CCR EXCEPTIONS

Jack Ward

May 12, 2016

Version 1

**********Table of Contents**********

This file contains a list of exceptions my program has, along with a possible solution for each.

****

Exception 1, Number of Trains

1) There are only four trains.
2) There is no way to remove one train at a time. They must all be removed at once.
3) The trains cannot be named by the user.

Solution: Make a modeless form that has the train controls. As each train is created, open a new form associated with the train. Disable the exit buttons on them, and exit them as trains are removed.

Exception 2 No Click to Add

There is no "click to add" interface for the trains.

Solution: Implement the same method used for the track in the track editor (Form1).

Exception 3 No Sensors

No sensors were implemented.

Solution:

1) Add a Boolean property bool has_sensor to Square.cs
2) Add a checkbox to the Toolbox in Edit mode (form1)
    a. If checked, set has_sensor to true
3) Write a function in Run.cs that detects when a train is on a Square with a sensor

Exception 4 No Communication Between Applications

No Communication with external applications

Solution

1) Write to a file for another application to read whenever any of the following happen:
    a. The user clicks the "Ready" button in run mode (Send "Ready" Message, and info about the track and a list of all the trains)
    b. A Turnout is changed (Send location and in/outs)
    c. Trains collide (Send location and train Id's)
    d. Trains are derailed (Send location and train Id)
    e. Trains are added or removed (Send train removed, and possibly the new list of trains)
    f. Speed of a train is changed (Send the id of the train and the new speed)
    g. Direction of a train is changed (Send the train Id and the new Direction)
    h. The Go or Stop buttons in the train controls are pressed (Send the train Id, direction, speed, and any other properties the train has.)
    i. A train trips a sensor – (Send the train Id, and the sensor id/location)

   The other program can then make decisions based on the information it received and write commands to another file.
2) Read from a file sent by another application every time the train timer ticks. Reads commands in similar to the following
    a. Has the file been modified? AKA do we have any new commands? (Indicated by a number at the top of the file)
    b. Change Speed (Read in Train Id and new speed)
    c. Stop/Go (Read in Train Id)
    d. Change Turnout (Read in coordinates)
    e. Change Direction (Read in Train Id and new direction)

   Then the CCR Simulator would force clicks and changed the values of the appropriate controls to account for the changes desired by the external application.

****

Exceptions Involving Bugs

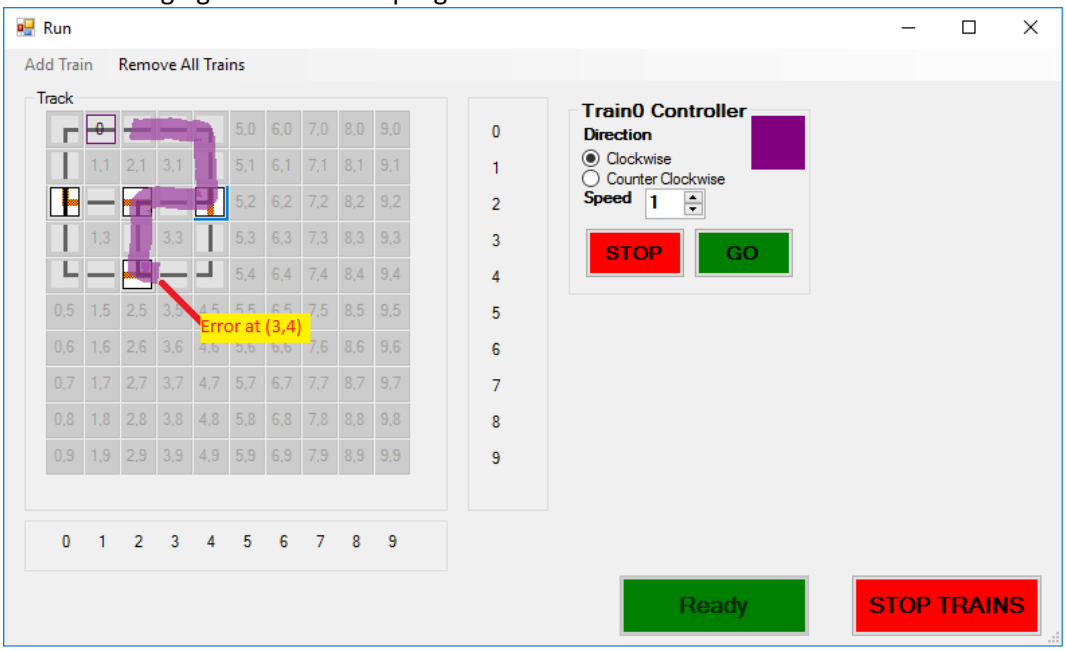
****

The following is a list of test cases that don't work, and possible solutions for them.

Bug 1

Double Reverse Loop

The double reverse loop occurs when it is possible to go two directions on the same section of track without changing direction. This program will cause the train to be derailed on a double reverse loop.



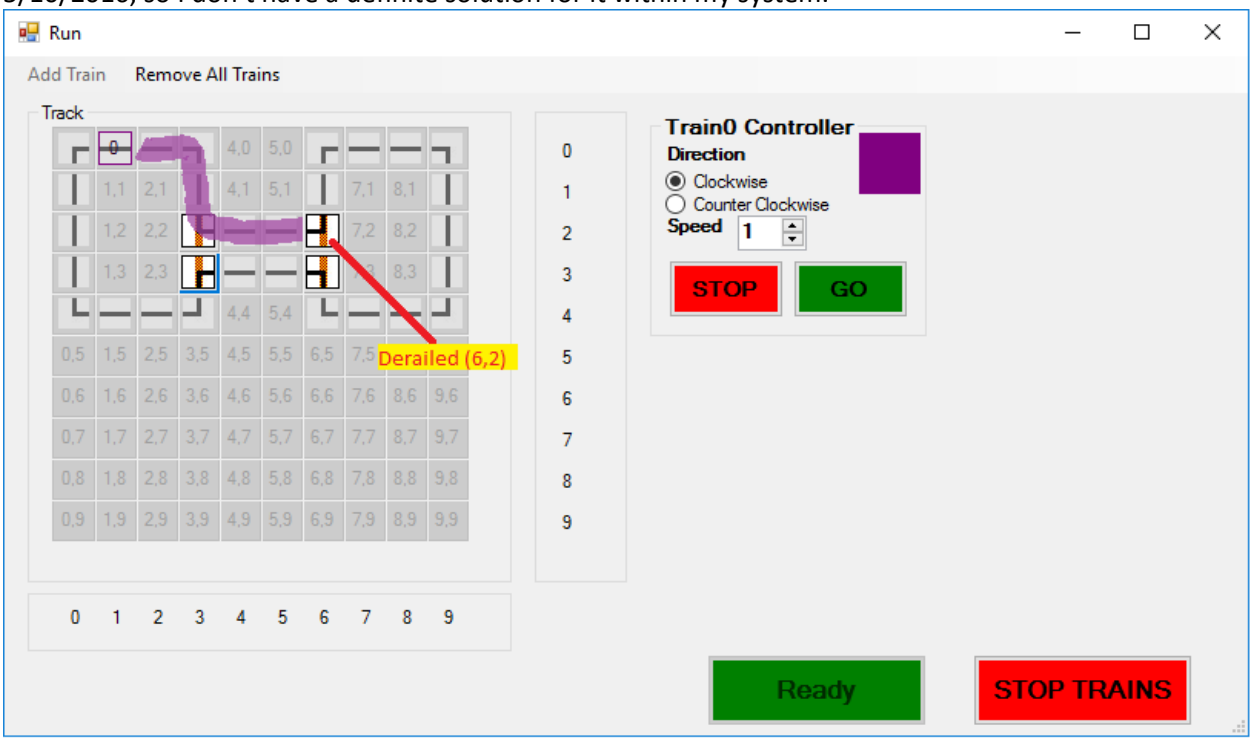
Train 0 will be derailed at position (3,4)

Bug 1 Solution: None. The user must be aware of this, and be careful not to make a double reverse loop

Bug 2

Connecting Two Tracks

This isn't a double reverse loop. The error occurs because the in for the Square at (6,2) expects an in from above, and the Square at (6,1) has an out on the right. This is a pretty recent error Discovered 5/10/2016, so I don't have a definite solution for it within my system.



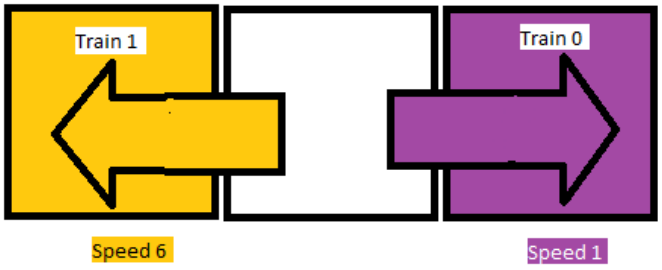
Bug 2 Solution: (See %% for a new theoretical system to be implemented in Version 2)

Bug 3

Non Crashing Trains

If Train 0 and Train 1 are headed towards each other, and Train 0 has a speed of 6 and Train 1 has a speed of 1, they will almost always pass over each other. This is a possibility for any trains going towards each other at different speeds (other than 0 of course). The problem is, before collision is checked, both trains move, and pass over each other. In a sense, they go right through each other.

Bug 3 Solution: When I check to see if the trains have collided, I would have to check to see if the trains in question were going in opposite directions, and if so, I would have to check if they were right next to each other going away from each other. If so, then they had crashed.



Bug 4

No Save on New, Open, Or Exit

When the user exits the program, opens a new file, or creates a new file, if the file that is currently open has been modified, or has never been saved, the application should really ask the user if they would like to save first.

Bug 4 Solution: This is an easy fix. I would just need to mark the document as edited every time it changes. Then, when the user hits New, Open, or Exit, I would first check to see if the current document had been changed before continuing, and if it did, I would ask the user if they would like to save. If they said yes, I would run the SaveAs function, before continuing on.

Bug 5

Non-Cyclical Track

This error is particularly bad, because it crashes my program. If, at any point in the track, there is a piece doesn't meet up with another piece, then the program will crash. An example of this would be a track that is a straight line.

Bug 5 Solution (A): Use a try/catch when defining a track. If there is an unmatched piece anywhere in the track, throw up an error message and quit.

Bug 5 Solution (B): (See %% for a new theoretical system to be implemented in Version 2)

****
**********Theoretical Solution**********

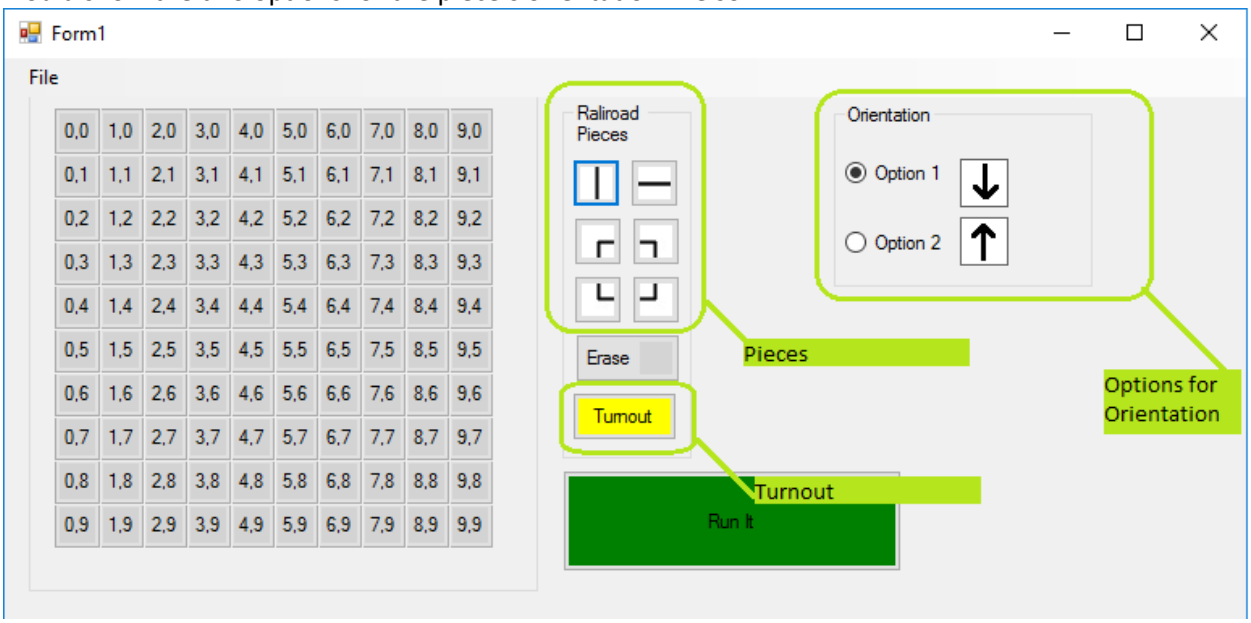
In this version of the CCR, a train runs based on the in/out value of the current piece it is on. It reads a piece's out, and goes in that direction. The program defines the ins/outs of the track based on the locations of the trains and the directions in which they are going. This makes for a very lengthy and complicated function that produces some bugs.

Balancing user input and calculations from the program is difficult. As a programmer, you don't want to put too much in the hands of the user, because the program gets complicated. On the other hand, it can sometimes be a lot easier for a user to enter data in than it is for the program to calculate it.

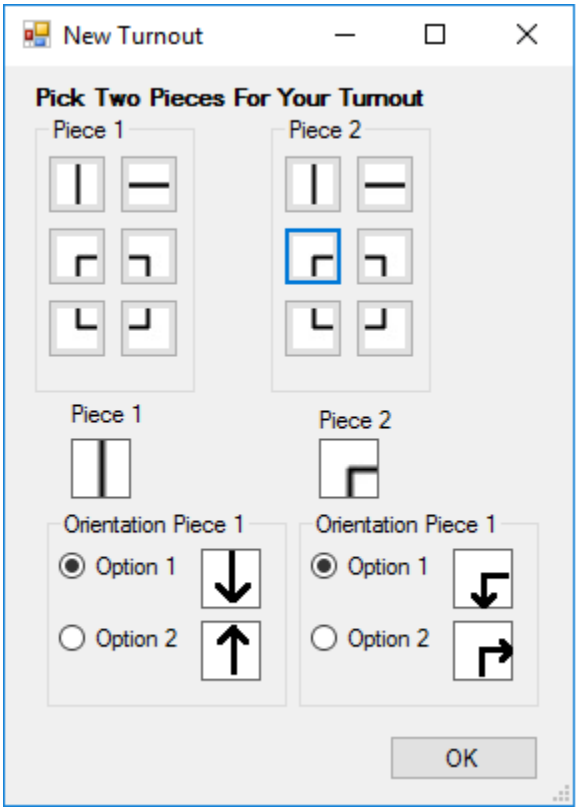
In this case, it would have been much easier for the user to specify the ins/outs of each track, than for me, as the programmer, to develop a system in which the ins/outs of a piece are calculated. Giving the responsibility of declaring ins and outs to the user opens up a wide margin of possibility for human error, but if a program can't do something as accurately as a user, then it is, perhaps, better to leave the task in the hands of the user.

The Idea

When the user clicks on a tool on the Toolbox (Railroad Pieces) if it's not an eraser or a turnout, a box would show the two options for the piece's orientation like so.



When a user selects the yellow Turnout tool, the turnout selector window would show the two possible orientations for both pieces selected like so.


New Turnout
Pick Two Pieces For Your Turnout
Piece 1
Piece 2
Piece 1
Piece 2
Orientation Piece 1
Option 1
Option 2
Orientation Piece 1
Option 1
Option 2
OK