# CS452 – Winter 2015
# Project Milestone 2

Bill Cowan
University of Waterloo

## A. Introduction

Having completed the first project milestone you can now track a train in real-time so you know where it is at all times. This is a prerequisite for the second milestone in which two trains occupy the track at the same time, moving along planned paths that have no time/space intersections.* Three new requirements occur in this milestone.

1. You must implement collision avoidance. This requires you to define a policy that makes it impossible for two trains to occupy the same location at the same time, and to implement your policy.
2. You must implement sensor attribution. When two or more trains are moving simultaneously, each sensor report is either spurious or caused by a train. Your implementation should consistently attribute sensor reports to the correct train.
3. You must implement route finding, which provides a train with a route to a given destination.
   i. The route finder must be able to respond to changes in track configuration caused by permanent or temporary unavailability of track segments. The primary cause of unavailability is that track is use by a different train.
   ii. The route finder should find routes that require the train to change direction. There are many more routes including direction changes than there are without, which is important to you for two reasons. When you include the possibility of a direction change, you can often find a much shorter route to a destination. And with more trains on the track the increase in the set of possible routes makes you project less likely to freeze up.
4. For the demo create an application that demonstrates the success of sensor attribution in the presence of single errors, and that two trains that attempt to use the same track at the same time are prevented from doing so.

   The gold standard for the second milestone has two trains on the track. Whenever a train stops at its destination it receives or generates

* 'Time/space intersection' is my way of saying collision, which occurs when two trains try to occupy the same space at the same time.

a new destination at random. The program should runs for several minutes without breaking.

For this milestone you will give us a short demo, and hand in documentation describing the algorithms and data structures used to attain the requirements listed above.

*Comment.* This documentation will be part of your final project documentation, so doing it well will save you time in the long run.

## B. Description

### B.1. General Description

Real-time computing encounters train control when more than one train is moving on the track, and where the trains have routes that intersect by requiring the same section of track. Sharing the track is accomplished by moving the trains so that they use the section of track at different times, which is a real-time problem because arriving too early is as serious a problem as arriving too late.

In this milestone, you are required to add two features to the user interface you created for milestone one.

1. Your interface should show the current detsination of each train.
2. The user should be able to give a train a new destination, to which the train must move.

Reaching the destination should be done without colliding with the other train.

This milestone is usually accomplished by adding the three capabilities described above to your implementation.

For this milestone, your program is expected to be robust against a 'reasonable' level of errors in sensor reports and changes in the state of the turn-outs. While the train is running we will occasionally trigger a sensor or switch a turn-out to test the robustness of your tracking.

### B.2. Technical Details

Here are a few technical details, some hints and some fine points about what you are and are not allowed to do.

1. You may find that creating a track server, which maintains the current state of the track, including reservations, and which answers questions about the track from other components, is an effective way of minimizing communication overhead.
2. During the demo you are certain to be asked what parts of the track are reserved. Putting reservations into your display of the track status is a good idea, and might well reduce debugging time. Graphical displays, if you are willing to call ASCII art 'graphical', might be a good solution.
3. In specifying a destination you need not specify the time at which the train is to arrive.

4. Trivial non-collision algorithms, such as stopping one train until the other reaches its destination, are just that, trivial, and for this reason not acceptable.
5. An implementation strategy that many students have found to be effective starts conservatively, making generous track reservations. You will probably notice that the more conservative the reservation system, the more time the trains spend motionless. When you think things are robust you can then make your reservations more aggressive and get better performance.
6. Your route finding algorithm should take into account the current state of the track. That is, it should be able to route trains around malfunctioning switches and unreservable sections of track.
7. Making coordinated routes is hard. It is probably good to start by finding routes that treat track obstacles, including other trains, as static, proving routes that avoid where they are now, with new routes being provided when trains discover that they are unable to complete the route that was given previously.

The above list may grow as students ask me questions.

## c. Hand in

Hand in the following, nicely formatted and printed.

1. A description of how to operate your program, including the full pathname of your executable file which we may download for testing.
2. A description of the algorithms and data structures used to reserve sections of track, to find routes and to break deadlocks. Remember to tell us how they work and why you chose them. We will judge your program on the basis of this description and on the demo.
3. The location of all source code you created for the assignment and a set of MD5 hashes of each file. The code must remain unmodified after submission until the assignments are returned.
4. A listing of all files submitted.