TMM4275 Knowledge Based Engineering, project

Hallvard Bjørgen, Johanne Glende, Sigve Sjøvold

Assignment 1: Barn Rail System

TMM4275 - Automatisering, prosjekt - Assignment 1
Hallvard Bjørgen, Johanne Glende and Sigve Sjøvold Deadline: 25.02

Product description and selected parameters.

The rail system product is an automated system that generates a rail system for the TKS feeding cart, using the following parameters:

- Number of visit areas: Indicates how many points the rail system will visit on its path. If the end user selects 10 visit areas, they will have to input the x- and y-coordinates of the 10 points.
- Length: Length of the barn
- Width: Width of the barn
- Height: Indicates the height at which the rail system will be installed.

A solution will be generated when the end user inputs their parameters on the online web server, which is linked to the system.

Unfortunately, our system is not fully functional in its given state, and the end user will not be able to successfully use the application.

Website:

For a more visual understanding of our website, please see *UI Sequence Diagram* in the UML-diagrams paragraph. The first and second page of the website is for the customer to fill in wanted visit areas and parameters for his or her barn.

A 3D-model is generated to illustrate the solution using Siemens NX. The model will be illustrated on the website so the end user can see how the rail system would look with their given input.

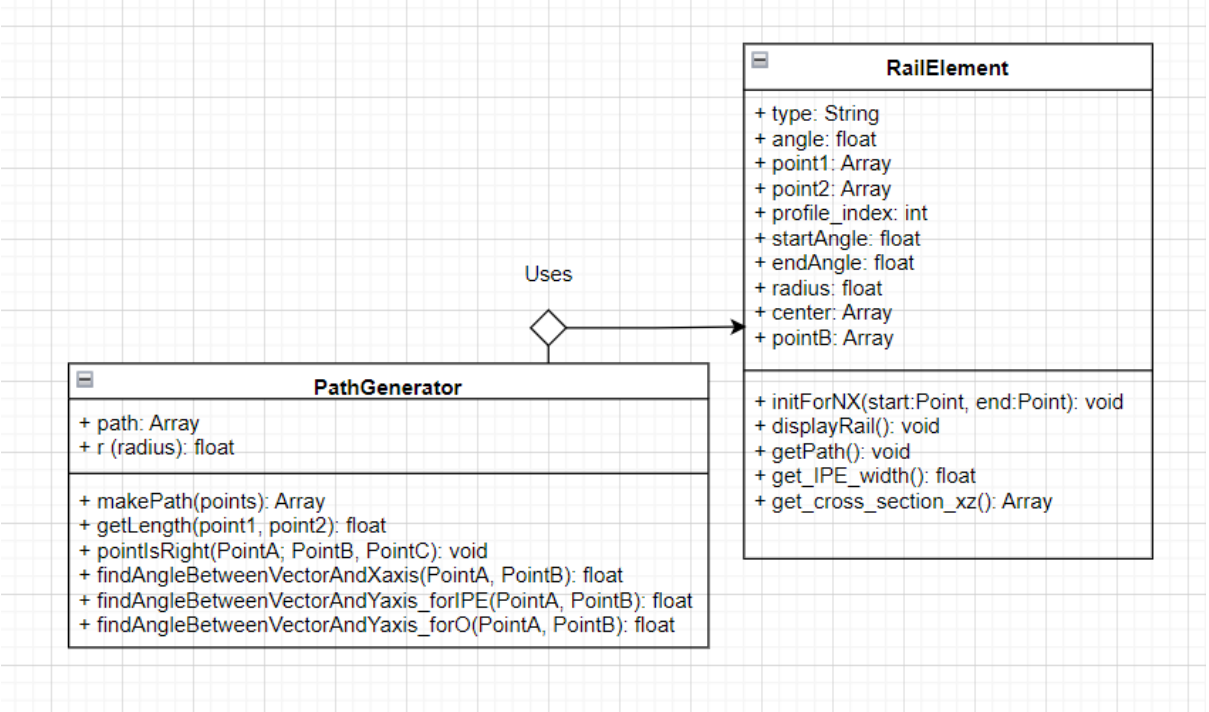
The page “See/modify data” is to give customers a chance to edit the parameters they gave if this is needed. If a customer confirms the solution and then wants to “go back” the solutions and values are lost and he or she will have to start from the top again.

KBE application architecture (main blocks and interconnections between those described). + UML class diagrams.

What we wanted to implement:

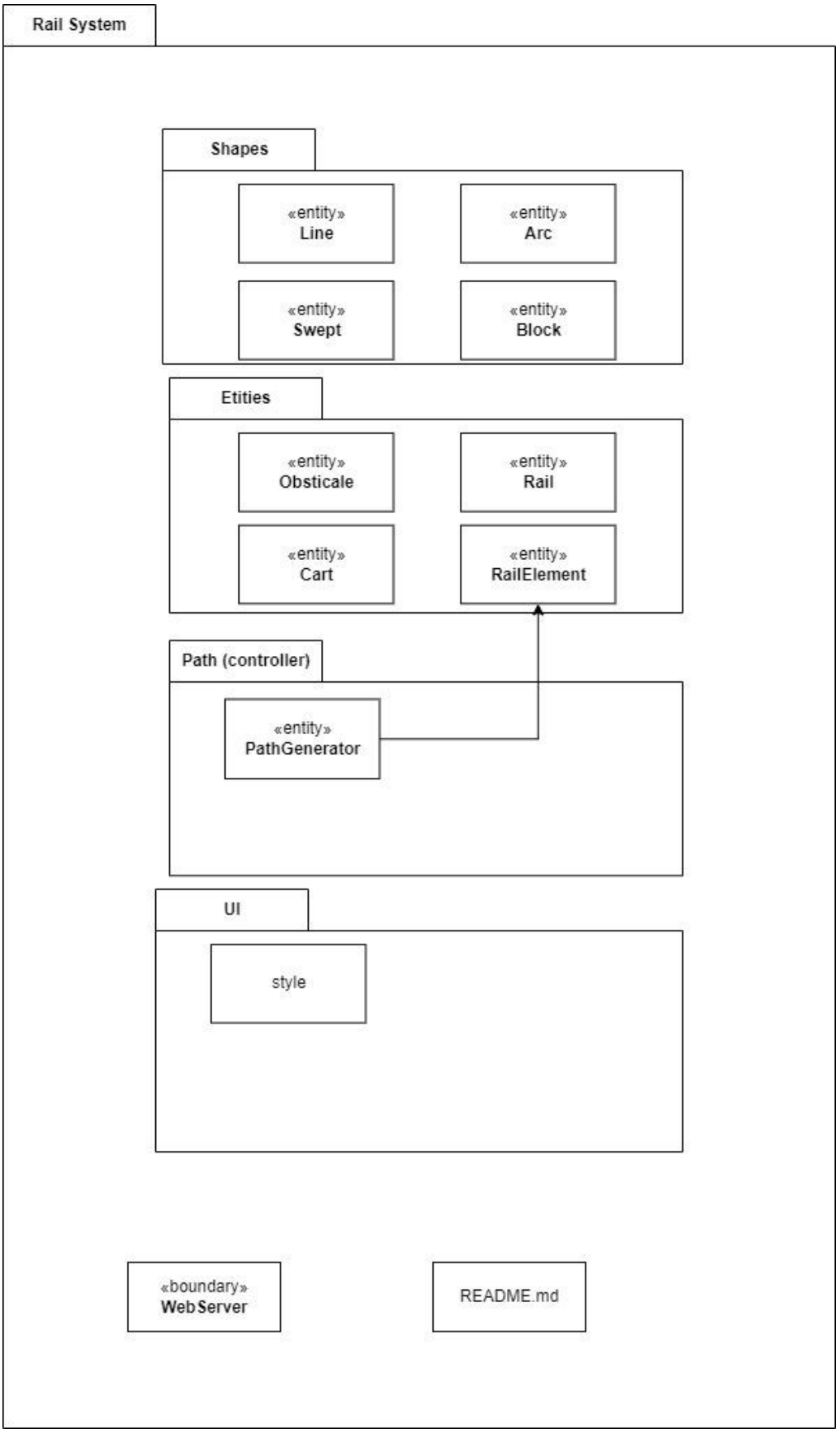
RailElement should contain a method for making a path that alignes with either an arc or a line. The displayRail method should rather be in Rail. The PathGenerator should make a path consisting of multiple RailElement paths, and then be able to use a displayRail call using a cross section from a singly expressed IPE-profile (not that we express it for every rail element), and using a height which also will be expressed just once. The Rail should be able to take a path (composed from the PathGenerator module), and decompose this into several smaller parts of a maximum length. The WebServer should host a (today) local web site, making it possible to plot in relevant information: barn height, width, length, obstacles and visit areas for the feeder. This information should then be sent in a “pipe-and-filter” pattern type of way through to PathGenerator, Rail, and then become RailElements. The RailElements should then, thanks to Rail, be of a certain max length.

Different UML diagrams:



Class diagram

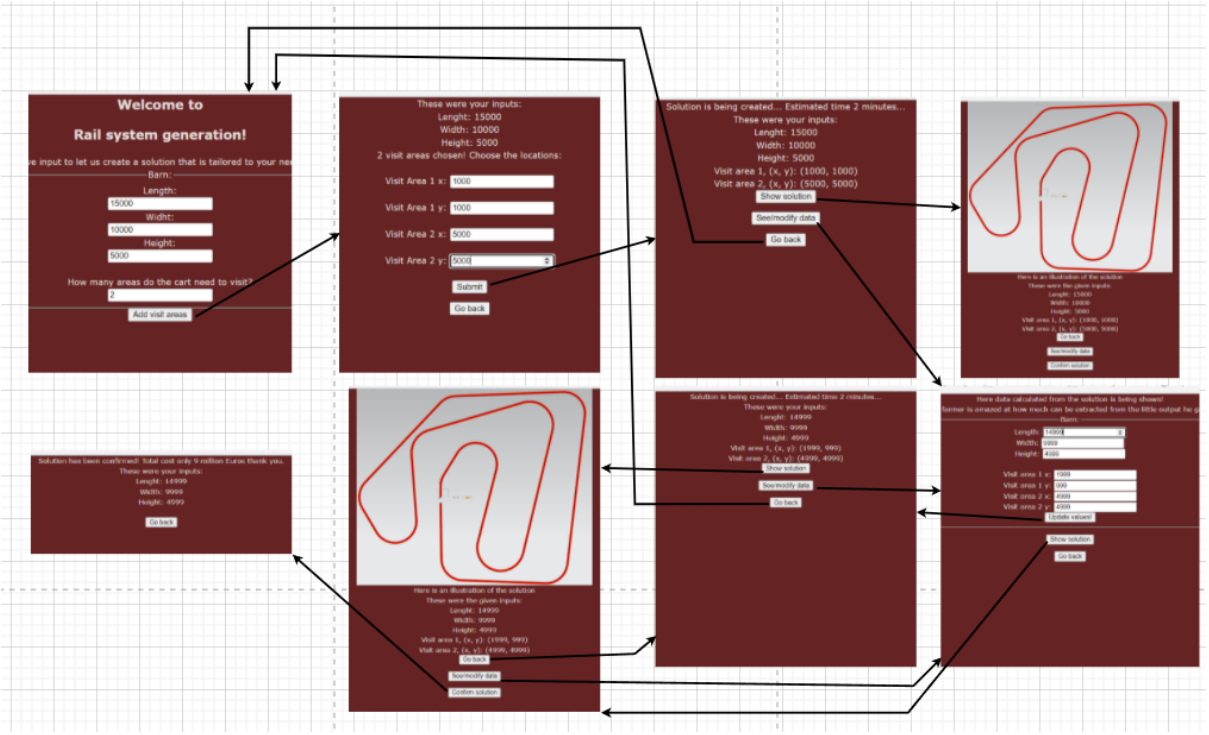
Class diagram as the application is now. The class diagram would be more complex if we succeeded with our original plan. For example classes like Obstacle and Rail would be present



Package diagram

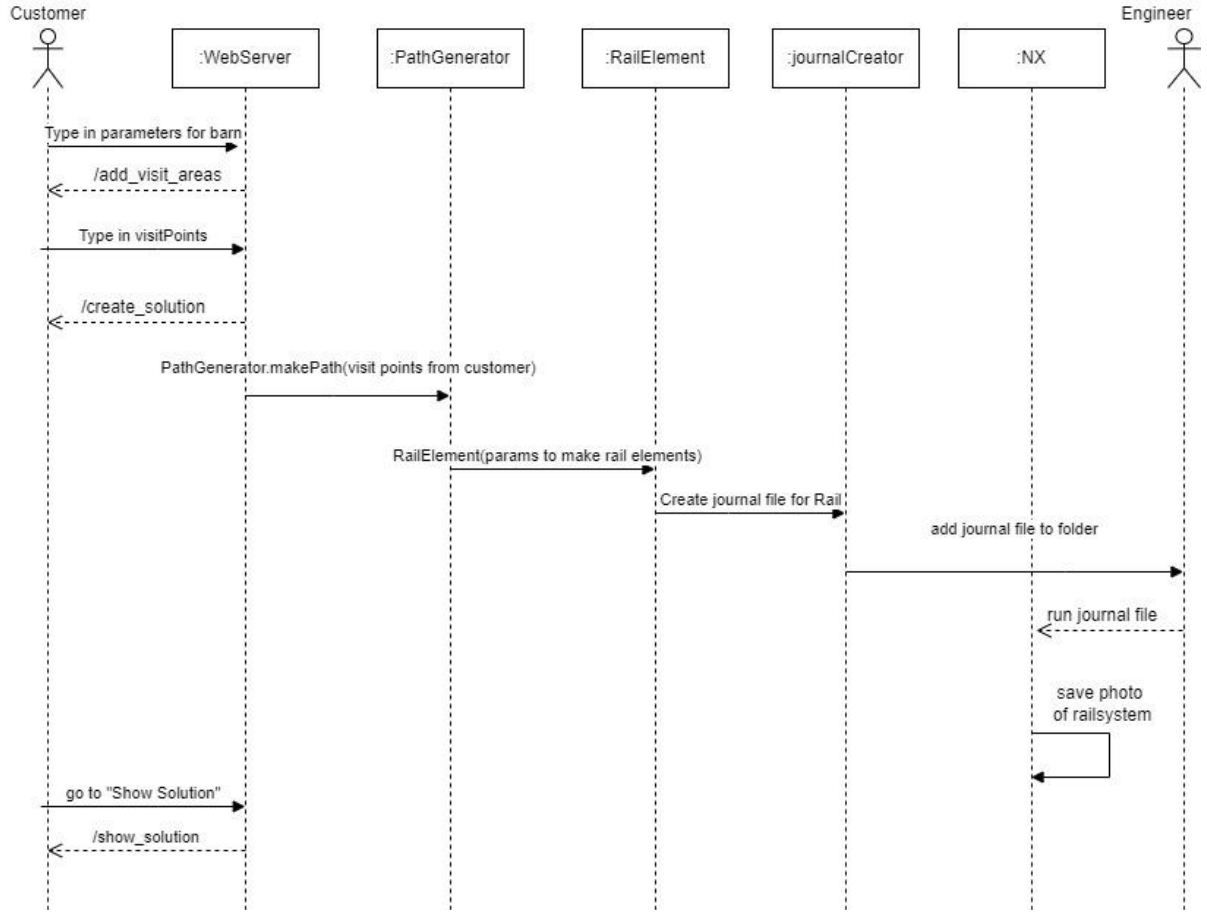
TMM4275 - Automatisering, prosjekt - Assignment 1
Hallvard Bjørgen, Johanne Glende and Sigve Sjøvold Deadline: 25.02

UML sequence diagrams showing an order making scenario (how the order is placed)



UI Sequence Diagram

UI for a retrieving customer. Contact information has not yet been implemented, but would be added for the final product after “Confirming solution”.



Sequence Diagram

A diagram showing how the solution should have worked, and how we would have implemented it if we were to work more with it.

Description of the code you have developed - module names, their purpose and main functions.

Our code consists of the following modules:

RailElement.py:

- Contains the RailElement class which creates the IPE-profile for the rail elements.
- Displays and extrudes the rail elements as 3D-models in Siemens NX.
- Creates rail elements either as an arc, or as a line.

PathGenerator.py:

- Uses an array of points to put together several rail elements, to create a path.
- Calculates the arcs position relative to the given lines.
- Calculates the angles for the arcs start- and endpoint.

WebServer.py:

- Hosts a local server and creates the user interface. Ideally this module would communicate with the other modules.

Cart.py:

- Currently just a block used to illustrate the TKS cart. This would be needed for a journal module.

Obstical.py:

- The idea for this module was to create an object located in the barn that the rail system would have to avoid. Obstacles are not implemented, but it is still possible to avoid obstacles with this solution by adding additional points (visit areas) that navigate around the object.

Arc.py:

- A python class for generating arcs in NX.

Line.py:

- A python class for generating lines in NX.

Swept.py:

- Receives lines and/or arcs and follows along their path.

Main challenges:

We struggled quite a lot completing this assignment, and even though we really wanted to succeed, we fell short. The biggest problem we faced was correctly calculating the angles needed to precisely align the arc to a line. We tried several methods, but none of them seemed to work as we wanted them to. We needed to identify the start angle and end angle for the arc in all the different scenarios. Initially we thought specifications for which of the four global quadrants the arc existed in was enough, but in reality the problem was much more complex than that, and we started off on the wrong foot. In this, we also needed to take which direction the arc would turn into account, and in doing this, we were even more confused by our angle expression. Our problems with the angles meant we also struggled to correctly identify point O for every possible outcome. This in turn leads to problems locating point B'.

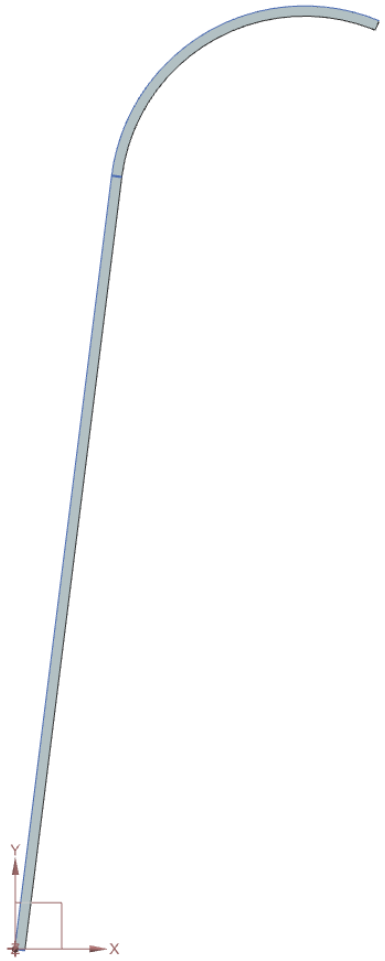
We surely underestimated how much work this project was to be, and if we were to start over again we would thoroughly calculate all of the angles and points before implementing. In this way we would not face as many difficulties as we did this time, hopefully.

In short, our main challenges was:

- figuring out how to decide if the arc is to turn right or left
- the position of O, and with that B' was difficult to calculate, and up to this point only found for a right turn. The next step is to "flip" the code so it is working for a left turn as well.
- The start angle and end angle for where the arc was to go was a difficult task as it depended on a lot more than we at first thought.

Results: 3 examples of the product ordered with different and arbitrary values of the parameters.

Example 1



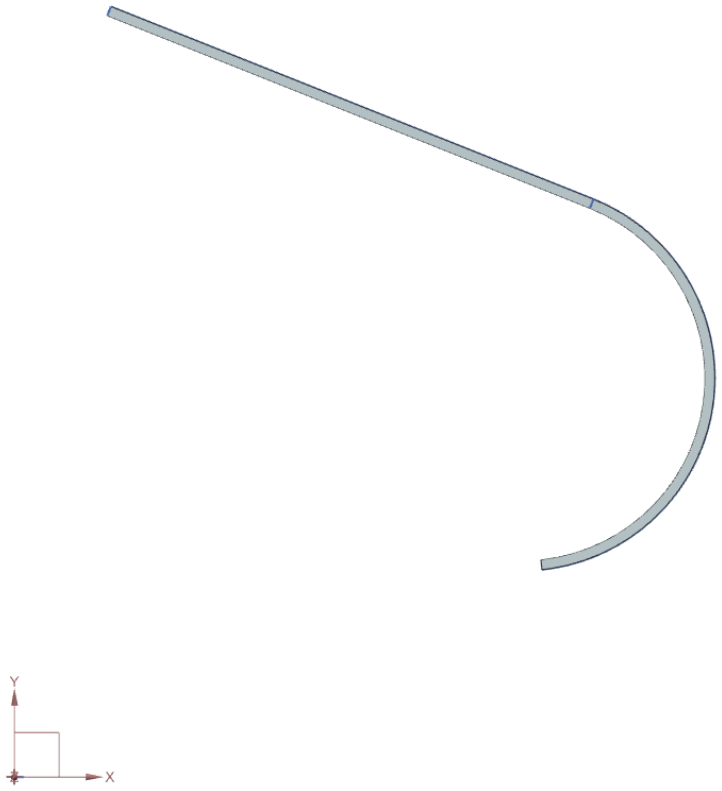
With points:

A = [0, 0, 0], B = [1000, 8000, 0], C = [10000, 7000, 0]

call to PathGenerator: makePath([[0, 0, 0], [1000, 8000, 0], [10000, 7000, 0]])

Rail to point C is not drawn until a fourth point is added to the makePath() function.

Example 2

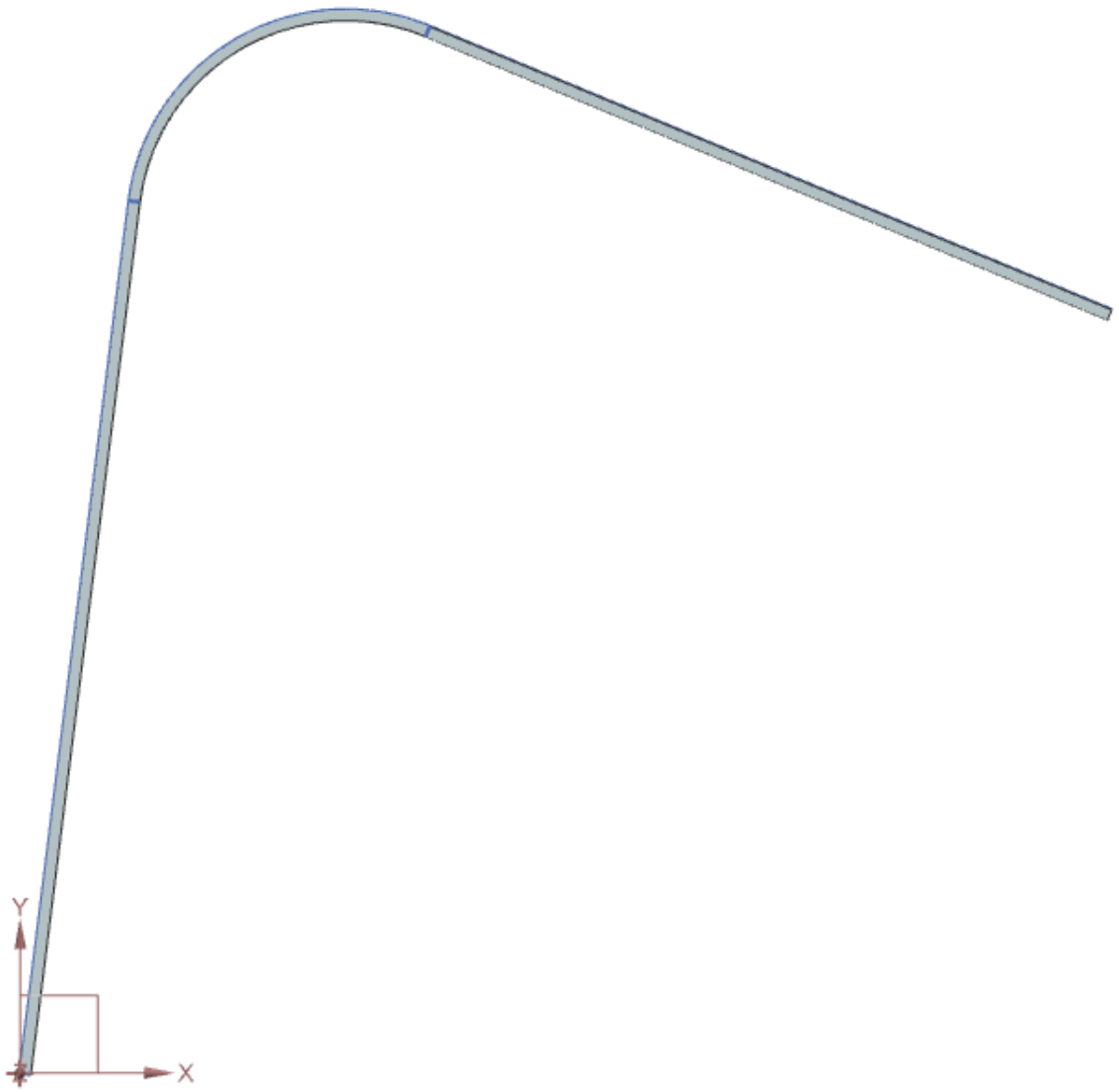


call: makePath([[1000, 8000, 0], [6000, 6000, 0], [4000,2000,0]])

A = [1000, 8000, 0], B = [6000, 6000, 0], C = [4000, 2000, 0]

Rail to point C is not drawn until a fourth point is added to the makePath() function.

Example 3:



call to PathGenerator: makePath([[0, 0, 0], [1000, 8000, 0], [10000, 7000, 0], [8000,2000,0]])

A = [0, 0, 0], B = [1000, 8000, 0], C = [10000, 7000, 0], D = [8000, 2000, 0]

Rail to Point C is drawn due to the fourth point in makePath().