. by running the Eclipse compiler). The marks for this section are broken down as follows:
  - **Eclipse Warnings (20 marks)**. The degree to which your code meets the requirements of Rule 9 (warnings as errors) will be assessed.
  - **Checkstyle (10 marks)**. The degree to which your code meets the requirements of Rule 10 (`checkstyle`) will be assessed.
  - **Others (10 marks)**. The degree to which your code meets the requirements of Rules 1 – 8 will be assessed.
- **Version Control (10)**. The manner in which you have used version control will also be assessed. Specifically, it is expected that all commits have sensible and coherent commit messages. As such, you are recommended to use a sensible style for your commit messages.[1]

NOTE: the test cases used for automated marking will differ from those in the supplementary code provided for this assignment. In particular, they may constitute a more comprehensive set of test cases than given in the supplementary code.

Submission

Your assignment solution should be submitted electronically via the *online submission system*, linked from the course homepage. The following file is required for submission:

`steam.boiler.core.MySteamBoilerController.java`

You must ensure your submission is accepted by the submission system without error. This means it must: (a) successfully compile against `sb-simulation.jar`; and, (b) pass at least one hidden test case.

[1] See "How to Write a Git Commit Message", `https://chris.beams.io/posts/git-commit/`.

Appendix — Analysis

This document provides an analysis of the steam boiler problem. This includes the operation of the steam boiler in normal mode and, more importantly, some discussion of the failure analysis required. It is assumed that the reader is familiar with the steam boiler problem.

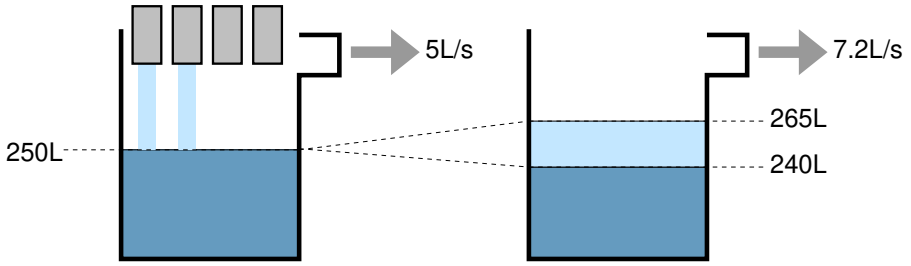
Normal Mode

The operation of the steam boiler in *normal mode* is relatively straightforward. The key challenge on each cycle is to determine how many pumps to turn on and how many to turn off. This is done based on a prediction of the water level at the next cycle. Suppose the current water level is given by L, the pump capacity is given by C, the maximum steam rate is W and the actual steam level reading is S.[2] Then, a reasonable prediction for the maximum and minimal water levels with n open pumps at the next cycle would be:

$$L^{max} = L + (5 \times C \times n) - (5 \times S) \tag{1}$$

$$L^{min} = L + (5 \times C \times n) - (5 \times W) \tag{2}$$

Let's consider a concrete example. Consider the following scenario where each pump has a capacity of 4L/s and the maximum steam rate is $10L/s$:



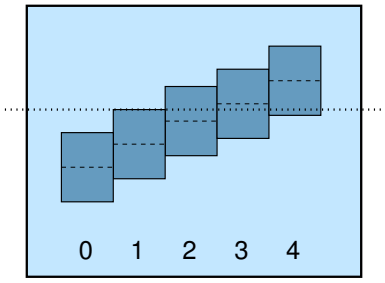
After 5 seconds the pumps will put in $5 \times 4 \times 2 = 40$ litres of water and, if no water was lost, then the predicted level would be $290L$. But, water is being lost to steam. In fact, as we can see, the rate in which steam is being produced is actively *increasing* over the $5s$ period. This is probably happening because, for example, the boiler is heating up during this time. Since we cannot know the exact amount of steam which will be emitted we have to make a *conservative* prediction. *We assume the rate of steam will not decrease*.[3] Thus, if the rate stayed as it is, then the amount emitted would be $5 \times 5 = 25$ litres. Or, we might consider a worst-case scenario where the rate went up immediately to its maximum value then the amount emitted would be $10 \times 5 = 50$ litres. *In practice, it will be somewhere in between.*

Hitting the Target

The steam boiler problem requires that the water level be maintained between a *minimum* and *maximum* (normal) level. In order to do this, we need to utilise our water level predictions to determine how many pumps to have on. *A simple strategy is to try and keep the level as close to the mid-point as possible*. For each combination of open/closed pumps we can then compare their predicted ranges against this midpoint value. The following illustrates:

[2] The original steam boiler spec includes a maximum increase in steam rate; but, our boiler characteristics does not include this. Having this, however, would improve our prediction accuracy

[3] This could only happen if the heating element itself failed, or we ran out of water



Here, we see a given target level against the predicted ranges for a given number of open pumps. For example, the predicted range if we turn all pumps off is well below our target. It seems that having three pumps on here would be ideal.

Failure Detection

Failure detection is by far the hardest part of the steam boiler problem. In completing the failure analysis, it becomes clear that some aspects of the physical hardward could be improved for better reliability. As is common for safety-critical systems, we will make the following assumption:

> "In any given period, at most one component can fail"

This significantly simplifies the problem whilst remaining an entirely reasonable assumption. It means that, for example, no two *pumps* can fail in the same period. Or, that a *pump* and its *controller* could fail together, etc. Without this assumption the problem is very hard to analyse indeed.

Failure Observations

As for operation under normal conditions, failure detection is based around *prediction*. That is, we predict what the hardware will do and, when we observe something different, we have detected a failure of some kind. The challenge with the steam boiler problem is that detecting a failure is not itself enough — *we must also attempt to detect what has failed*. We can identify the following kinds of failure observations:

- **Invalid Sensor Reading**. If one of the sensors reports a reading which is obviously incorrect, then something is wrong. For example, a negative reading or a reading which is outside the valid bounds. Or, a change in reading beyond the physical limits. An example of the latter would be the water level dropping from say 200L to 0L in a given period, as this would be considered physically impossible.[4]
- **Unexpected Pump Response.** If a pump does not respond as requested, then it has clearly failed. For example, if we requested the pump to open then we would expect to see it registered as open in the next period.
- **Unexpected Controller Response**. If a pump was requested to be opened, then we would expect the controller to register flow in the next period. [5] The challenge here, however, is to determine whether or not it is the controller or the pump which is at fault!
- **Unexpected Water Level**. Based on our water level prediction (see above), we may notice that the water level is not where we might expect. It could be below the minimum level we expect, or above it. There are many reasons why this might happen. For example, a pump has failed (and, if so, we would expect a signal from the controller to corroborate this). Likewise, the level sensor may have failed. Alternatively, the steam sensor may be reporting an incorrect level leading to an incorrect prediction.

[4] Though, note, of course if the steam boiler suddenly developed a very large hole it might just be possible!

[5] Note, the steam boiler specification states that pumps take 5s to open; however, our physical hardware does not reflect this!

- **Unexpected Steam Output**. In general, we expect the amount of steam being produced not to decrease.[6] This is because, provided there is water in the boiler and the heating element is working, then steam will continue to be produced.

To make these observations, we must record the expected state of the *entire system*. However, interpreting the signals is not trivial and a careful analysis is required.

Failure Analysis

We attempt to perform an exhaustive failure analysis. The goal is to identify situations where the controller can detect individual component failures, and response appropriately. To do this, we base our analysis on a binary representation of the boiler state, illustrated as follows:

P_i	C_i	L
$\times$	$-$	$-$

Here, P_i refers to the ith pump, C_i refers to the ith pump controller, and L to the water level. For simplicity we ignore the steam sensor, though this could easily be incorporated into the analysis.

To understand what our state above means, first explain the intuition behind the mark $\times$ and the dash $-$. The former is used to signal that the response from the component did not match expectations, whilst the latter signals that it did match expectations. For example, in our example state above, we have that the ith pump's response did not match our expectations (and this clearly signals a pump failure).

In our analysis, we only consider the responses from the ith pump and controller. This follows the assumption of a single component failure as an incorrect response from two unrelated pump components indicates a *multiple component failure* (and can be ignored).

Figure **??** provides an exhaustive discussion of all possible states. The conclusion from this is that some observations clearly indicate component failures, and others can be more ambiguous. Another important conclusion is that *corroboration* between observations is critical. Indeed, one could design a more reliable system, for example, by including two controllers per pump. This would then make distinguishing a pump failure versus a controller failure much more accurate.

[6] This does not account for the change in water temperature caused by pumping in cold water!

$\mathbf{P}_i$	$\mathbf{C}_i$	**L**	**Commentary**
–	–	–	Indicates physical units operating as expected.
–	–	×	Indicates water level is not within predicted range. Since everything else appears to be functioning correctly, indicates a failure of water level sensor. Observe could also indicate *valve failure* or *heating element failure*, but we are ignoring these possibilities here.
–	×	–	Controller i has responded incorrectly (e.g. showing water flowing when we not expected). This is *most likely* a controller failure, but could signal a pump failure. For latter, might expect to see a corroborating level signal. However, if predicted water range is large, then pump might have failed (e.g. stopped pumping) but water level remained within expected range.
–	×	×	Controller i has responded incorrectly (e.g. showing water not flowing when is expected to be), and water level is not within predicted range. Assuming controller response and water level *corroborate* each other, indicates a failure of pump i. For example, if controller indicates water is *not flowing* and water level *below* predicted levels, indicates pump i has failed. Otherwise, suggests a *multiple component failure* (which is ignored).
×	–	–	Pump i has responded incorrectly (e.g. failed to open when requested) which indicates it has failed. In this case controller *does not corroborate* observation, suggesting pump is behaving as requested but just not responding correctly.
×	–	×	Pump i has responded incorrectly (e.g. failed to open when requested) which indicates it has failed. In addition, water level is not within predicted range. These responses could be consistent with a pump failure. For example, if pump is *not pumping* and water level is *below* expected levels. Otherwise, suggests a *multiple component failure* (which is ignored).
×	×	–	Pump i has responded incorrectly (e.g. failed to open when requested) which indicates it has failed. The ith controller *corroborates* observation.
×	×	×	Pump i has responded incorrectly (e.g. failed to open when requested) which indicates it has failed. The ith controller *corroborates* observation and water level *may corroborate* it. For example, if pump has failed off and water level is below prediction, then is consistent. But, if pump has failed off and water level is *above* predication, then suggests a *multiple component failure* (which is ignored).

Figure 1: An exhaustive analysis of the failure state space.