Introduction to Computers and Engineering Problem Solving
Spring 2012
Problem Set 8: Rail switching
Due: 12 noon, Friday, April 27, 2012

1. Problem Statement

Railroads use radio remote control systems for an operator on the ground to control engines switching cars in a rail yard. We will simulate a simple version of such a system in this homework, which will have just one track, one railroad car, and one engine. Your program will allow remote control of the engine speed, direction and coupling to a car. It must have a safety feature that stops the engine if it detects a lighted object in its range, such as a car with an end marker.

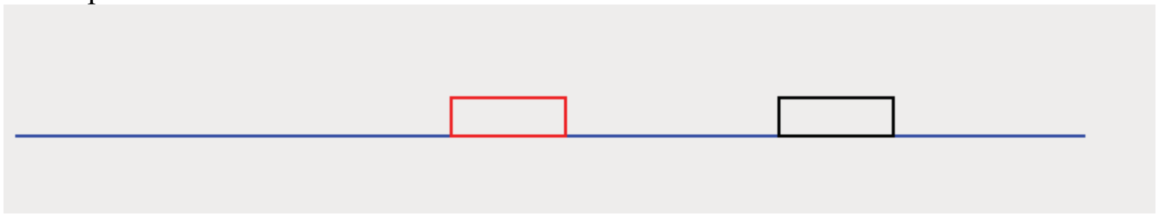
You should base your program on the Phidget and Swing vehicle application from lecture, although you will need to make substantial modifications:

- The slider sensor must control engine speed in either direction. A value near the middle is zero mph; allow a range (e.g., 450 to 550) to be zero speed. The high range (550 to 1000) indicates positive speeds and the low range (0-450) indicates negative speeds (going backwards). The GUI must show the engine moving across the display.
- You will have to keep moving the slider sensor to keep generating events so that the engine keeps moving. This feature exists in real systems, and is both an annoyance and a safety feature: if the operator is incapacitated, the engine will stop.
- The force sensor must be used to couple or uncouple a car from the engine. The car and engine must be adjacent to be coupled or uncoupled, and the speed of the engine must be zero. Use a force threshold of 300.
- The light sensor must be used to initiate an emergency stop. Cars and employees have reflective surfaces and the engine has lights and a light sensor that detects reflected light. When the light sensor detects a high light reading, the engine speed is immediately set to zero. The engine can be restarted only after the light sensor reading is low. Use a light sensor threshold of 300.

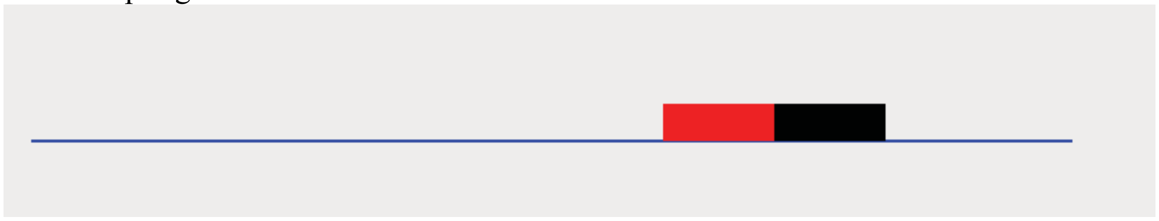
In a real system, you would use the Phidgets digital outputs to send digital signals to control the engine. These are being simulated by calls to repaint() in the GUI in this homework.

The figure below is a very simple example of the GUI. The engine is red, the car is black, and the track is blue. The figure shows the starting position, with the engine and car separated. In the solution GUI, the car and engine are “filled” (rather than drawn) when they are coupled. You must indicate when the car and engine are coupled, but you may do it in any way that you wish.

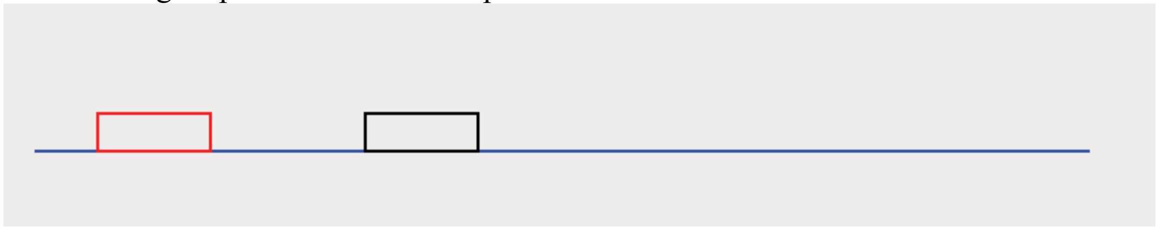
Initial position:



After coupling:



After the engine pulls and then decouples from the car:



2. Program

Use the basic structure from the vehicle example in lecture 26.

- The GUI must show at least the track, the engine, and the car.
- The display must show the location, direction and speed of the engine. No text is required; the display must show the movements of the engine and car.
- After the engine has approached within a meter/pixel of the car, it can then be coupled to the car by pushing the force sensor, but not otherwise.
- When the engine and car are touching each other but are not coupled, moving the engine in the direction of the car should cause neither the engine nor the car to move.
- The engine can be coupled to or uncoupled from the car by pushing the force sensor only when the engine is stationary (i.e., only when the slider value is between 450 and 550). Commands for coupling or uncoupling must be ignored when slider has a value outside the 450-550 range.
    - · You may get multiple coupling events when pushing the force sensor, since it will generate a number of events at similar pressure values. Implement a simple way to limit this problem if you can see one; it's ok if the user needs to push the force sensor a few times.
- Your engine and car must stay on the track segments shown in the GUI. If they hit the end of track, speed is set to zero. Have your track go from x=100 to x=800.
- The GUI can be quite simple—it does not need to be beautiful, just functional.

- Your application must use a model-view-controller pattern. The view just does the drawing, as shown in the examples. The model computes speeds, locations, coupling, etc. The controller draws the frame, opens and closes the Phidgets interface kit, and handles sensor events. The controller should have a "quit" button so that the text file being written (see below) can be closed correctly.

You must write the key events from your simulation to a text file using a Java stream. Let the x location of engine or car be its leftmost coordinate. You must write the following data to the text file:

- The engine's x location of any coupling event and uncoupling event.
- The x location if the engine hits the start of the track.
- The x location if the car hits the end of the track.
- The engine's x location and sensor value of any light event that causes the engine to stop.
- The final x locations of both engine and car.

Decide the appropriate method and class design to write these events. Also:

- Write the event type as well as the required data.
- Call the file HW8Out.txt.
- Minimize the number of duplicate entries in the text file. Use booleans to detect when you're in a given state, and only write to the text file when the state changes (e.g., if you're at end of track and stay there, write that event only once).

When you run your program you must:

- Move the engine next to the car.
- Couple to the car.
- Go to the right end of track.
- Go to the left end of track.
- Go back toward the center of the track.
- Uncouple from the car.
- Move the engine to the left, and stop it with a light sensor event.

These events must be recorded in the text file that you submit.

Your sensors must be on the following analog ports:

1. Force sensor
2. (Rotation sensor—used only in extra credit)
3. Light sensor
4. Slider

We will deduct points if your sensors are not on these ports. (We need to be able to grade your submissions on our test setup efficiently.)

Sample output is shown below. Yours will be different, based on how you manipulate your sensors:

```
Coupled at: 600
Engine and car at end of track: 720
Engine at start of track: 100
Uncoupled at: 394
Engine at start of track: 100
Light event: 205
Last position of engine: 205
Last position of car: 394
```

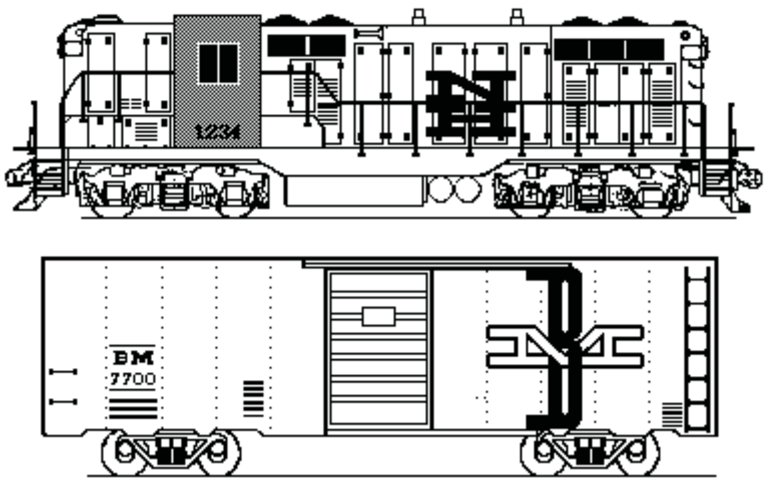
3. Extra Credit

In addition to your solution above, you may implement additional functionality for this problem set and get up to 40 extra credit points. **You must first complete and submit the entire standard solution as outlined above.** Do not attempt to develop the extra credit solution until you have completed the normal assignment since it will be graded separately. When you submit your solution to the 1.00 Web site, first submit your original solution. Then upload your extra credit solution as a second .zip file. Both versions should contain all the files needed to compile and run your solution.

You can get extra credit on only one homework from problem sets 8 to 10. If you don't have time do to the extra credit this time, you still have the opportunity to do so in the future.

For extra credit, your program must have two cars on the track in addition to the engine. The user must be able to couple and uncouple either of them, one at a time, with remote control. Use the rotation sensor to select which car to couple or uncouple; place it on analog port 2. Connect two LEDs to digital output ports 1 and 2, and turn the first LED on if and only if engine and the first car are coupled; turn the second LED on if and only if the two cars are coupled. When you write the events to the text file, include which car is being coupled or uncoupled.

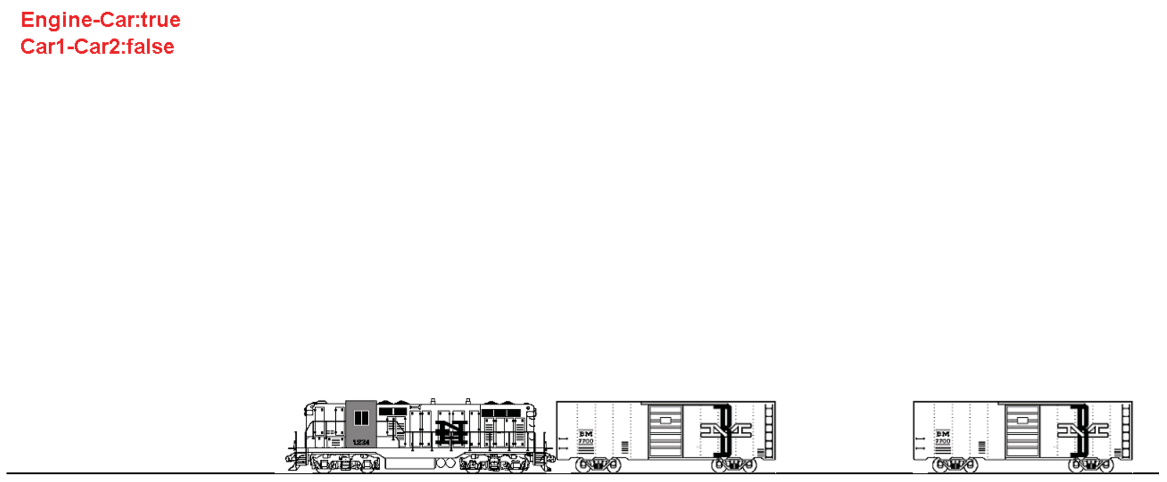
For the extra credit problem we provide images of the engine and car. See the homework 8 download zip file. Instead of drawing rectangles, you must render pictures of the engine and cars and move them around. You are welcome to use your own images.



Images courtesy of Ken Houghton Rail Images, via http://www.rrhistorical.com/

For methods of image loading and rendering, see online documentation at:
http://docs.oracle.com/javase/tutorial/2d/images/index.html

Below is a snapshot of an engine and two cars when the engine and the first car are coupled. You are welcome to add any additional cool features.



```
Engine-Car:true
Car1-Car2:false
```

Turn In

1. Place a comment with your and your partner's full names, sections, TA names and assignment number at the beginning of all `.java` files in your solution. In this homework, you will have multiple `.java` files.
2. Place all of the files in your solution in a single zip file.
   a. Do not turn in electronic or printed copies of compiled byte code (`.class` files) or backup source code (`.java~` files)
   b. Do not turn in printed copies of your solution.
3. Submit this single zip file on the 1.00 Web site under the appropriate section and problem set number. For directions see **How To: Submit Homework** on the 1.00 Web site.
4. Your solution is due at noon. Your uploaded files should have a timestamp of no later than noon on the due date.
5. After you submit your solution, please recheck that you submitted your .java file. If you submitted your .class file, you will receive **zero credit.**

Penalties

- 30 points off if you turn in your problem set after Friday noon but before noon on the following Monday. You have one no-penalty late submission per term for a turn-in after Friday noon and before Monday noon.
- No credit if you turn in your problem set after noon on the following Monday.