Design Document

Team 4A

Purpose:

The *Track Display Creator* is a QT program written in C++ that allows a user to create a graphical representation of a train track.

Reference Material:

Repository: *https*://github.com/ctag/cpe453

Sprint Tracker: https://trello.com/b/8oFdolpq/senior-design-studio

QT information: http://doc.qt.io/

System Overview:

The user should be able to create a track system (which includes nodes, track pieces, switches, and track sections) that could be uploaded to a SQL table. The table would provide detailed information about connectivity between nodes, information on the switches (pass, bypass), as well as what section each node is part of.

Procedure:

QT Designer was used to create the skeleton interface for the project.

A QGraphicsView was sub-classed (track.cpp) within the mainwindow.cpp to allow overriding of the class's functions. (http://doc.qt.io/qt-4.8/designer-using-custom-widgets.html)

Inside the track.cpp constructor, a QGraphicsScene was created, which would be the location for all the objects to be placed.

Core Features:

Vertices:

Affiliated User Stories:

- As a Developer, I want to t investigate the tools needed for painting the track display.

- As a user, I want to be able to create a detection node(vertex)

- As a user, I want to be able to move objects around

- As a user, I want to be able to create a switch

Vertices were created by overriding the *mousePressEvent() (line 68)* of track.cpp. If the right mouse button was pressed a custom QGraphicsItem (vertex.cpp) was created and added to the scene.

-vertex.cpp/vertex.h: custom QGraphicsItem class used to create node and switches.

-When the right mouse button was pressed and no object was beneath the cursor, it would initialize a new *vertex* with arguments being the event position on the scene, and a unique ID.

-each vertex had a specified “type” that was used to set or determine whether or not the vertex was a node or a switch. (*line 40-60)*

-each vertex had a label (text.cpp) that would appear above the node which it’s ID and type and follow the vertex’s position

-text.cpp/text.h: custom QGraphicsItem

-Overriding the *paint() (line 86,vertex.cpp)* event of vertex class allowed us to design the appearance of our vertex to distinguish between selected/unselected and node/switch.

Edges:

Affiliated user stories:

- As a user, I want there to be a distinction between mainpass and bypass of a switch

Edges were created once two nodes had been selected and the connect button on the interface had been clicked.

-*(line 163, track.cpp)* connect_button_clicked(): Checked the two vertices to ensure neither of them were switches. The function created a solid line between the two if they were nodes, else if the connection was a switch, it created a dash-line (main pass), or a dotted-line (bypass). Each time a line was created, a variable inside each vertex was set to point at the corresponding edge.

Sections:

Affiliated user stories:

- As a user I want to be able to create detection sections for the track

This feature was to select two vertices and then press “New Group” button to assign a string of format 1-1, 1-2, and so on.

- (line 502,track.cpp) do_assignDS()

Delete:

Affiliated user stories:

- As a user I want to be able to remove items from the track

For both edges and vertices a deletion function was applied.

- (line 327, track.cpp) deleteSelected(): Grabbed all selected items and removed them from the scene as well as from the QLists used to store all edges and vertices.

SQL:

Affiliated user stories:

- As a user, I want to be able to upload the track to a SQL table.

An SQL tab was implemented on the interface which would allow the user to connect to the database.

- Sql.cpp/sql.h: custom class used for the SQL portion of this project.

Additional Features:

- The user can move the whole track display up/down/left/right. (line 262-line 320)

- Edges track vertex’s location and move accordingly (line 122 – 143)